Федеральное государственное автономное образовательное учреждение
высшего образования
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(национальный исследовательский университет)»

На правах рукописи

КРАЕВА Яна Александровна

МАСШТАБИРУЕМЫЕ МЕТОДЫ И АЛГОРИТМЫ ПОИСКА АНОМАЛИЙ ВО ВРЕМЕННЫХ РЯДАХ

Специальность 2.3.5. Математическое и программное обеспечение
вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:

ЦЫМБЛЕР Михаил Леонидович,
доктор физ.-мат. наук, доцент

Челябинск — 2023

Оглавление

Введение

Актуальность темы исследования

Актуальность темы диссертационного исследования основывается на следующих основных факторах:

1) всепроникающий характер временных рядов в современном информационном обществе и востребованность задачи поиска в них аномалий;

2) наличие больших временных рядов в широком спектре приложений;

3) массовое распространение кластерных вычислительных систем с графическими процессорами.

Рассмотрим указанные факторы более подробно.

Всепроникающий характер временных рядов. В современном информационном обществе задачи анализа данных в виде временных рядов возникают в широком спектре предметных областей: финансовое прогнозирование [144], персональное здравоохранение [130], спорт высших достижений [71], прогноз погоды и моделирование климата [89], технологии Интернета вещей [79], цифровое производство [14], интеллектуальное управление зданиями [150], умные города [62], предсказание природных катаклизмов [91], мониторинг производительности суперкомпьютерных систем [129] и др. Временной ряд представляет собой хронологическую последовательность вещественных значений, где временны́е метки элементов ряда существенным образом определяют способ постановки и выбор методов решения задач анализа указанных данных [40, 44].

Востребованность задачи поиска аномалий. Поиск аномалий является одной из основных задач анализа временных рядов [20, 40, 44, 55, 116]. За последние пять лет востребованность научных исследований, посвященных поиску аномалий во временных рядах, существенно повысилась по сравнению с темами поиска аномалий в других видах данных — изображения,

тексты, видео и др. [26]. Важным и актуальным аспектом проблемы поиска аномалий во временных рядах является случай потокового временного ряда, элементы которого поступают в систему для обработки один за другим в режиме реального времени [12]. В системах подобного рода поиск аномалий должен выполняться за определенный в данной предметной области конечный период времени, чтобы поддерживать постоянное и своевременное взаимодействие со средой [11].

Большие временные ряды. В современных приложениях, связанных с обработкой и интеллектуальным анализом временных рядов, типичной является ситуация, когда анализу подлежат большие временные ряды (time series big data), которые не могут быть целиком размещены в оперативной памяти и требуют специализированных методов обработки и визуализации [12]. Приведем примеры приложений, в которых осуществляются сбор и обработка больших временных рядов. Система контроля дорожного движения в Мадриде [80] насчитывает более 3500 автоматических регистраторов транспортных средств, которые собирают показания каждые 15 мин., начиная с 2014 г. Океанографическая станция MAREL Carnot [66] с 2004 г. каждые 20 мин. регистрирует более чем 15 химических и биологических характеристик воды в проливе Ла-Манш. Временные ряды набора данных DEBS challenge [100] представляют собой показания сенсоров пространственного позиционирования с частотой 200 Гц, которые были закреплены на бутсах игроков и перчатках вратаря футбольных команд, а также на мяче (с частотой 2000 Гц), и во время реального футбольного матча продуцировали данные о 15 000 событий в секунду. Одним из наиболее важных классов современных приложений, в которых требуется поиск аномалий в больших временных рядах, являются приложения Интернета вещей и цифрового производства, в которых датчики киберфизических систем имеют высокую дискретность снятия показаний (десятки–сотни раз в секунду) и за короткое время продуцируют временные ряды, состоящие из сотен миллионов элементов [14, 79]. В указанных приложениях своевременное обнаружение аномалий позволяет заблаговременно уведомлять оператора

технологического процесса и организовать предиктивное техническое обслуживание оборудования. Эффективный поиск аномалий в больших временных рядах фактически обуславливает необходимость применения для решения данной задачи параллельных алгоритмов и высокопроизводительных вычислений.

Массовое распространение многоядерных вычислителей и построение высокопроизводительных кластеров на их основе. В настоящее время увеличение количества вычислительных ядер является одной из основных тенденций развития процессорной техники [96, 137], и производители выпускают соответствующие устройства, содержащие до десятков тысяч ядер, поддерживающих векторную обработку данных. Одной из наиболее распространенных многоядерных архитектур являются графические процессоры (GPU, graphics processing unit) производства компании NVIDIA [69]. Графические процессоры хорошо подходят для организации вычислений в соответствии с парадигмой SIMD (Single Instruction for Multiple Data, одна инструкция на много данных), поскольку они состоят из симметричных потоковых мультипроцессоров, каждый из которых, в свою очередь, состоит из симметричных ядер CUDA (Compute Unified Device Architecture). Интерфейс прикладного программирования CUDA позволяет назначить нескольким потокам выполнение одного и того же набора инструкций над несколькими данными. В силу указанных особенностей в настоящее время графические процессоры стали де-факто стандартной платформой для обучения глубоких нейронных сетей [137]. Кроме того, одной из современных тенденций организации высокопроизводительных вычислений является объединение нескольких вычислительных узлов посредством высокоскоростной соединительной сети в один вычислительный комплекс, называемый кластером. В качестве узла такого кластера нередко фигурирует центральный процессор, дополненный графическим процессором. Например, большинство систем, входящих в рейтинг Топ50 наиболее производительных вычислительных систем России (редакция № 38 от 28.03.2023),

являются высокопроизводительными кластерами с узлами на базе графических процессоров [128].

Исходя из рассмотренных выше факторов, можно заключить, что *актуальным* является исследование, направленное на разработку методов и алгоритмов поиска аномалий во временных рядах на платформе современных высокопроизводительных вычислительных систем: массово распространенных графических процессоров и кластеров с узлами на их основе.

Степень разработанности темы

В 1999–2002 гг. Коудас (Koudas) и др. [63], Кнорр (Knorr) и др. [70], Рамасвами (Ramaswamy) и др. [110] и Ангиулли (Angiulli) и др. [13] предприняли попытки формализовать понятие аномальной подпоследовательности и разработали соответствующие алгоритмы поиска аномалий. Большой вклад в теорию интеллектуального анализа временных рядов в целом и в тему поиска аномалий временного ряда в частности внес американский ученый И. Кеог (E. Keogh). Под его руководством в 2005 г. разработаны концепция диссонанса временного ряда и алгоритм HOTSAX [86] для поиска диссонансов ряда, который может быть целиком размещен в оперативной памяти. В отличие от предшественников, диссонанс имеет интуитивно более понятное определение и более точно находит аномалии [86]. Позднее, в 2007 г. Кеог и др. представили алгоритм DRAG [138] для поиска диссонансов временного ряда, который не может быть целиком размещен в оперативной памяти. Далее, в 2020 г. Кеог и др. предложили алгоритм MERLIN [101] и в 2023 г. его улучшение — алгоритм MERLIN++ [102], которые выполняют поиск диссонансов временного ряда произвольной длины. В 2023 г. Кеогом и др. предложен алгоритм DAMP [92] для обнаружения аномалий в потоковых временных рядах. Однако алгоритмы HOTSAX, DRAG, MERLIN и MERLIN++, DAMP являются последовательными. Кроме того, в 2016 г. Кеог предложил концепцию матричного профиля временного ряда [141], которая может быть эффективно применена для поиска различного рода шаблонов во временных рядах (смысловые моти-

вы [58], сниппеты [59], цепочки [145] и др.). Диссонансы могут быть тривиально найдены как побочный результат вычисления матричного профиля. Однако в силу квадратичной вычислительной сложности относительно длины ряда [142, 146] решение задачи поиска диссонансов произвольной длины, даже с помощью параллельных алгоритмов вычисления матричного профиля на графическом процессоре и кластерной системе (например, GPU-STAMP [142]) становится невозможным за приемлемое время. Значимый вклад в тему обнаружения аномальных подпоследовательностей временного ряда внес коллектив, в который входят французские ученые П. Бониоль (P. Boniol), Т. Палпанас (T. Palpanas), М. Линарди (M. Linardi) и американский ученый Дж. Папарризос (J. Paparrizos). Ими в 2020–2023 гг. разработаны последовательные алгоритмы Series2Graph [24], GraphAn [25], SAD [22], NorM [21], NormA [23] поиска аномалий временного ряда, а также оболочка TSB-UAD [106] для оценки качества методов обнаружения аномальных подпоследовательностей. М.Л. Цымблер и др. разработали параллельные алгоритмы поиска диссонансов [9, 148, 151], однако эти разработки не решают задачу поиска диссонансов произвольной длины и не предназначены для высокопроизводительных вычислительных кластеров с графическими процессорами.

Исходя из рассмотренного выше, можно заключить, что на сегодняшний день задача разработки эффективных методов и алгоритмов поиска аномалий во временных рядах остается в фокусе интенсивных научных исследований и практических разработок.

Цель и задачи исследования

Цель данной работы состояла в разработке и исследовании новых методов и алгоритмов поиска аномальных подпоследовательностей временного ряда, основанных на концепции диссонанса, на платформе современных высокопроизводительных вычислительных систем.

Данная цель предполагает решение следующих *задач*.

1. Разработать и исследовать следующие параллельные алгоритмы поиска диссонансов временного ряда:
   - поиск диссонансов фиксированной длины на графическом процессоре;
   - поиск диссонансов произвольной длины на графическом процессоре;
   - поиск диссонансов произвольной длины на высокопроизводительном вычислительном кластере с графическими процессорами.
2. Разработать нейросетевую модель для поиска аномалий в потоковом временном ряде на основе концепции диссонанса.
3. Провести вычислительные эксперименты с синтетическими и реальными временными рядами, подтверждающие эффективность предложенных методов и алгоритмов.

Научная новизна

Научная новизна проведенного диссертационного исследования заключается в следующем.

1. Разработан новый параллельный алгоритм поиска диссонансов временного ряда, имеющих фиксированную длину, для графического процессора. В отличие от известных аналогов, алгоритм позволяет находить во временном ряде все диссонансы, которые имеют заданную длину.
2. Впервые разработан параллельный алгоритм поиска диссонансов временного ряда, имеющих произвольную длину, для графического процессора. Алгоритм позволяет находить во временном ряде все диссонансы, которые имеют длину в заданном диапазоне.
3. Впервые разработан параллельный алгоритм поиска диссонансов временного ряда, имеющих произвольную длину, для высокопроизводительного вычислительного кластера с графическими процессорами. Алгоритм позволяет находить во временном ряде, который не может быть

целиком размещен в оперативной памяти вычислительного узла, все диссонансы, имеющие длину в заданном диапазоне.

4. Разработан новый метод поиска аномалий в потоковом временном ряде. В отличие от известных аналогов, метод позволяет выявлять аномальные подпоследовательности ряда, отражающие нетипичную и редко встречающуюся активности исследуемого субъекта.

Теоретическая и практическая значимость работы

Теоретическая ценность диссертационной работы состоит в том, что в разработанных алгоритмах поиска диссонансов временного ряда предложены оригинальные схемы распараллеливания вычислений и организации данных, которые обеспечивают сокращение избыточных вычислительных операций и увеличение быстродействия поиска.

Практическая ценность диссертационной работы заключается в том, что предложенные параллельные алгоритмы поиска диссонансов временного ряда не требуют обучения и разработаны для аппаратной платформы массово распространенных и доступных в настоящее время графических процессоров. В силу этого результаты исследования могут быть применены для эффективного поиска аномалий во временных рядах широкого спектра предметных областей при минимальном участии аналитика-эксперта (от которого требуется задать лишь длины искомых аномальных подпоследовательностей).

Методология и методы исследования

Проведенные в работе исследования базируются на методах машинного обучения и интеллектуального анализа данных, а также теории временных рядов. В реализации параллельных алгоритмов использованы методы параллельного программирования для графических процессоров и распределенной памяти на основе стандартов CUDA и MPI соответственно.

Положения, выносимые на защиту

На защиту выносятся следующие новые научные результаты.

1. Разработан параллельный алгоритм PD3 для графического процессора, позволяющий найти все диссонансы временного ряда, имеющие заданную длину.

2. Разработан параллельный алгоритм PALMAD для графического процессора, позволяющий найти все диссонансы временного ряда, имеющие длину в заданном диапазоне. Предложена техника ранжирования диссонансов различной длины и их визуализация в виде тепловой карты диссонансов.

3. Разработан параллельный алгоритм PADDi для высокопроизводительного вычислительного кластера с графическими процессорами, позволяющий найти все диссонансы временного ряда (который не может быть целиком размещен в оперативной памяти одного узла кластера), имеющие длину в заданном диапазоне.

4. Разработан метод DiSSiD поиска аномалий потокового временного ряда, включающий в себя нейросетевую модель и алгоритм построения обучающей выборки для указанной модели. DiSSiD позволяет выявлять аномальные подпоследовательности ряда, отражающие нетипичную и редко встречающуюся активности исследуемого субъекта.

5. Проведены вычислительные эксперименты с синтетическими и реальными временными рядами, подтверждающие эффективность разработанных методов и подходов.

Степень достоверности результатов

Эффективность разработанных методов и алгоритмов подтверждена результатами вычислительных экспериментов над реальными и синтетическими данными, проведенными в соответствии с общепринятыми стандар-

тами. Репродуцируемость результатов, полученных в исследовании, обеспечивается размещением исходных текстов программ, реализующих предложенные разработки, и наборов данных, с которыми проведены вычислительные эксперименты, в сети Интернет в свободно доступных репозиториях.

Апробация результатов исследования

Основные положения диссертационной работы, разработанные методы, алгоритмы и результаты вычислительных экспериментов докладывались на *девяти научных конференциях*:

— РСД'2023: Международная научная конференция «Суперкомпьютерные дни в России», 25–26 сентября 2023 г., Москва *(диплом II степени в конкурсе докладов молодых ученых)*;

— ПаВТ'2023: Всероссийская научная конференция с международным участием «Параллельные вычислительные технологии 2023», 28–30 марта 2023 г., Санкт-Петербург *(диплом II степени в конкурсе докладов молодых ученых)*;

— DAMDID'2023: International Conference on Data Analytics and Management in Data Intensive Domains, 25–27 October 2023, Moscow, Russia;

— ЦИСП'2023: Всероссийская научная конференция с международным участием «Цифровая индустрия: состояние и перспективы развития 2023», 21–23 ноября 2023 г., Челябинск;

— ПаВТ'2022: Всероссийская научная конференция с международным участием «Параллельные вычислительные технологии 2022», 29–31 марта 2022 г., Дубна *(диплом II степени в конкурсе докладов молодых ученых)*;

— DAMDID'2022: International Conference on Data Analytics and Management in Data Intensive Domains, 5–7 October 2022, St. Petersburg, Russia;

— ПаВТ'2021: Всероссийская научная конференция с международным участием «Параллельные вычислительные технологии 2021», 30 марта – 1 апреля 2021 г., Волгоград;

— ПаВТ'2020: Всероссийская научная конференция с международным участием «Параллельные вычислительные технологии 2020», 31 марта – 2 апреля 2020 г., Пермь;

— ЦИСП'2020: Всероссийская научная конференция с международным участием «Цифровая индустрия: состояние и перспективы развития 2020», 17–19 ноября 2020 г., Челябинск.

Публикации соискателя по теме диссертации

Основные результаты диссертации опубликованы в *пяти научных работах*, в том числе 4 статьи в журналах, входящих в Ядро РИНЦ и категорию К1 Перечня ВАК (где одна из статей опубликована в журнале, который также входит в квартиль Q2 библиографической базы данных Scopus) и одна статья в журнале, входящем в квартиль Q1 библиографической базы данных Web of Science. Получено *два свидетельства Роспатента* о государственной регистрации программ для ЭВМ.

Статьи в журналах

1. Kraeva, Ya. A Parallel Discord Discovery Algorithm for a Graphics Processor / Ya. Kraeva, M. Zymbler // Pattern Recognition and Image Analysis. – 2023. – Vol. 33, no. 2. – P. 101–112. DOI: 10.1134/S1054661823020062. (Ядро РИНЦ, Перечень ВАК К1, Scopus Q2)

2. Краева, Я.А. Поиск аномалий в сенсорных данных цифровой индустрии с помощью параллельных вычислений / Я.А. Краева // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. – 2023. – Т. 12, № 2. – С. 47–61. DOI: 10.14529/cmse230202. (Ядро РИНЦ, Перечень ВАК К1)

3. Краева, Я.А. Обнаружение аномалий временного ряда на основе технологий интеллектуального анализа данных и нейронных сетей / Я.А. Краева // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. – 2023. – Т. 12, № 3. – С. 50–71. DOI: 10.14529/cmse230304. (Ядро РИНЦ, Перечень ВАК К1)

4. Краева, Я.А. Поиск аномалий в больших временных рядах на кластере с GPU узлами / Я.А. Краева, М.Л. Цымблер // Вычислительные методы и программирование. – 2023. – Т. 24, № 3. – С. 291–304. DOI: 10.26089/NumMet.v24r320. (Ядро РИНЦ, Перечень ВАК К1)

5. Zymbler, M. High-Performance Time Series Anomaly Discovery on Graphics Processors / M. Zymbler, Ya. Kraeva // Mathematics. – 2023. – Vol. 11, no. 14. – Article 3193. DOI: 10.3390/math11143193. (Web of Science Q1)

Свидетельства о регистрации программ

6. Краева, Я.А. DiSSiD: детектор аномалий временного ряда в режиме реального времени / Я.А. Краева // Свидетельство Роспатента о государственной регистрации программы для ЭВМ № 2023685034 от 22.11.2023.

7. Краева, Я.А. PALMAD: детектор аномалий различной длины во временном ряде на графическом процессоре / Я.А. Краева, М.Л. Цымблер // Свидетельство Роспатента о государственной регистрации программы для ЭВМ № 2022667716 от 23.09.2022.

В статьях [1, 4, 5] научному руководителю М.Л. Цымблеру принадлежит постановка задачи, Я.А. Краевой — все полученные результаты.

Структура и объем работы

Диссертация состоит из введения, пяти глав, заключения, списка литературы, списков рисунков, таблиц, алгоритмов и приложения. Объем диссертации составляет 158 страниц, объем списка литературы — 151 наименование.

Содержание работы

В первой главе, «Современные методы поиска аномалий во временных рядах», рассматриваются современные методы и алгоритмы поиска аномалий во временных рядах. Дается классификация подходов к обнаружению аномалий: методы с привлечением учителя (supervised), методы без привлечения учителя (unsupervised) и методы с частичным привлечением учителя (semi-supervised). Описаны основные представители двух последних групп методов, поскольку в настоящее время методы с привлечением учителя, как правило, не применяются на практике. Приведены формальные определения и нотация для обозначения основных понятий, связанных с обнаружением аномалий временных рядов, используемых далее в тексте. Дается обзор публикаций, наиболее близко относящихся к теме диссертации, которые охватывают последовательные и параллельные алгоритмы поиска диссонансов (аномальных подпоследовательностей) временного ряда.

Во второй главе, «Параллельный алгоритм поиска диссонансов фиксированной длины для графических процессоров», представлен новый параллельный алгоритм поиска диссонансов временного ряда, имеющих заданную длину, названный PD3 (Parallel DRAG-based Discord Discovery). Рассмотрены принципы распараллеливания и параллельная реализация стадий отбора кандидатов в диссонансы и очистки диссонансов (отбрасывание ложноположительных экземпляров). Представлены результаты вычислительных экспериментов над реальными и синтетиче-

скими временными рядами, подтверждающие высокую эффективность алгоритма PD3.

Третья глава, «Параллельный алгоритм поиска диссонансов произвольной длины для графических процессоров», описывает новый параллельный алгоритм поиска диссонансов временного ряда, имеющих длину в заданном диапазоне, получивший название PALMAD (Parallel Arbitrary Length MERLIN-based Anomaly Discovery). Описаны принципы применения разработанного ранее алгоритма PD3 для распараллеливания указанной задачи. Доказано утверждение о рекуррентных формулах вычисления среднего арифметического и среднеквадратичного отклонения подпоследовательностей временного ряда, применение которых в вычислении расстояний между подпоследовательностями ряда позволяет существенно сократить объем вычислений при поиске диссонансов. Описан алгоритм ранжирования диссонансов, имеющих различную длину, и способ визуализации результатов его работы в виде тепловой карты диссонансов. Представлены результаты вычислительных экспериментов над реальными и синтетическими временными рядами, подтверждающие высокую эффективность алгоритма PALMAD.

Четвертая глава, «Параллельный алгоритм поиска диссонансов произвольной длины для кластерных вычислительных систем с графическими процессорами», посвящена новому параллельному алгоритму поиска диссонансов временного ряда (который не может быть целиком размещен в оперативной памяти), имеющих длину в заданном диапазоне, названному PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs). Описаны принципы применения разработанных ранее алгоритмов PD3 и PALMAD для распараллеливания указанной задачи. Рассмотрена реализация фазы очистки кандидатов в диссонансы, найденных с помощью алгоритмов PD3 и PALMAD. Представлены результаты вычислительных экспериментов над реальными и синтетическими временными рядами, подтверждающие высокую эффективность алгоритма PADDi.

В пятой главе, «Метод поиска аномалий в потоковых данных», рассмотрен новый метод поиска аномальных подпоследовательностей временного ряда, элементы которого поступают один за другим в режиме реального времени, получивший название DiSSiD (Discord, Snippet, and Siamese Neural Network-based Detector of anomalies). Описаны входящие в DiSSiD нейросетевая модель для поиска аномалий и алгоритм подготовки обучающей выборки для нее. Формирование обучающей выборки использует концепции диссонансов и сниппетов — подпоследовательностей временного ряда, выражающих типичные активности субъекта, описываемого данным рядом. Описана модификация алгоритма поиска сниппетов, позволяющая более точно по сравнению с оригинальным алгоритмом находить указанные подпоследовательности заданного временного ряда. Представлены результаты вычислительных экспериментов над реальными и синтетическими временными рядами, подтверждающие высокую точность DiSSiD при поиске аномалий и быстродействие, достаточное для ее применения в задачах потоковой обработки временных рядов, возникающих в приложениях интеллектуального управления зданиями и сооружениями.

Заключение подводит итоги диссертационной работы и содержит описание ключевых отличий данного исследования от ранее выполненных родственных работ других авторов, а также рекомендации по использованию полученных результатов и направления дальнейших исследований в данной области.

В приложение А вынесены детализированные результаты экспериментального исследования модели DiSSiD.

Глава 1. Современные методы поиска аномалий во временных рядах

В данной главе рассматривается понятие аномалии временного ряда и две его разновидности, точечная аномалия и аномальная подпоследовательность, последняя из которых является предметом диссертационного исследования. Приводится таксономия методов поиска аномалий во временных рядах: поиск с учителем, поиск без учителя и поиск с частичным привлечением учителя. Рассмотрены методы, являющиеся основными представителями указанных групп (за исключением методов поиска с учителем, поскольку в настоящее время они редко применяются на практике). Вводятся формальные определения и нотация для обозначения используемых в последующих главах понятий. Дается детальный обзор публикаций, посвященных последовательным и параллельным алгоритмам поиска аномалий на основе концепции диссонанса временного ряда, наиболее близко относящихся к теме диссертации.

1.1. Таксономия методов поиска аномалий

Общепринятое неформальное определение аномалии дано Д. Хоукинсом (D. Hawkins) в работе [51]: «наблюдение, которое настолько сильно отличается от других наблюдений, что вызывает подозрения в том, что оно было создано иным механизмом» (“an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism”). Применительно к временным рядам целью поиска аномалий являются точечные аномалии и аномальные подпоследовательности [20].

Точечная аномалия представляет собой элемент временного ряда, значение которого существенно отличается от всех остальных или соседних с ним элементов ряда (*глобальная* и *локальная* точечная аномалия соответственно). *Аномальной подпоследовательности* соответствует непре-

рывный набор расположенных друг за другом элементов ряда, коллективное поведение которых существенно необычно, хотя каждый элемент такой подпоследовательности не обязательно является точечной аномалией.

В отличие от поиска точечных аномалий, который может быть осуществлен с помощью относительно простых статистических методов («правило трех сигм» [119], оценка χ^2 [140], алгоритм Пейджа [105] и др.), поиск аномальных подпоследовательностей представляет собой существенно более сложную и гораздо чаще востребованную на практике задачу, поскольку при ее решении необходимо в том числе учитывать длину искомой подпоследовательности [20]. В соответствии с этим в рамках данного исследования рассматривается задача обнаружения аномальных подпоследовательностей временного ряда.

Методы поиска аномальных подпоследовательностей временного ряда разделяют на три группы [20, 55, 116]: поиск с учителем (supervised), поиск без учителя (unsupervised) и поиск с частичным привлечением учителя (semi-supervised).

Методы поиска аномалий *с учителем* моделируют нормальное и девиантное поведение во временном ряде и включают в себя этап обучения, после которого возможно их применение во временных рядах, не известных модели. Методы данной группы требуют для обучения предварительно размеченный экспертом или специализированной программой временной ряд (ряды), где подпоследовательности имеют одну из двух меток: «норма» или «аномалия». По своей природе методы поиска аномалий с учителем ограничены в своей способности обнаруживать аномалии, не задействованные на этапе обучения, и поэтому в настоящее время они редко применяются на практике [116]. В соответствии с обзором [116] можно привести лишь три относительно недавние разработки, входящие в данную группу методов: MultiHMM [83], HIF [95] и NF [114].

Методы поиска аномалий *без учителя* не требуют предварительных знаний о временном ряде и не включают в себя этап обучения. Указанные методы основываются на предположениях о свойствах, которыми обладают

аномальные подпоследовательности: они встречаются реже, имеют иную форму, происходят из другого распределения вероятностей и др.

Методы *с частичным привлечением учителя* включают в себя этап обучения, на котором пытаются изучить только нормальное поведение временного ряда, который фигурирует в качестве обучающей выборки модели. Модель, будучи примененной к тестовому временному ряду, помечает как аномальные подпоследовательности, которые не соответствуют нормальному поведению.

1.2. Методы без привлечения учителя

В недавно опубликованных обширных обзорах методов поиска аномалий во временных рядах [107, 116] суммарно рассматривается более 150 различных методов, около половины из которых относятся к методам без привлечения учителя. Поэтому в данном разделе кратко рассмотрены основные представители данной группы методов, некоторые из которых далее были задействованы в вычислительных экспериментах данного исследования.

Алгоритмы MP (Matrix Profile) [142] и DAMP (Discord Aware Matrix Profile) [92] предполагают вычисление матричного профиля временного ряда и нахождение в нем локальных максимумов. Соответственно, аномалиями считаются те подпоследовательности ряда, которые имеют наибольшие расстояния до своих ближайших соседей. В случае алгоритма DAMP рассматриваются соседи данной подпоследовательности, имеющие меньшие по сравнению с ней индексы (находящиеся слева от нее).

Нижеследующие алгоритмы трактуют подпоследовательности временного ряда как точки многомерного метрического пространства и определяют аномальные подпоследовательности с учетом расстояния соответствующих точек и/или плотности их скопления.

Метод изолирующего леса (IForest, Isolation Forest) [88] строит бинарное дерево решений на основе разделения пространства признаков. Образцы,

которые находятся в листьях дерева на более низком уровне от его корня, с большей вероятностью будут считаться аномалиями.

Метод локального фактора выброса (LOF, Local Outlier Factor) [27] вычисляет отношение локальной плотности образца к локальной плотности его соседей. Образцы считаются аномальными, если они имеют существенно меньшую плотность, чем их соседи.

Метод NormA [23] идентифицирует нормальные шаблоны временного ряда на основе кластеризации и вычисляет расстояние от них до образцов. Если расстояния превышают некоторое пороговое значение, то такие образцы считаются аномалиями.

Метод главных компонент (PCA) [10] проецирует данные на гиперплоскость меньшей размерности, а образцы данных, находящиеся на значительном расстоянии от этой плоскости, могут быть отмечены как аномалии.

Одноклассовый метод опорных векторов (OCSVM, One-class Support Vector Machine) [117] находит гиперплоскость, разделяющую нормальные данные от аномальных в пространстве признаков.

Отдельную крупную группу в классе методов без привлечения учителя составляют алгоритмы, основанные на концепции диссонанса временного ряда. Концепция диссонанса была предложена Кеогом (Keogh) и др. в работе [86] и в настоящее время рассматривается как один из лучших подходов к поиску аномалий во временных рядах [31, 32]. Диссонанс имеет интуитивно понятное определение: это подпоследовательность временного ряда, которая наиболее удалена от своего ближайшего соседа (ближайшей подпоследовательности, не пересекающейся с данной). Диссонансы привлекательны для поиска аномалий, поскольку требуют от аналитика задать лишь один параметр — длину подпоследовательности (искомых диссонансов). Поскольку данное исследование базируется на концепции диссонанса, алгоритмы-представители данной группы детально рассмотрены ниже в разделе 1.5.

1.3. Методы с частичным привлечением учителя

В недавно опубликованных обширных обзорах методов поиска аномалий во временных рядах [107, 116] суммарно рассматривается более 150 различных методов, около половины из которых относятся к методам с частичным привлечением учителя. Поэтому в данном разделе кратко рассмотрены основные представители данной группы методов, некоторые из которых далее были задействованы в вычислительных экспериментах данного исследования.

Автокодировщик (AE) [115], Bagel [84], IE-CAE [45] и TAnoGAN [19] проецируют данные в скрытое пространство меньшей размерности и затем восстанавливают их. При этом ожидается, что аномалии будут иметь большую разницу между исходными и восстановленными данными. Перечисленные методы отличаются архитектурой используемой нейронной сети. Метод AE использует классический автокодировщик, где энкодером и декодером являются многослойные перцептроны, Bagel и IE-CAE — условный вариационный автокодировщик, TAnoGAN — генеративно-состязательную сеть, где генератор и дискриминатор являются рекуррентными нейронными сетями LSTM.

Методы LSTM-AD [94], полиномиальная аппроксимация (POLY) [85], DeepAnT [99] и OceanWNN [132] находят связь между текущими и предыдущими значениями временного ряда. Для обнаружения аномалий вычисляется разница между исходными и прогнозными значениями. Методы LSTM-AD и DeepAnT применяют рекуррентную и сверточную нейронные сети соответственно, рассмотрим их более детально ниже.

Метод LSTM-AD [94] выполняет обнаружение аномалий в многомерных временных рядах на основе применения нейронных сетей долгой краткосрочной памяти (LSTM, Long Short-Term Memory). Метод предполагает, что размеченные данные распределяются на следующие группы. Нормальные подпоследовательности делятся на четыре группы: обучающая выборка (s_N), две валидационные выборки (v_{N1} и v_{N2}) и тестовая выборка (t_N).

Аномальные подпоследовательности делятся на две группы: валидационная (v_A) и тестовая выборки (t_A). Метод использует двухступенчатую схему «предсказание-детекция»: сперва модель на основе многослойной сети LSTM предсказывает значения временного ряда, а затем вычисляется распределение ошибок предсказания, с помощью которого обнаруживаются аномалии. Многослойная сеть LSTM организуется следующим образом. Во входном слое для каждой из m размерностей ряда имеется один нейрон, $d \times \ell$ нейронов в выходном слое таких, что на каждое из ℓ предсказанных значений для каждой из d размерностей имеется один нейрон (где d, ℓ — параметры и $1 \leqslant d \leqslant m$). Нейроны скрытого слоя LSTM являются полносвязными, что реализовано с помощью рекуррентных связей. Несколько LSTM слоев (обычно два) объединяются в стек таким образом, чтобы каждый нейрон в скрытом слое LSTM снизу был полностью соединен с каждым нейроном в скрытом слое LSTM над ним посредством прямых соединений. Обучение описанной модели выполняется на выборке s_N, выборка v_{N1} используется для раннего останова обучения при подборе весов нейросети. Фаза детекции выполняется следующим образом. Для каждой из выбранных d размерностей ℓ раз выполняется предсказание ℓ значений. Далее применяется вектор ошибок, элемент которого представляет собой разность между реальным и предсказанным значениями. Модель, обученная на выборке s_N, используется для вычисления векторов ошибок для последовательностей валидационной и тестовой выборок. Векторы ошибок моделируются таким образом. Векторы ошибок для элементов из выборки v_{N1} используются для оценки параметров распределения с использованием оценки максимального правдоподобия. Подпоследовательность классифицируется как аномалия, если функция оценка максимального правдоподобия меньше наперед заданного параметра τ, иначе она помечается как «норма». При этом выборки v_{N2} и v_A применяются для определения τ посредством максимизации значения F-меры, когда аномальные подпоследовательности считаются принадлежащими положительному классу, а нормальные — напротив, отрицательному.

Метод DeepAnT [99] позволяет обнаруживать одиночные выбросы и аномальные подпоследовательности временного ряда как в онлайн, так и в офлайн режиме. DeepAnT использует не содержащие разметку временные ряды для изучения распределения данных, которое затем используется для прогнозирования нормального поведения временного ряда. DeepAnT состоит из двух модулей: предсказателя и детектора аномалий временного ряда. Модуль предсказания использует глубокую сверточную нейронную сеть для прогнозирования будущего значения ряда на определенном горизонте, используя в качестве контекста окно предыдущих значений ряда. Нейронная сеть включает в себя два сверточных слоя с размером ядра 32 и Линейным выпрямителем (ReLU, Rectified linear unit) в качестве функции активации. К выходу каждого из слоев применяется операция подвыборки по максимальному значению (MaxPooling). Последним слоем нейросети является полносвязный. В качестве функции потерь пр обучении нейросети применяется средняя абсолютная ошибка (MAE, Mean Absolute Error). Полученное прогнозное значение затем передается детектору аномалий, который отвечает за разметку значения как нормального или аномального. DeepAnT допускает обучение на временных рядах, из которых не удаляются выбросы и аномальные подпоследовательности.

1.4. Формальные определения и обозначения

В данном разделе вводятся формальные определения и нотация для обозначения используемых в последующих главах понятий в соответствии с работами [40, 60, 68, 138].

1.4.1. Временной ряд и подпоследовательность

Временной ряд (time series) T представляет собой последовательность вещественных значений, взятых в хронологическом порядке:

$$T = \{t_i\}_{i=1}^{n}, \quad t_i \in \mathbb{R}. \tag{1.1}$$

Число n называется длиной ряда и обозначается $|T|$.

Подпоследовательность (subsequence) $T_{i,m}$ временного ряда T представляет собой непрерывный промежуток из m элементов ряда, начиная с позиции i:

$$T_{i,m} = \{t_k\}_{k=i}^{i+m-1}, \quad 3 \leqslant m \ll n, \quad 1 \leqslant i \leqslant n - m + 1. \tag{1.2}$$

Множество всех подпоследовательностей ряда T, имеющих длину m, обозначим как S_T^m. Мощность указанного множества обозначим как N, $N = |S_T^m| = n - m + 1$.

Потоковый временной ряд (streaming time series) представляет собой временной ряд, элементы которого непрерывно поступают для обработки один за другим в режиме реального времени. Обработке подлежит выбранное аналитиком *окно (window)* потокового временного ряда, имеющее фиксированную длину n, которое заканчивается текущим поступившим значением t_n:

$$T = (\ldots,\ \{t_i\}_{i=1}^{n},\ \ldots), \quad t_i \in \mathbb{R}. \tag{1.3}$$

1.4.2. Диссонансы

Подпоследовательности $T_{i,m}$ и $T_{j,m}$ ряда T считаются *не пересекающимися*, если $|i - j| \geqslant m$. Некая подпоследовательность ряда, не пересекающаяся с данной подпоследовательностью C, обозначается как M_C.

Подпоследовательность D ряда T является *диссонансом (discord)* [138], если

$$\min_{M_D \in T} \mathrm{Dist}(D, M_D) \geqslant r, \tag{1.4}$$

где $\mathrm{Dist}(\cdot\,,\cdot\,)$ — симметричная неотрицательная функция расстояния и r — порог расстояния (параметр, задаваемый аналитиком). Иными словами, некая подпоследовательность ряда является диссонансом, если ее *ближайший сосед* (ближайшая в смысле расстояния $\mathrm{Dist}(\cdot\,,\cdot\,)$ и не пересекающаяся с ней подпоследовательность) находится на расстоянии не менее чем r.

Для поиска диссонансов в качестве функции расстояния выбираются, как правило, евклидова метрика или ее производные: квадрат евклидова расстояния, z-нормализованное евклидово расстояние или его квадрат [138], определяемые следующим образом.

Пусть имеются подпоследовательности X и Y длины m временного ряда T, тогда евклидово расстояние ED между X и Y вычисляется как

$$\mathrm{ED}(X,\ Y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}. \tag{1.5}$$

Z-нормализация подпоследовательности (ряда) T представляет собой подпоследовательность (ряд) $\hat{T} = \{\hat{t}_i\}_{i=1}^{m}$, где элементы вычисляются следующим образом:

$$\hat{t}_i = \frac{t_i - \mu}{\sigma}, \quad \mu = \frac{1}{m}\sum_{i=1}^{m} t_i, \quad \sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m} t_i^2 - \mu^2}. \tag{1.6}$$

В данном исследовании в качестве функции $\mathrm{Dist}(\cdot\,,\cdot\,)$ используется квадрат нормированного евклидова расстояния:

$$\mathrm{Dist}(T_{i,m}, T_{j,m}) = \mathrm{ED}^2_{\mathrm{norm}}(T_{i,m}, T_{j,m}) = \mathrm{ED}^2(\hat{T}_{i,m}, \hat{T}_{j,m}). \tag{1.7}$$

Для вычисления нормированного евклидова расстояния применяется формула, предложенная в работе [98], которая позволяет выполнить вычисления быстрее, чем по формулам (1.5)–(1.6):

$$\mathrm{ED}^2_{\mathrm{norm}}(T_{i,m}, T_{j,m}) = 2m\left(1 - \frac{\langle T_{i,m},\ T_{j,m}\rangle - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right), \tag{1.8}$$

где подпоследовательности $T_{i,m}$ и $T_{j,m}$ рассматриваются как вектора в евклидовом пространстве $\mathbb{R}^m$, μ_i и μ_j, σ_i и σ_j — среднее арифметическое и стандартное отклонение координат указанных векторов соответственно.

1.4.3. Сниппеты

Сниппеты (snippets) [60] временного ряда представляют собой подпоследовательности, выражающие типичные активности некоего субъекта, деятельность которого описывает данный ряд. Формальное определение сниппетов выглядит следующим образом.

Пусть имеется временной ряд T и задана длина подпоследовательности m ($m \ll n$). Разобьем ряд на не пересекающиеся *сегменты* длины m, без ограничения общности считая, что n кратно m. Рассмотрим множество сегментов Seg_T^m:

$$Seg_T^m = \{Seg_i\}_{i=1}^{n/m}, \quad Seg_i = T_{m\cdot(i-1)+1,\,m}. \tag{1.9}$$

Сниппеты представляют собой непустое подмножество Seg_T^m из K сегментов, где K — параметр, отражающий количество активностей субъекта, интересующее исследователя ($1 \leqslant K \leqslant n/m$). Обозначим множество сниппетов ряда T, имеющих длину m, как C_T^m:

$$C_T^m = \{C_i\}_{i=1}^{K}, \quad C_i \in Seg_T^m. \tag{1.10}$$

С каждым сниппетом ассоциированы следующие атрибуты: индекс, профиль, ближайшие соседи и мощность (значимость) данного сниппета. *Индекс сниппета* $C_i \in C_T^m$ обозначается как $C_i.index$ и представляет собой номер j сегмента Seg_j, которому соответствует подпоследовательность ряда $T_{m\cdot(j-1)+1,\,m}$.

Профиль сниппета, обозначаемый как $C_i.profile$, представляет собой вектор MPdist-расстояний [46] между данным сниппетом и подпоследовательностями ряда:

$$C_i.profile = \{d_k\}_{k=1}^{n-m+1}, \quad d_k = \mathrm{MPdist}(C_i,\, T_{k,\,m}). \tag{1.11}$$

Множество ближайших соседей сниппета $C_i \in C_T^m$ обозначается как $C_i.NN$ и содержит подпоследовательности ряда, которые более близки дан-

ному сниппету, чем другим сегментам ряда, в смысле расстояния MPdist:

$$C_i.NN = \{T_{j,m} \mid Seg_{C_i.index} = \arg\min_{1 \leqslant q \leqslant n/m} \mathrm{MPdist}(T_{j,m},\, Seg_q),\ 1 \leqslant j \leqslant n-m+1\}. \tag{1.12}$$

Мощность сниппета $C_i \in C_T^m$ обозначается как $C_i.frac$ и вычисляется как доля мощности множества ближайших соседей сниппета от общего количества подпоследовательностей ряда, имеющих длину m, при этом сниппеты упорядочиваются по убыванию их мощности:

$$C_i.frac = \frac{|\,C_i.NN|}{N}. \tag{1.13}$$

$$\forall C_i, C_j \in C_T^m :\ i < j \iff C_i.frac \geqslant C_j.frac. \tag{1.14}$$

Расстояние $\mathrm{MPdist}(\cdot\,,\cdot\,)$ между подпоследовательностями A и B ($|A| = |B| = m$) определяется следующим образом [46]. Фиксируем задаваемый аналитиком параметр ℓ, который отражает длину семантически значимого непрерывного промежутка точек подпоследовательности ($\lceil 0.3m \rceil \leqslant \ell \leqslant \lceil 0.8m \rceil$, типичное значение — $\ell = \lceil 0.5m \rceil$). Тогда близость A и B в смысле MPdist пропорциональна количеству в каждой из них подпоследовательностей длины ℓ, близких в смысле метрики, основанной на нормализованном евклидовом расстоянии.

Хотя функция MPdist не удовлетворяет неравенству треугольника, она устойчива к выбросам, шумам и пропущенным значениям, а также инвариантна к амплитуде, сдвигу и фазе временного ряда [46]. Вычисление MPdist предполагает последовательное выполнение следующих операций:

1) вычисление матричных профилей для рядов A и B, взятых в указанном и обратном порядке;

2) конкатенация вычисленных профилей в один временной ряд;

3) упорядочение элементов полученного временного ряда по возрастанию;

4) взятие в качестве ответа k-го элемента упорядоченного ряда.

Формальная запись выглядит следующим образом:

$$\mathrm{MPdist}_\ell(A, B) = \mathrm{AscSort}\big(P_{ABBA}\big)(k), \quad P_{ABBA} = P_{AB} \bullet P_{BA}, \tag{1.15}$$

где $\mathrm{AscSort}(\cdot)$ — операция упорядочивания элементов последовательности по возрастанию, символ $\bullet$ обозначает операцию конкатенации, k — задаваемый аналитиком параметр ($0 < k < m$, типичное значение — $k_{\text{default}} = \lceil 0.1m \rceil$).

Матричным профилем [141] рядов A и B для длины подпоследовательности ℓ называется ряд P_{AB}, i-м элементом которого является расстояние между i-й подпоследовательностью ряда A, имеющей длину ℓ, и ее ближайшим соседом в ряде B:

$$P_{AB} = \{\mathrm{ED}^2_{\text{norm}}(A_{i,\ell}, B_{j,\ell})\}_{i=1}^{m-\ell+1}, \quad B_{j,\ell} = \arg \min_{1 \leqslant q \leqslant m-\ell+1} \mathrm{ED}^2_{\text{norm}}(A_{i,\ell},\, B_{q,\ell}). \tag{1.16}$$

Аналогичным образом определяется матричный профиль рассматриваемых рядов, взятых в порядке B и A, и обозначается как P_{BA}.

1.5. Обзор близких работ

1.5.1. Эволюция концепции диссонанса

В работе [86], Keog и др. предложили алгоритм HOTSAX (Heuristically Ordered Time series using Symbolic Aggregate ApproXimation, эвристическое упорядочивание временного ряда с использованием символьного агрегатного приближения) для поиска диссонансов ряда, который может быть целиком размещен в оперативной памяти. HOTSAX выполняет кодирование подпоследовательностей временного ряда на основе техники SAX (символьное агрегатное приближение) [87] и евклидовой метрики. HOTSAX сканирует пары подпоследовательностей временного ряда, вычисляя расстояния между ними и находит максимум среди расстояний до ближайших соседей.

Алгоритм задействует префиксное дерево [42] для индексирования подпоследовательностей. Во время сканирования бесперспективные подпоследовательности отбрасываются без вычисления расстояний. Бесперспективной является подпоследовательность, сосед которой ближе, чем текущий максимум расстояний среди всех ближайших соседей. HOTSAX применяет эвристику, которая позволяет отбрасывать как можно больше бесперспективных кандидатов. Улучшениями HOTSAX являются алгоритмы iSAX [120] и HOT-iSAX [29] (индексированное символьное агрегатное приближение), WAT [28, 43] (применение вейвлетов Хаара вместо SAX и дополненного префиксного дерева), HashDD [126] (применение хэш-таблиц вместо префиксного дерева), HDD-MBR [33] (применение R-деревьев [48] вместо префиксного дерева), BitClusterDiscord [82] (кластеризация битовых представлений подпоследовательностей) и HST (HOTSAX Time) [15] (уменьшение пространства поиска диссонансов за счет предложенной техники прогрева и сходства между подпоследовательностями, близкими по времени).

В работе [118] Сенин (Senin) и др. предложили алгоритм RRA (Rare Rule Anomaly, аномалии редких правил) поиска диссонансов переменной длины, основанный на алгоритме HOTSAX. RRA работает с временным рядом, дискретизированным с помощью преобразования SAX. Поскольку символы, редко используемые в грамматических правилах, не повторяются и, следовательно, потенциально необычны, диссонансы соответствуют нечастым грамматическим правилам, отличными по длине. Принимая во внимание, что длины подпоследовательностей различаются, расстояние между ними вычисляется путем сокращения самой длинной подпоследовательности с помощью кусочно-агрегатного приближения (PAA, Piecewise Aggregate Approximation) [67] для получения подпоследовательностей одинаковой длины. Хотя алгоритм RRA является шагом вперед от алгоритма HOTSAX к обнаружению диссонансов без параметров, как и его предшественник, он ограничен рассмотрением случая временных рядов, которые могут быть целиком размещены в оперативной памяти.

В работе [138] Кеог, Янков (Yankov) и др. представили алгоритм DRAG (Discord Range Aware Gathering, отбор диапазонных диссонансов) для по-

иска диссонансов временного ряда, который не может быть целиком размещен в оперативной памяти. В алгоритме DRAG предлагается новая концепция диапазонного диссонанса — диссонанса, удаленного по крайней мере на порог r от ближайшего соседа, где r — параметр, который задается аналитиком. DRAG выполняется в две фазы, каждая из которых требует одного прохода по исходному временному ряду: отбор и очистка, на которых выполняются соответственно поиск кандидатов в диапазонные диссонансы и отбрасывание ложноположительных образцов. В работе [122] Сон (Son) предложил улучшение алгоритма DRAG путем применения хэш-структуры данных для ускорения фазы отбора.

Авторы алгоритма DRAG предложили следующую эвристическую процедуру подбора параметра r. Посредством равномерной выборки получаем фрагмент исходного временного ряда максимальной длины, который может быть размещен в оперативной памяти. Далее в указанном фрагменте алгоритм HOTSAX обнаруживает диссонанс. В итоге, порог r выбирается равным расстоянию от найденного диссонанса до его ближайшего соседа.

Однако, описанная эвристика не дает формального пути для подбора параметра r, который гарантировал бы эффективный поиск диссонансов [98]. В идеальном случае параметр r должен быть установлен таким образом, чтобы он был немного меньше, чем расстояние от найденного в итоге диссонанса до его ближайшего соседа [101]. Тогда вычислительная и пространственная сложности алгоритма DRAG составляют $O(mn)$, где n — длина временного ряда, m — длина диссонанса. Если значение r существенно меньше вышеупомянутого расстояния, то алгоритм найдет диссонанс, но его вычислительная и пространственная сложности будут выше, $O(n^2)$. Наконец, если r больше вышеупомянутого расстояния, то диссонансы не будут найдены. Кроме того, алгоритм DRAG, подобно своему предшественнику, алгоритму HOTSAX, способен находить диссонансы фиксированной длины, но не диссонансы произвольной длины. Подход «грубой силы», при котором алгоритм DRAG запускается циклически для каждой длины диссонанса и заданного диапазона длин, является вычислительно невозмож-

ным, поскольку на каждой итерации такого цикла необходимо подбирать параметр r заново с нуля.

Недавно предложенный Keogом и др. алгоритм MERLIN [101] снимает ограничение на поиск диссонансов произвольной длины, имеющееся у алгоритма DRAG. MERLIN выполняет повторные вызовы алгоритма DRAG, при этом адаптивно подбирая параметр r. В экспериментах MERLIN эффективно и точно находит диссонансы произвольной длины, опережая аналоги по точности и быстродействию [101]. Кроме того, в экспериментах авторы показывают, что MERLIN способен находить точечные, коллективные и контекстные аномалии в соответствии с таксономией в работе [31]. Алгоритм MERLIN, имея в качестве параметра лишь диапазон длин искомых диссонансов независимо от рассматриваемой предметной области, выгодно отличается от многочисленных нейросетевых подходов к поиску аномалий во временных рядах, которые требуют подбора большого количества параметров (как правило, не менее семи) [36, 116].

В продолжении своих исследований Keог и др. в работе [102] представили алгоритм MERLIN++, который ускоряет поиск диссонансов за счет применения на фазе очистки диссонансов алгоритма кластеризации Орчарда [104, 131]. В экспериментах над синтетическим временным рядом длины 2^{16}, полученным с помощью модели случайных блужданий [108], алгоритм MERLIN++ показал на порядок лучшее быстродействие [102], чем MERLIN.

Тем не менее, алгоритмы MERLIN и MERLIN++ являются последовательными, и их распараллеливание могло бы увеличить производительность поиска диссонансов. Кроме того, имеющиеся в алгоритмах MERLIN и MERLIN++ повторные вызовы алгоритма DRAG дают частично повторяющиеся вычисления (например, при нормализации подпоследовательностей, имеющих длину в диапазоне длин искомых диссонансов), которые, будучи выполненными лишь однажды, могли бы увеличить быстродействие поиска диссонансов.

1.5.2. Базовые последовательные алгоритмы

Алг. 1.1. DRAG (IN T, m, r; OUT $\mathcal{D}$)

Фаза 1. Отбор кандидатов	Фаза 2. Очистка диссонансов
1: $\mathcal{C} \leftarrow \{T_{1,m}\}$	1: $\mathcal{D} \leftarrow \varnothing$; $\forall c \in \mathcal{C}\ c.nnDist \leftarrow +\infty$
2: **for all** $s \in S_T^m \setminus T_{1,m}$ **do**	2: **for all** $s \in S_T^m$ **do**
3: $isCand \leftarrow$ TRUE	3: **for all** $c \in \mathcal{C}$ **where** $c \in M_s$ **and** $s \neq c$ **do**
4: **for all** $c \in \mathcal{C}$ **and** $c \in M_s$ **do**	4: $d \leftarrow \text{EarlyStop ED}(s, c)$
5: **if** $\text{ED}(s, c) < r$ **then**	5: **if** $d < r$ **then**
6: $\mathcal{C} \leftarrow \mathcal{C} \setminus c$	6: $\mathcal{C} \leftarrow \mathcal{C} \setminus c$
7: $isCand \leftarrow$ FALSE	7: **else**
8: **if** $isCand$ **then**	8: $\mathcal{D} \leftarrow \mathcal{D} \cup c$
9: $\mathcal{C} \leftarrow \mathcal{C} \cup s$	9: $c.nnDist \leftarrow \min(c.nnDist, d)$
10: **return** $\mathcal{C}$	10: **return** $\mathcal{D}$

Рассмотрим более детально алгоритмы DRAG и MERLIN, на которых базируются параллельные алгоритмы поиска диссонансов, разработанные в рамках данного исследования. Псевдокод алгоритма DRAG [138] представлен в алг. 1.1. DRAG выполняется в две фазы, отбор кандидатов и очистка диссонансов, в рамках которых выполняется соответственно селекция потенциальных диапазонных диссонансов и отбрасывание ложноположительных экземпляров. На первой фазе DRAG сканирует временной ряд T с помощью скользящего окна длины, равной длине искомых диссонансов, и для каждой подпоследовательности $s \in S_T^m$ проверяет возможность каждого кандидата c из множества кандидатов $\mathcal{C}$ быть диссонансом. Если кандидат c не проходит проверку, то он удаляется из указанного множества. В итоге новый элемент s либо добавляется в множество кандидатов, если он, вероятно, является диссонансом, либо отбрасывается. На второй фазе алгоритм сначала инициализирует у каждого кандидата расстояние до его ближайшего соседа значением $+\infty$. Затем DRAG сканирует временной ряд T, вычисляя расстояние между каждой подпоследовательностью $s \in S_T^m$ и каждым кандидатом c. Вычисляя расстояние $\text{ED}(s, c)$, функция EarlyStopED останавливает вычисление суммы $\sum_{k=1}^{m}(s_k - c_k)^2$ при достижении значения $k = \ell$ $(1 \leqslant \ell \leqslant m)$ такого, что

$\sum_{k=1}^{\ell}(s_k - c_k)^2 \geqslant c.nnDist^2$, где запись $c.nnDist$ обозначает расстояние подпоследовательности c до ее ближайшего соседа. Если расстояние меньше r, то кандидат является ложноположительным и навсегда удаляется из множества $\mathcal{C}$. Если вышеуказанное расстояние меньше, чем текущее значение $c.nnDist$ (и по-прежнему больше, чем r, иначе кандидат был бы отброшен), то текущее расстояние до ближайшего соседа обновляется. Корректность описанной процедуры доказана в оригинальной статье [138].

Алгоритм MERLIN [101] получает на входе временной ряд T длины n, диапазон длин диссонансов $minL..maxL$ ($minL \leqslant maxL \ll n$) и выдает множество $\mathcal{D}$, которое состоит из диссонансов, имеющих длину в указанном выше диапазоне: $\mathcal{D} = \cup_{m=minL}^{maxL} D_m$, где D_m обозначает подмножество диссонансов, имеющих длину m.

Алгоритм подбирает параметр r следующим образом. Поиск диссонансов осуществляется от меньшего значения длины из заданного диапазона к большему. На каждом шаге алгоритм MERLIN вычисляет среднее значение μ и стандартное отклонение σ расстояний от пяти последних найденных диссонансов до их ближайших соседей и затем вызывает алгоритм DRAG, передавая ему в качестве параметра значение порога $r = \mu - 2\sigma$. Если DRAG не находит диссонанс, то из r вычитается величина σ до тех пор, пока завершение DRAG не станет успешным (т.е. будет найден диссонанс). Для первых пяти длин порог r устанавливается следующим образом. Для диссонансов минимальной длины $minL$ порог полагается равным $r = 2\sqrt{minL}$, поскольку это максимально возможное расстояние между парой подпоследовательностей таковой длины. Затем r уменьшается вполовину до тех пор, пока поиск диссонансов алгоритмом DRAG с таким порогом не принесет успех. Для установки значения порога у следующих четырех длин диссонансов, алгоритм берет расстояние от диссонанса, полученного на предыдущем шаге, до его ближайшего соседа, за вычетом небольшой величины в 1%. Вычитание одного процента продолжается, пока поиск диссонансов с таким порогом не принесет успех. Обоснование вышеописанной процедуры приводится в оригинальной работе [101].

1.5.3. Параллельные алгоритмы

В работах [9, 151] Цымблером и др. предложена схема распараллеливания алгоритма поиска диссонансов HOTSAX для многоядерных процессоров Intel и графических процессоров на основе применения технологий OpenMP [124] и OpenACC [111] соответственно. Алгоритм использует матрично-векторные структуры данных, чтобы организовать вычисления с как можно бóльшим количеством векторизуемых циклов, имеющих фиксированное количество повторений. Подобно последовательному предшественнику, алгоритм различает следующие виды подпоследовательностей: редкие (SAX-коды которых имеют низкую частоту возникновения в исходном временном ряде) и частые (остальные), а также для каждой подпоследовательности — соседи (SAX-коды которых совпадают с SAX-кодом данной подпоследовательности) и чужаки (остальные). Параллельный алгоритм выполняет просмотр всех подпоследовательностей ряда посредством двух вложенных циклов, распараллеливание которых зависит от числа нитей, используемых приложением, и вида сканируемых подпоследовательностей.

В работе [139] (которая является расширенной версией статьи [138]) Янков и др. рассмотрели параллельную версию алгоритма DRAG на основе применения парадигмы MapReduce [38], предложив следующее. Пусть исходный временной ряд разбит на равные по длине фрагменты, размещенные на P узлах вычислительного кластера. Каждый узел выполняет отбор кандидатов в своем фрагменте с одинаковым порогом r, результатом которого является множество локальных кандидатов $\mathcal{C}_i$. Тогда множество глобальных кандидатов $\mathcal{C}$ вычисляется как $\mathcal{C} = \cup_{i=1}^{P} \mathcal{C}_i$ и выполняется его пересылка каждому узлу кластера. Далее, узел выполняет очистку полученного множества глобальных кандидатов в пределах своего фрагмента и получает множество локально очищенных кандидатов $\tilde{\mathcal{C}}_i$. Итоговое множество глобальных диссонансов $\mathcal{D}$ вычисляется как $\mathcal{D} = \cap_{i=1}^{P} \tilde{\mathcal{C}}_i$. Описанная схема была смоделирована авторами на восьми не связанных в кластерную

вычислительную систему компьютерах, и по итогу симуляции (без реальных обменов кандидатами) показала ускорение, близкое к линейному.

В работе [148] Цымблер и др. предложили улучшенную схему распараллеливания алгоритма DRAG для вычислительного кластера на базе многоядерных процессоров Intel. Алгоритм имеет два уровня параллелизма — на уровне узлов кластера и внутри узла, реализуемые соответственно путем фрагментации временного ряда с применением для обменов данными технологии MPI (Message Passing Interface) [121] и организацией данных узла в виде матрично-векторных структур, обрабатываемых с помощью технологии OpenMP [124]. В отличии от описанной выше схемы, авторы сначала выполняют очистку множества локальных кандидатов $\mathcal{C}_i$ с одинаковым для всех фрагментов порогом r, результатом которого является множество $\tilde{\mathcal{C}}_i$, а затем создают множество глобальных кандидатов как $\mathcal{C} = \cup_{i=1}^{P} \tilde{\mathcal{C}}_i$ в силу того, что кандидат не может быть диссонансом, если он был отброшен хотя бы одним узлом в ходе отбора. В вычислительных экспериментах [148] авторы показали, что описанное выше изменение позволяет существенно сократить мощность множества глобальных кандидатов и повысить тем самым быстродействие алгоритма. Разработанный алгоритм также показал существенно лучшее быстродействие, чем у следующих параллельных алгоритмов-конкурентов, основанных на алгоритме DRAG, поскольку они предполагают интенсивные обмены данными между узлами кластера: DDD (Distributed Discord Discovery) [136] и PDD (Parallel Discord Discovery) [56].

К методам поиска диссонансов может быть также отнесена концепция матричного профиля (matrix profile, далее МП), предложенная Кеогом и др. в работе [142]. Для заданных временного ряда и длины его подпоследовательности МП представляет собой временной ряд, в котором i-й элемент является расстоянием от i-й подпоследовательности ряда до ее ближайшего соседа. МП играет роль базовой структуры данных для различных алгоритмов интеллектуального анализа временных рядов, нацеленных на поиск шаблонов (смысловые мотивы [58], сниппеты [59], цепочки [145] и др.). В силу определения (1.4) диссонансы могут быть выявлены как побочный результат вычисления МП нахождением его локальных макси-

мумов. Однако сложность вычисления МП высока и составляет $O(n^2)$ (где n — длина ряда) [142, 146], что в случае больших временных рядов приводит к низкому быстродействию, что подтверждалось следующими экспериментами. Последовательный алгоритм вычисления МП SCRIMP [142] уступает в быстродействии алгоритму MERLIN [101]. Параллельные алгоритмы вычисления МП на графическом процессоре и кластерной системе, соответственно GPU-STAMP [142] и MP-HPC [109], показывают быстродействие ниже, чем параллельный алгоритм, предложенный Цымблером и др. [148] (при равенстве пиковой производительности соответствующих аппаратных платформ).

В работе [127] Тай (Thuy) и др. представили концепцию K-дистанционного диссонанса, представляющего собой подпоследовательность, для которой сумма расстояний от нее до K ее ближайших соседей максимальна, где K — параметр. Данная концепция направлена на решение проблемы «уродливых близнецов» (“twin freak”) [133], когда с помощью диссонанса не может быть найдена аномальная (редко встречающаяся) подпоследовательность, если она встречается во временном ряде более чем однажды. Указанная концепция является модификацией понятия J-дистанционного диссонанса [57], предложенного ранее Хуангом (Huang) и др., где используется расстояние до k-го ближайшего соседа, где k — параметр. Авторы разработали алгоритм KBF_GPU (Brute-Force for K-distance discord), который выполняет поиск K-дистанционного диссонанса на графическом процессоре. Алгоритм KBF_GPU осуществляет просмотр всех подпоследовательностей заданного временного ряда посредством двух вложенных циклов, из которых внешний распараллеливается и выполняет вычисление суммы расстояний. В вычислительных экспериментах авторы, однако, сравнили свою разработку только с последовательным алгоритмом HOTSAX [86], которая ожидаемо показала существенно более высокое быстродействие.

В работе [143] Жу (Zhu) и др. предложили алгоритм параллельного поиска диссонансов на графическом процессоре. Авторы применили нормализованное евклидово расстояние и его быстрое вычисление на основе корреляции Пирсона с помощью техники, предложенной Муином (Mueen)

и др. в работе [98]. Для ускорения поиска авторы разработали два вычислительных шаблона. Первый шаблон предписывает следующую двухшаговую процедуру. Сначала вычисляется минимум расстояний между подпоследовательностью-кандидатом и всеми остальными подпоследовательностями временного ряда, которые не пересекаются с кандидатом. Затем осуществляется поиск кандидата, для которого достигается максимум расстояний по всем кандидатам. Второй шаблон предполагает ранний останов вычислений в рамках первого шаблона в случае, если расстояние между кандидатом и текущей рассматриваемой подпоследовательностью меньше, чем текущее лучшее (best-so-far) расстояние. В этом случае и кандидат, и текущая рассматриваемая подпоследовательность отбрасываются как заведомо не являющиеся диссонансами, и нет необходимости вычислять расстояния от кандидата до других не пересекающихся с ним подпоследовательностей. В вычислительных экспериментах [143] данный алгоритм опередил по быстродействию алгоритм SCAMP [147], который в настоящее время является наиболее быстрым алгоритмом вычисления матричного профиля. Однако, предложенные вычислительные шаблоны ограничивают результат поиска одним (хотя и наиболее важным) диссонансом временного ряда, в то время как применение концепции диапазонного диссонанса позволяет находить список top-k диссонансов, где k — параметр, наперед задаваемый экспертом в предметной области.

1.6. Выводы по главе 1

В данной главе приведен обзор современных методов поиска аномалий во временных рядах. Особое внимание уделено поиску аномальных подпоследовательностей (в противовес точечным аномалиям) временного ряда, поскольку соответствующая задача является существенно более сложной и востребованной на практике и потому служит предметом данного диссертационного исследования. Введены формальные определения и нотация для обозначения используемых в последующих главах понятий: временной ряд, подпоследовательность, диссонанс (аномалия) и др.

Приведена таксономия методов поиска аномалий во временных рядах: поиск с учителем, поиск без учителя и поиск с частичным привлечением учителя. Рассмотрены основные представители указанных групп (за исключением методов поиска с учителем, поскольку по своей природе слабо приспособлены к поиску аномалий, не задействованных на этапе обучения, и в настоящее время редко применяются на практике). Обзор показывает, что в настоящее время задача разработки моделей, методов и алгоритмов поиска аномалий в потоковых временных рядах является актуальной и остается предметом интенсивных научных исследований и практических разработок.

Обзор публикаций, посвященных последовательным и параллельным алгоритмам поиска аномалий на основе концепции диссонанса временного ряда, наиболее близко относящихся к теме диссертации, показывает следующее. Существующие алгоритмы DRAG и MERLIN поиска всех диссонансов фиксированной и произвольной длин соответственно представляют собой один из наиболее перспективных подходов к поиску аномальных подпоследовательностей временного ряда. Однако указанные алгоритмы являются последовательными и их распараллеливание могло бы существенно увеличить производительность поиска диссонансов. В то же время пока не разработаны соответствующие параллельные алгоритмы для массово распространенных в настоящее время графических процессоров и высокопроизводительных кластеров с вычислительными узлами на их основе. В соответствии с этим актуальной задачей является разработка эффективных параллельных алгоритмов поиска диссонансов для указанных платформ.

Глава 2. Параллельный алгоритм поиска диссонансов фиксированной длины для графических процессоров

Данная глава посвящена новому параллельному алгоритму поиска диссонансов фиксированной длины PD3 (Parallel DRAG-based Discord Discovery) для графических процессоров. Рассмотрены основные принципы распараллеливания алгоритма PD3 и приведены матрично-векторные структуры данных, которые позволили эффективно распараллелить вычисления на графическом процессоре. Описана параллельная реализация фаз, составляющих алгоритм PD3: предварительная обработка данных, отбор кандидатов в диссонансы и очистка диссонансов. Представлены результаты вычислительных экспериментов на реальных и синтетических временных рядах, в которых исследуется производительность алгоритма PD3 в сравнении с известными аналогами.

2.1. Принципы распараллеливания

Распараллеливание алгоритма PD3 основано на следующих основных принципах: параллелизм по данным, векторизация вычислений и выравнивание данных в памяти.

Параллелизм по данным реализуется с помощью *сегментации* временного ряда. Временной ряд разбивается на сегменты равной длины, каждый из которых обрабатывается отдельным блоком нитей графического процессора. Для обеспечения баланса загрузки нитей в блоке длине сегмента задается такое значение, чтобы количество подпоследовательностей длины m, принадлежащих сегменту, было кратно размеру варпа графического процессора. При запуске приложения блоки нитей распределяются для исполнения между потоковыми мультипроцессорами и выполняются

далее параллельно без возможности их синхронизации. Распределение блоков обеспечивает максимальную загрузку аппаратных средств.

Под *векторизацией вычислений* понимается способность компилятора заменить несколько скалярных операций в теле цикла с фиксированным количеством повторений в одну векторную операцию [17]. Векторизация вычислений, выполняемая на скалярных ядрах графического процессора, обеспечивается с помощью модели исполнения SIMT (Single Instruction Multiple Data) [64]. Внутри блока нити разбиваются на варпы (warp) — группы по 32 нити. Все нити варпа исполняют физически одну и ту же инструкцию над назначенной им частью разделяемых данных. При выполнении операции для считывания данных нити варпа обращаются к адресам памяти из диапазона в 128 или 256 байт. Если данные лежат в одном блоке (участке) памяти, то запросы к памяти объединяются в одну транзакцию. В алгоритме PD3 вычисления организуются таким образом, чтобы увеличить, насколько это возможно, количество векторизуемых циклов.

Выравнивание данных предполагает, что общее количество элементов подлежащего обработке массива в оперативной памяти кратно ширине векторного регистра (числу элементов, загружаемых в векторный регистр) [17]. Отсутствие выравнивания снижает эффективность векторизации циклов ввиду эффекта разделения цикла (loop peeling). Указанный эффект заключается в том, что компилятор разбивает цикл на три части, где первая и третья части итераций, соответствующих обращениям к памяти с начального адреса до первого выровненного адреса, и обращениям с последнего выровненного адреса до конечного адреса, векторизуются отдельно друг от друга. Для выравнивания данных в алгоритме PD3 предусмотрено дополнение временного ряда фиктивными элементами справа для того, чтобы длина ряда была кратна размеру варпа.

2.1.1. Структуры данных

Для хранения данных в оперативной памяти графического процессора алгоритм PD3 использует векторные структуры данных, что обеспечивает

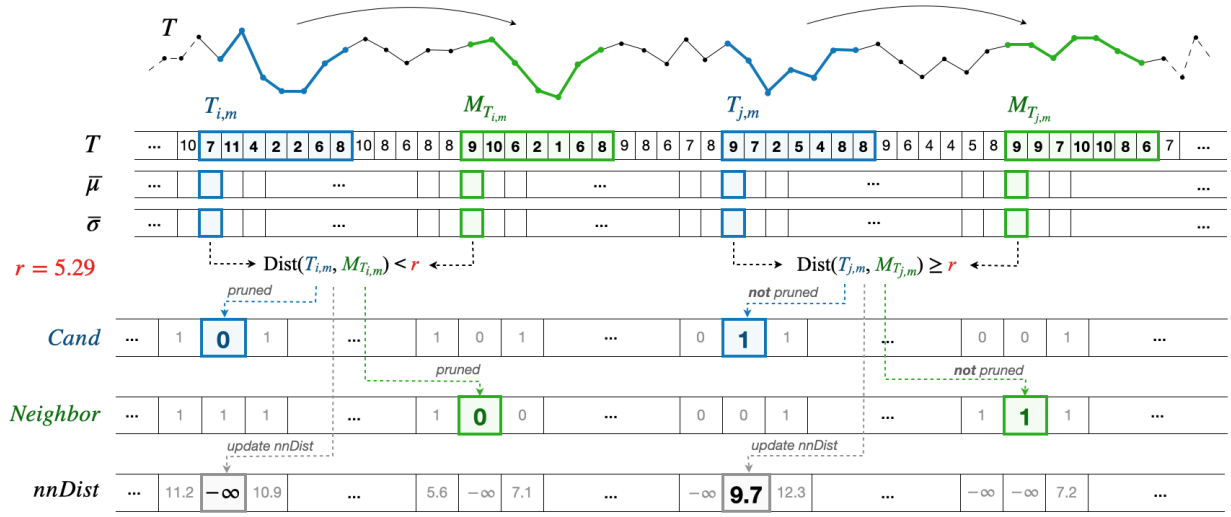


Рис. 2.1. Структуры данных алгоритма PD3

возможность векторизации распараллеливаемых вычислений. На рис. 2.1 представлены разработанные структуры данных для алгоритма PD3.

Вектора $\bar{\mu}, \bar{\sigma} \in \mathbb{R}^N$ хранят средние значения и значения стандартных отклонений всех подпоследовательностей исходного временного ряда, вычисляемые по формуле (1.6): $\bar{\mu}(i) = \mu_{T_{i,m}}$, $\bar{\sigma}(i) = \sigma_{T_{i,m}}$, соответственно.

Далее на основе векторов $\bar{\mu}$ и $\bar{\sigma}$ для каждой подпоследовательности вычисляется расстояние до ее ближайшего соседа. Все вычисленные расстояния хранятся в *рейтинге аномальности*, представляющем собой вектор $nnDist \in \mathbb{R}^N$, элемент которого выражает расстояние соответствующей подпоследовательности ряда до ее ближайшего соседа: $nnDist(i) = \min\limits_{T_{j,m} \in M_{T_{i,m}}} \mathrm{ED}^2_{\mathrm{norm}}(T_{i,m}, T_{j,m})$.

Два вектора $Cand, Neighbor \in \mathbb{B}^N$ являются битовыми картами для подпоследовательностей и их ближайших соседей соответственно. В указанных векторах i-й элемент принимает значение `TRUE`, если подпоследовательность $T_{i,m}$ и, соответственно, ее ближайший сосед является диссонансом, и `FALSE` в противном случае. Битовые карты инициализируется значениями `TRUE`. Кроме того, выполняется поэлементная конъюнкция приведенных выше битовых карт. Данная логическая операция позволяет отбросить большее количество кандидатов во время их обработки, основываясь

на том, что подпоследовательность, которая не является диссонансом, не может иметь ближайшего соседа, который является диссонансом.

2.1.2. Сегментация временного ряда

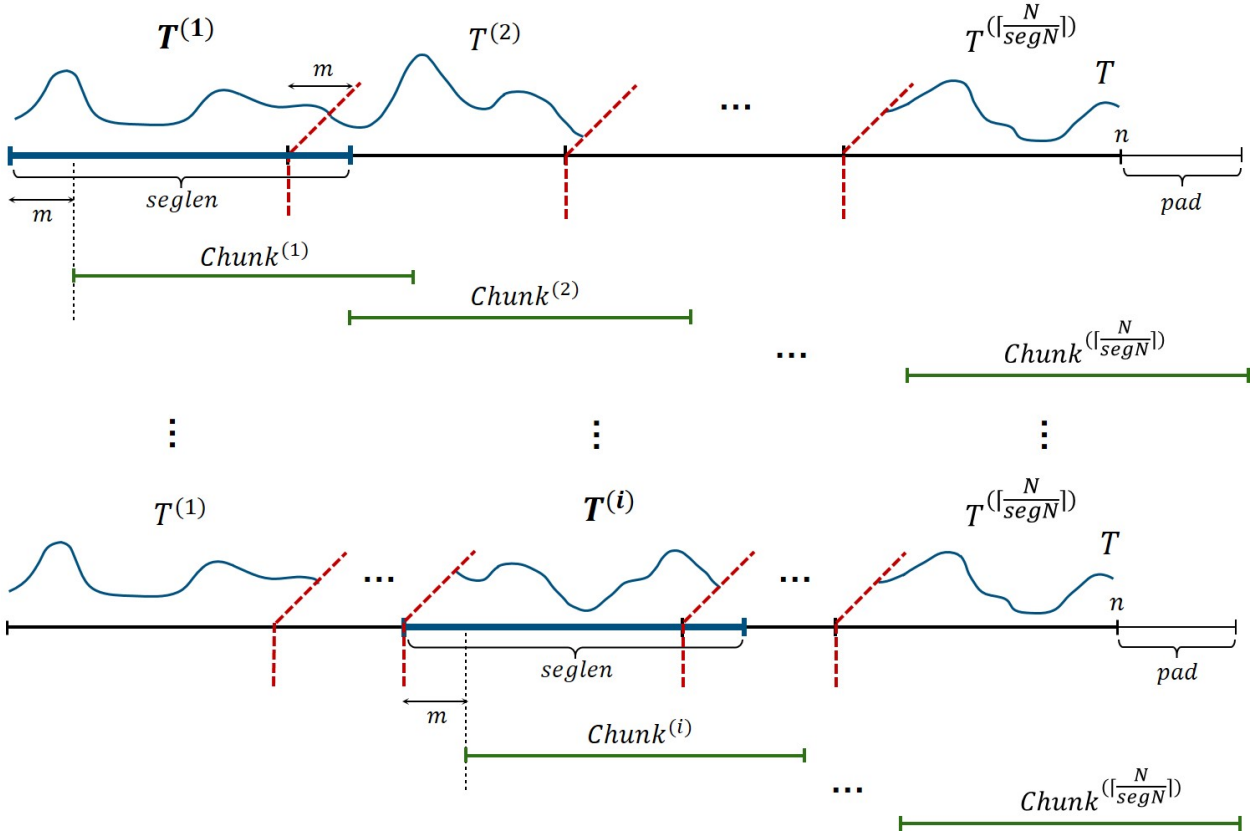


Рис. 2.2. Схема сегментирования ряда в алгоритме PD3

Для параллельной реализации фаз отбора и очистки кандидатов используется концепция параллелизма по данным, проиллюстрированная на рис. 2.2. Временной ряд делится на сегменты, имеющие равную длину, где каждый сегмент обрабатывается отдельным блоком нитей графического процессора. При выполнении каждой фазы блок нитей полагает подпоследовательности своего сегмента (локальными) кандидатами в диссонансы и выполняет сканирование и обработку подпоследовательностей ряда порциями. Количество элементов в порции равно длине сегмента. При этом первая порция таких элементов начинается с m-го элемента в сегменте.

Такая техника позволяет избежать избыточных проверок на пересечение кандидатов и подпоследовательностей порции.

Для предотвращения потери результирующих подпоследовательностей, находящихся на стыке сегментов (и порций соответственно), предлагается *техника разбиения с перекрытием*, которая заключается в следующем. В конец каждого сегмента временного ряда, за исключением последнего по порядку, добавляется $m - 1$ элементов ряда, взятых с начала следующего сегмента, где m — длина подпоследовательности. Формальное определение разбиения с перекрытием выглядит следующим образом.

Пусть N — количество подпоследовательностей длины m временного ряда T, S — количество сегментов, $T^{(i)}$ — i-й сегмент временного ряда T, где $1 \leqslant i \leqslant S$. Тогда сегмент $T^{(i)}$ определяется как подпоследовательность $T_{start,\,len}$, где

$$start = k \cdot \left\lfloor \frac{N}{S} \right\rfloor + 1, \tag{2.1}$$

$$len = \begin{cases} \lfloor \frac{N}{S} \rfloor + (N \bmod S) + m - 1, & k = S - 1 \\ \lfloor \frac{N}{S} \rfloor + m - 1, & \text{otherwise.} \end{cases} \tag{2.2}$$

Длина сегмента представляет собой параметр алгоритма и устанавливается кратной размеру варпа графического процессора. Для обеспечения баланса загрузки нитей в блоке необходимо, чтобы количество обрабатываемых подпоследовательностей ряда, имеющих длину m, было кратно количеству подпоследовательностей указанной длины в сегменте. Если таковая кратность не имеет место, то временной ряд дополняется справа фиктивными элементами, имеющими значение $+\infty$.

Для формализации описанной схемы выбора длины сегмента введем следующие обозначения. Длину сегмента обозначим за *seglen*. Мощность множества подпоследовательностей сегмента, имеющих длину m, обозначим *segN*, тогда $segN = seglen - m + 1$. Количество фиктивных элементов

в последнем справа сегменте ряда обозначим за pad, тогда

$$pad = \begin{cases} m-1, & N \bmod segN = 0 \\ \left\lceil \frac{N}{segN} \right\rceil \cdot segN + 2(m-1) - n, & \text{otherwise} \end{cases}. \qquad (2.3)$$

2.1.3. Параллельная предобработка данных

По сравнению с оригинальным последовательным алгоритмом в вычислительную схему алгоритма PD3 добавляется фаза предварительной обработки данных, за которой следуют фазы отбора и очистки кандидатов. Каждая фаза распараллеливается отдельно.

Фаза предварительной обработки данных предполагает параллельное вычисление на графическом процессоре средних значений и значений стандартных отклонений всех подпоследовательностей ряда в соответствии с формулой (1.6). Соответствующее CUDA-ядро формирует одномерную сетку из N нитей, в которой каждые *blocksize* нитей составляют отдельный блок, где *blocksize* является параметром алгоритма и устанавливается кратным размеру варпа графического процессора. Каждая нить выполняет вычисление одного элемента векторов $\bar{\mu}$ и $\bar{\sigma}$.

2.2. Распараллеливание фазы отбора кандидатов

Алг. 2.1 показывает реализацию параллельного отбора кандидатов. Соответствующее CUDA-ядро формирует одномерную сетку из $\lceil \frac{N}{segN} \rceil$ блоков, в каждом из которых $segN$ нитей. Блок нитей полагает подпоследовательности своего сегмента (локальными) кандидатами в диссонансы и выполняет сканирование и обработку подпоследовательностей ряда, которые не пересекаются с кандидатами и расположены *справа* от данного сегмента.

Обработка подпоследовательностей заключается в следующем. Если расстояние от кандидата до рассматриваемой подпоследовательности меньше, чем параметр r, то кандидат и подпоследовательность исключаются из дальнейшей обработки как заведомо не являющиеся диссонансами: соот-

ветствующие флаги битовых карт устанавливаются в значение `FALSE`. При этом, если все локальные кандидаты отброшены, блок досрочно завершает свою работу.

Алг. 2.1. PD3SELECT (IN T, m, r; OUT $\mathcal{C}$)

▷ Инициализация битовых карт

1: $Cand \leftarrow \overline{\texttt{TRUE}}$; $Neighbor \leftarrow \overline{\texttt{TRUE}}$

▷ Обработка сегментов

2: **for all** $T^{(i)} \in T$ **do**

▷ Сканирование порций, стоящих справа от сегмента

3: **for all** $Chunk^{(j)} \in T^{(i)}$ **where** $i \leqslant j$ **do**

▷ Обработка кандидатов и первой подпоследовательности порции

4: **if** $i = j$ **then**

5: $QTrow \leftarrow$ CALCDOTPRODUCTS($T^{(i)}_{1,\,m}$, $Chunk^{(j)}$)

6: **continue**

7: $QTrow \leftarrow$ UPDATEDOTPRODUCTS($QTrow, T^{(i)}_{1,\,m}$, $Chunk^{(j)}$)

8: $QTcol \leftarrow$ CALCDOTPRODUCTS($Chunk^{(j)}_{1,\,m}, T^{(i)}$)

9: $dist \leftarrow$ CALCDIST($Chunk^{(j)}_{1,\,m}, T^{(i)}$, $QTcol, \bar{\mu}, \bar{\sigma}$)

10: **if** $dist < r$ **then**

11: $Cand(i \cdot segN + tid) \leftarrow \texttt{FALSE}$; $Neighbor(j \cdot segN + 1) \leftarrow \texttt{FALSE}$

12: **else**

13: $nnDist(j \cdot segN + 1) \leftarrow \min\big(dist, nnDist(j \cdot segN + 1)\big)$

14: **if not** $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN} Cand(k)$ **then**

15: **break**

▷ Обработка кандидатов и остальных подпоследовательностей порции

16: **for all** $Chunk^{(j)}_{k,\,m} \in S^{m}_{Chunk^{(j)}} \setminus Chunk^{(j)}_{1,\,m}$ **do**

17: $QTcol \leftarrow$ UPDATEDOTPRODUCTS($QTcol$, $QTrow$, $Chunk^{(j)}_{k,\,m}, T^{(i)}$)

18: $dist \leftarrow$ CALCDIST($Chunk^{(j)}_{k,\,m}, T^{(i)}$, $QTcol, \bar{\mu}, \bar{\sigma}$)

19: **if** $dist < r$ **then**

20: $Cand(i \cdot segN + tid) \leftarrow \texttt{FALSE}$; $Neighbor(j \cdot segN + k) \leftarrow \texttt{FALSE}$

21: **else**

22: $nnDist(j \cdot segN + 1) \leftarrow \min\big(dist, nnDist(j \cdot segN + 1)\big)$

23: **if not** $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN} Cand(k)$ **then**

24: **break**

▷ Формирование множества кандидатов

25: $\mathcal{C} \leftarrow \big\{\{T_{i,\,m} \in S^{m}_{T};\ nnDist(i)\} \mid 1 \leqslant i \leqslant n - m + 1, Cand(i) = \texttt{TRUE}\big\}$

26: **return** $\mathcal{C}$

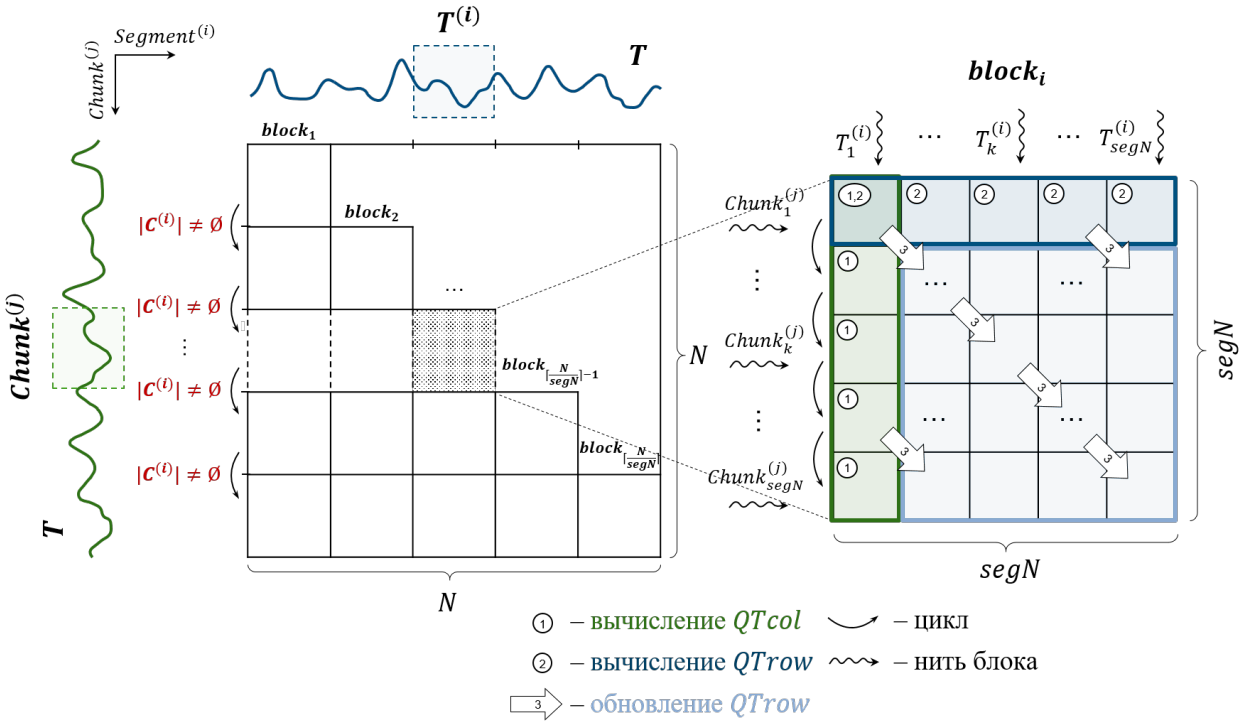


Рис. 2.3. Схема работы блока нитей в алгоритме PD3

Работа нитей внутри блока детализирована на рис. 2.3. Блок нитей однократно загружает в разделяемую память свой сегмент перед началом вычислений, а на каждом шаге сканирования загружает туда же текущую порцию, находящуюся справа от сегмента. Данный прием позволяет сократить количество обращений к глобальной памяти для чтения элементов временного ряда, что повышает быстродействие алгоритма. Далее нити блока вычисляют скалярные произведения, сохраняя результаты в разделяемой памяти: сперва между первой подпоследовательностью текущей порции и всеми подпоследовательностями сегмента, затем — между первой подпоследовательностью сегмента и всеми подпоследовательностями текущей порции (векторы $QTrow$ и $QTcol \in \mathbb{R}^{segN}$ в строках 4–8 в алг. 2.1 соответственно).

Далее на основе полученных результатов (вектор $QTrow$) и предварительно вычисленных векторов $\bar{\mu}$ и $\bar{\sigma}$ на основе формулы (1.8) вычисляются расстояния между первой подпоследовательностью порции и всеми подпоследовательностями сегмента (см. строку 9 в алг. 2.1). С помощью

вычисленного расстояния отбрасываются бесперспективные кандидаты в диссонансы, находящиеся в сегменте и текущей порции (см. строки 10–11 в алг. 2.1). Если кандидаты не были отброшены, выполняется обновление рейтинга аномальности кандидата, находящегося в сегменте (см. строку 13 в алг. 2.1). Если в итоге обработки сегмента отброшены все его кандидаты, то блок нитей досрочно завершает работу (см. строки 14–15 в алг. 2.1).

После этого нити выполняют сходные действия над оставшимися подпоследовательностями текущей порции, при этом, однако, вычисляя скалярные произведения более эффективно (см. строки 16–24 в алг. 2.1). Вычисление скалярных произведений между текущей подпоследовательностью обрабатываемой порции и всеми подпоследовательностями сегмента (вектор $QTrow$) выполняется на основе вычисленного ранее вектора $QTcol$ и вектора $QTrow$, вычисленного на предыдущей итерации (см. строку 17 в алг. 2.1). Каждая нить обрабатывает подпоследовательность сегмента, номер которой совпадает с номером нити в блоке. Нить, вычисляющая одно скалярное произведение между k-й ($1 < k \leqslant segN$) подпоследовательностью порции $Chunk^{(j)}$ и подпоследовательностью сегмента $T^{(i)}$, использует следующую формулу:

$$QTrow(tid) = \begin{cases} QTrow(tid-1) - T^{(i)}_{tid-1,\,m}(1) \cdot Chunk^{(j)}_{k-1,\,m}(1) \;+\; T^{(i)}_{tid,\,m}(m) \cdot Chunk^{(j)}_{k,\,m}(m), & 1 < tid \leqslant segN \\ QTcol(k), & tid = 1 \end{cases}, \tag{2.4}$$

где tid обозначает номер нити в блоке.

Результатом фазы отбора кандидатов является множество $\mathcal{C}$ тех подпоследовательностей, для которых в битовой карте $Cand$ элементы равны `TRUE` (см. строку 25 в алг. 2.1).

2.3. Распараллеливание фазы очистки кандидатов

Распараллеливание фазы очистки кандидатов реализуется в виде двух вызываемых друг за другом вычислительных CUDA-ядер. Первое CUDA-ядро выполняет поэлементную конъюнкцию векторов $Cand$ и $Neighbor$, за-

Алг. 2.2. PD3REFINE (IN T, m, r; OUT $\mathcal{D}$)

▷ Дополнительное отбрасывание кандидатов

1: **for all** $T_{i,m} \in S_T^m$ **do**
2: $\quad Cand(i) \leftarrow Cand(i) \wedge Neighbor(i)$

▷ Обработка сегментов с непустым множеством кандидатов

3: **for all** $T^{(i)} \in T$ **where** $\bigwedge_{k=i\cdot segN}^{(i+1)\cdot segN} Cand(k) = \texttt{TRUE}$ **do**

▷ Сканирование порций, стоящих слева от сегмента

4: $\quad$ **for all** $Chunk^{(j)} \in T^{(i)}$ **where** $i \geqslant j$ **do**

▷ Обработка кандидатов и первой подпоследовательности порции

5: $\quad\quad$ **if** $i = j$ **then**
6: $\quad\quad\quad QTrow \leftarrow$ CALCDOTPRODUCTS$(T_{1,m}^{(i)}, Chunk^{(j)})$
7: $\quad\quad\quad$ **continue**
8: $\quad\quad QTrow \leftarrow$ UPDATEDOTPRODUCTS$(QTrow, T_{1,m}^{(i)}, Chunk^{(j)})$
9: $\quad\quad QTcol \leftarrow$ CALCDOTPRODUCTS$(Chunk_{1,m}^{(j)}, T^{(i)})$
10: $\quad\quad dist \leftarrow$ CALCDIST$(Chunk_{1,m}^{(j)}, T^{(i)}, QTcol, \bar{\mu}, \bar{\sigma})$
11: $\quad\quad$ **if** $dist < r$ **then**
12: $\quad\quad\quad Cand(i\cdot segN + tid) \leftarrow \texttt{FALSE}$
13: $\quad\quad$ **else**
14: $\quad\quad\quad nnDist(j \cdot segN + tid) \leftarrow \min\big(dist, nnDist(j \cdot segN + tid)\big)$
15: $\quad\quad$ **if not** $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN} Cand(k)$ **then**
16: $\quad\quad\quad$ **break**

▷ Обработка кандидатов и остальных подпоследовательностей порции

17: $\quad\quad$ **for all** $Chunk_{k,m}^{(j)} \in S_{Chunk^{(j)}}^m \setminus Chunk_{1,m}^{(j)}$ **do**
18: $\quad\quad\quad QTcol \leftarrow$ UPDATEDOTPRODUCTS$(QTcol, QTrow, Chunk_{k,m}^{(j)}, T^{(i)})$
19: $\quad\quad\quad dist \leftarrow$ CALCDIST$(Chunk_{k,m}^{(j)}, T^{(i)}, QTcol, \bar{\mu}, \bar{\sigma})$
20: $\quad\quad\quad$ **if** $dist < r$ **then**
21: $\quad\quad\quad\quad Cand(i\cdot segN + tid) \leftarrow \texttt{FALSE}$
22: $\quad\quad\quad$ **else**
23: $\quad\quad\quad\quad nnDist(j \cdot segN + tid) \leftarrow \min\big(dist, nnDist(j \cdot segN + tid)\big)$
24: $\quad\quad$ **if not** $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN} Cand(k)$ **then**
25: $\quad\quad\quad$ **break**

▷ Формирование множества диссонансов

26: $\mathcal{D} \leftarrow \big\{\{T_{i,m} \in S_T^m;\ nnDist(i)\} \mid 1 \leqslant i \leqslant n - m + 1, Cand(i) = \texttt{TRUE}\big\}$
27: **return** $\mathcal{D}$

писывая ее результат в вектор *Cand*. Данная операция позволяет дополнительно отбросить подпоследовательности, которые являются ближайшими

соседями подпоследовательностей, отброшенных на фазе отбора кандидатов. Ядро организуется как одномерная сетка нитей из $\lceil \frac{N}{blocksize} \rceil$ блоков, состоящих из *blocksize* нитей, где каждая нить вычисляет один элемент результирующего вектора *Cand* (см. строки 1–2 в алг. 2.2).

Второе CUDA-ядро имеет идеологическое сходство с описанной выше процедурой параллельного отбора кандидатов. В очистке участвуют только те сегменты ряда, множество локальных кандидатов которых не пусто (см. строки 3 в алг. 2.2). Очистка кандидатов, принадлежащих некоторому сегменту, заключается в сканировании и обработке подпоследовательностей ряда, которые не пересекаются с кандидатами и расположены *слева* от данного сегмента (см. строку 4 в алг. 2.2). Если расстояние от кандидата до рассматриваемой подпоследовательности меньше, чем параметр r, то кандидат заведомо не входит в результирующее множество диссонансов и отбрасывается.

2.4. Вычислительные эксперименты

2.4.1. Описание экспериментов

Для проверки эффективности разработанного алгоритма PD3 проведены вычислительные эксперименты, в которых исследовалась производительность разработанного алгоритма. *Производительность* понимается как время работы алгоритма, в которое не включены накладные расходы на считывание данных с диска и их размещение в памяти и проч.

Выполнено сравнение алгоритма PD3 с конкурентами, рассмотренные выше в разделе 1.5. В качестве конкурентов взяты алгоритмы KBF_GPU [127] и алгоритм, представленный авторами Жу и др. в работе [143]. Авторы указанных алгоритмов не предоставляют их исходных текстов, поэтому для обеспечения справедливого сравнения в экспериментах алгоритм PD3 исследовался в идентичных экспериментальных условиях, что и алгоритмы-конкуренты: запускался на тех же аппаратных платформах и обрабатывал те же временные ряды с той же длиной диссонанса. Соот-

ветствующие сведения об аппаратуре и данных взяты из статей, в которых были предложены эти алгоритмы.

Результаты алгоритма PD3 сравнивались с теми, что опубликовали авторы алгоритмов-конкурентов в указанных выше работах. Кроме того, поскольку алгоритмы-конкуренты ограничивают поиск одним диссонансом заданной длины, а разработанный алгоритм PD3 выполняет нахождение всех диссонансов для заданных длины и параметра r, то выполняется сравнение среднего времени, затрачиваемое алгоритмами на поиск одного диссонанса. Во всех экспериментах параметры алгоритма PD3 устанавливались следующим образом. Длина сегмента (см. раздел 2.2) берется как $seglen = 512$. Подбор параметра r осуществлялся в соответствии с процедурой на основе применения алгоритма HOTSAX, описанной выше (см. раздел 1.2).

Табл. 2.1. Наборы данных экспериментов с алгоритмом PD3

Временной ряд	Длина ряда, n	Длина диссонанса, m	Предметная область
Space shuttle	5 000	150	Показания датчика на механизме космического корабля NASA
ECG	45 000	200	ЭКГ пациентов с болезнью сердца
ECG2	21 600	400	
ECG (mitdbx)	12 000	835	
ECG_24 (mitdbx)	24 000	835	
Koski-ECG	20 000	458	
Power demand	33 220	750	Годовое энергопотребление учреждения
POWER	20 000	965	
AEM	20 000	1043	Почасовое энергопотребление города
Respiration	24 125	250	Грудное дыхание человека
218cEEG	20 000	1093	Электроэнцефалограмма головного мозга

Временные ряды, задействованные в экспериментах, представлены в табл. 2.1 (указанные ряды из предметных областей индустрии и медицины использованы авторами алгоритмов-конкурентов в своих экспериментах и находятся в общедоступном архиве [37]). Временной ряд Space Shuttle содержит показания датчика тока соленоида, установленного на клапане космического корабля NASA [41]. Временные ряды ECG, ECG2, ECG (mitdbx) [47], Koski-ECG [72] представляют собой показания элек-

трокардиограмм взрослых пациентов с хронической сердечной недостаточностью. Ряд ECG_24 (mitdbx) является удвоенным рядом ECG (mitdbx). Временные ряды Power demand и POWER содержат показания энергопотребления исследовательского центра в Голландии, снятые в 1997 г. [135]. Временной ряд AEM представляет собой данные о почасовом потреблении электроэнергии некоторого города в Италии за 3 года, начиная с 1 января 1995 г. [68]. Временной ряд Respiration представляет собой показания дыхания человека по расширению грудной клетки [68]. Временной ряд 218cEEG содержит показания электроэнцефалограммы головного мозга человека [68].

Табл. 2.2. Аппаратная платформа экспериментов с алгоритмом PD3

Характеристика	GPU-SUSU	GPU-MSU
Производитель, семейство	NVIDIA Tesla	
Модель	V100	P100
Количество CUDA-ядер	5 120	3 584
Частота ядра, ГГц	1.3	1.19
Оперативная память, Гб	32	16
Пиковая производительность (двойная точность), TFLOPS	7	4

В табл. 2.2 приведены характеристики аппаратных платформ, на которых были проведены вычислительные эксперименты. Эксперименты по сравнению с алгоритмом KBF_GPU выполнялись на графическом процессоре GPU-SUSU узла вычислительного кластера «Нейрокомпьютер», установленного в Суперкомпьютерном центре Южно-Уральского государственного университета [2]. Эксперименты по сравнению с алгоритмом Жу и др. [143] выполнены на графическом процессоре GPU-MSU узла суперкомпьютера Ломоносов-2, установленного в Суперкомпьютерном центре Московского государственного университета [129].

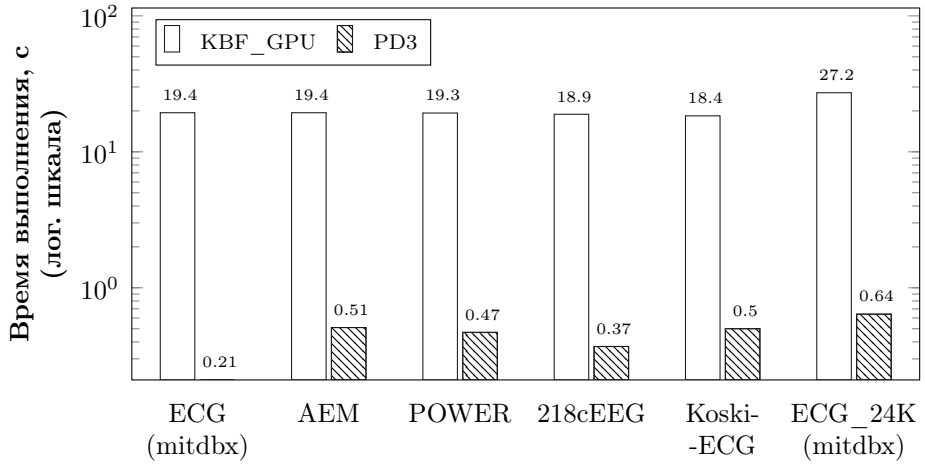


Рис. 2.4. Производительность алгоритма PD3 в сравнении с KBF_GPU

2.4.2. Анализ результатов

Результаты эксперимента по сравнению PD3 с KBF_GPU представлены на рис. 2.4. Можно видеть, что PD3 существенно (до двух порядков) опережает по производительности KBF_GPU. Это является ожидаемым, поскольку KBF_GPU лишь распараллеливает полный перебор подпоследовательностей, в то время как PD3 использует отбрасывание подпоследовательностей, продвинутые структуры данных и др.

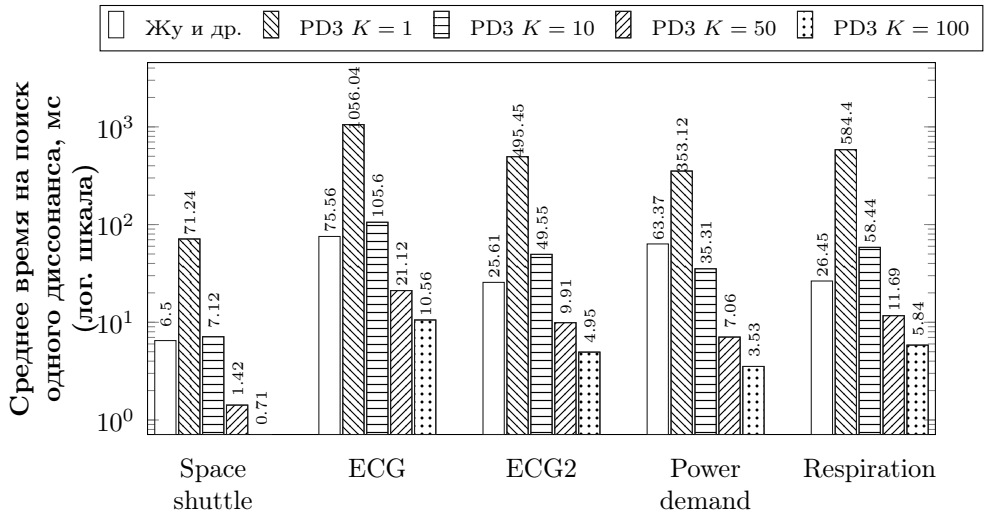


Рис. 2.5. Производительность алгоритма PD3 в сравнении с Жу и др.

Результаты экспериментов по сравнению PD3 с алгоритмом Жу и др. представлены на рис. 2.5. На диаграмме показано среднее время, затраченное алгоритмами на поиск одного диссонанса для различных временных рядов. Для алгоритма Жу и др. этот показатель совпадает с общим временем работы, поскольку данный алгоритм выполняет поиск одного (наиболее важного) диссонанса. Для алгоритма PD3 среднее время на поиск одного диссонанса показано для различных значений параметра K, количество искомых (наиболее важных) диссонансов: 1, 10, 50 и 100. Можно видеть, что при поиске одного диссонанса алгоритм Жу и др. существенно (на порядок) опережает PD3. Однако при увеличении параметра K мы видим изменение картины в пользу алгоритма PD3. Уже при поиске 10 диссонансов PD3 либо отстает от алгоритма-конкурента в 1.5–2 раза (см. ряды ECG, ECG2, Respiration), либо затрачивает в среднем примерно столько же времени на поиск одного диссонанса, что и алгоритм-конкурент (см. ряд Space Shuttle), либо опережает его в 1.8 раз (см. ряд Power demand). При поиске top-50 диссонансов PD3 опережает конкурента в 2–5 раз. При поиске top-100 диссонансов этот разрыв увеличивается от 5 до 17 раз. Соответственно, в приложениях, где требуется нахождение всех возможных аномалий временного ряда (а не наиболее важной из них), алгоритм PD3 будет более ценен, чем алгоритм Жу и др.

2.5. Выводы по главе 2

В этой главе предложен параллельный алгоритм PD3 (Parallel DRAG-based Discord Discovery) для поиска диссонансов фиксированной длины временного ряда на графическом процессоре, разработанный на основе алгоритма DRAG. В PD3 добавлена фаза предварительной обработки данных, на которой происходит вычисление средних значений и стандартных отклонений всех подпоследовательностей ряда. Полученные результаты далее применяются на фазах отбора и очистки кандидатов для вычисления расстояний между подпоследовательностями ряда. Каждая фаза алгорит-

ма распараллеливается отдельно на основе концепции параллелизма по данным и использования векторных структур данных.

Временной ряд делится на сегменты равной длины, каждый сегмент обрабатывается отдельным блоком нитей GPU. На фазе параллельного отбора кандидатов блок нитей полагает подпоследовательности своего сегмента (локальными) кандидатами в диссонансы и выполняет обработку подпоследовательностей ряда, которые расположены справа от данного сегмента и не пересекаются с кандидатами. Фаза параллельной очистки использует схожую технику, однако в ней участвуют только те сегменты ряда, множество локальных кандидатов которых не пусто, и обработке подлежат подпоследовательности ряда, которые расположены слева от данного сегмента. Обработка подпоследовательностей заключается в вычислении расстояния от кандидата до рассматриваемой подпоследовательности: если оно меньше, чем параметр r, то кандидат и подпоследовательность исключаются из дальнейшей обработки как заведомо не являющиеся диссонансами. В итоге оставшиеся не отброшенными подпоследовательности составляют результирующее множество искомых диссонансов исходного временного ряда.

В вычислительных экспериментах было выполнено сравнение производительности PD3 с двумя известными аналогами KBF_GPU [127] и алгоритм Жу и др. [143]. Поскольку указанные аналоги ограничивают поиск одним диссонансом, в то время как PD3 находит все диссонансы заданной длины, то сравнивается среднее время на поиск одного диссонанса.

Полученные результаты позволяют сделать вывод, что PD3 до двух порядков опережает по производительности KBF_GPU, поскольку KBF_-GPU распараллеливает полный перебор подпоследовательностей ряда. При сравнении с алгоритмом Жу и др. разработанный алгоритм PD3 опережает своего конкурента в 2–5 раз, когда выполняется поиск top-50 диссонансов. Таким образом, PD3 более ценен, чем аналоги, в приложениях, где требуется нахождение всех возможных аномалий временного ряда (а не наиболее важной из них).

Результаты этой главы опубликованы в работе [78].

Глава 3. Параллельный алгоритм поиска диссонансов произвольной длины для графических процессоров

В данной главе представлен новый параллельный алгоритм PALMAD (Parallel Arbitrary Length MERLIN-based Anomaly Discovery), предназначенный для поиска диссонансов временного ряда произвольной длины из заданного диапазона на графических процессорах. Описана общая схема вычислений алгоритм PALMAD, в которой задействуется представленный в предыдущей главе алгоритм PD3. Отличительными особенностями алгоритма PALMAD являются использование выведенных рекуррентных формул для сокращения объема вычислений и разработка векторных структур данных для эффективной параллельной обработки данных. Представлены результаты вычислительных экспериментов, исследующих эффективность предложенного алгоритма PALMAD. Исследовано применение алгоритма PALMAD на сенсорных данных, взятых из различных областей цифровой индустрии. Для визуализации найденных аномалий предложены новая техника тепловой карты диссонансов, имеющих различные длины, и алгоритм нахождения наиболее значимых диссонансов независимо от их длин на основе построенной тепловой карты.

3.1. Принципы распараллеливания

Псевдокод предлагаемого алгоритма PALMAD представлен в алг. 3.1. Алгоритм PALMAD в целом повторяет вычислительную схему оригинального последовательного алгоритма MERLIN на графическом процессоре. PALMAD многократно вызывает алгоритм PD3 (см. главу 2), являющийся параллельной версией последовательного алгоритма DRAG для графического процессора.

Алг. 3.1. PALMAD (IN T, $minL$, $maxL$, $topK$; OUT $\mathcal{D}$)

1: $\mathcal{D} \leftarrow \varnothing$; $r \leftarrow 2\sqrt{minL}$; $nnDist_{minL} \leftarrow -\infty$
2: $\{\bar{\mu}, \bar{\sigma}\} \leftarrow$ CalcMeanStd(S_T^{minL})
3: **while** $nnDist_{minL} < 0$ **and** $|\mathcal{D}| < topK$ **do**
4: $\quad D_{minL} \leftarrow \text{PD3}(T, \bar{\mu}, \bar{\sigma}, r^2)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_{minL}$; $nnDist_{minL} \leftarrow \min\limits_{d \in D_{minL}} d.nnDist$
5: $\quad r \leftarrow 0.5 \cdot r$
6: **for** $i \leftarrow minL + 1$ **to** $minL + 4$ **do**
7: $\quad nnDist_i \leftarrow -\infty$
8: $\quad \{\bar{\mu}, \bar{\sigma}\} \leftarrow$ UpdateMeanStd($S_T^i, \bar{\mu}, \bar{\sigma}$)
9: $\quad$ **while** $nnDist_i < 0$ **and** $|\mathcal{D}| < topK$ **do**
10: $\quad\quad r \leftarrow 0.99 \cdot nnDist_{i-1}$
11: $\quad\quad D_i \leftarrow \text{PD3}(T, \bar{\mu}, \bar{\sigma}, r^2)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_i$; $nnDist_i \leftarrow \min\limits_{d \in D_i} d.nnDist$
12: $\quad\quad r \leftarrow 0.99 \cdot r$
13: **for** $i \leftarrow minL + 5$ **to** $maxL$ **do**
14: $\quad \mu \leftarrow \text{Mean}(\{nnDist_k\}_{k=i-1}^{i-5})$; $\sigma \leftarrow \text{Std}(\{nnDist_k\}_{k=i-1}^{i-5})$; $r \leftarrow \mu - 2\sigma$
15: $\quad \{\bar{\mu}, \bar{\sigma}\} \leftarrow$ UpdateMeanStd($S_T^i, \bar{\mu}, \bar{\sigma}$)
16: $\quad D_i \leftarrow \text{PD3}(T, \bar{\mu}, \bar{\sigma}, r^2)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_i$; $nnDist_i \leftarrow \min\limits_{d \in D_i} d.nnDist$
17: $\quad$ **while** $nnDist_i < 0$ **and** $|\mathcal{D}| < topK$ **do**
18: $\quad\quad D_i \leftarrow \text{PD3}(T, \bar{\mu}, \bar{\sigma}, r^2)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_i$; $nnDist_i \leftarrow \min\limits_{d \in D_i} d.nnDist$
19: $\quad\quad r \leftarrow r - \sigma$
20: **return** $\mathcal{D}$

В отличие от исходного алгоритма, PALMAD позволяет избежать избыточных вычислений при итерационных вызовах DRAG, когда длина подпоследовательности на единицу больше, чем на предыдущем шаге. Действительно, чтобы вычислить расстояние между любыми двумя подпоследовательностями-кандидатами, необходимо частично повторить те же вычисления для подпоследовательностей этого ряда, длина которых на единицу меньше. Формально говоря, при вычислении $\text{ED}^2_{\text{norm}}(T_{i,m}, T_{j,m})$ по формуле (1.8) для любых i, j ($1 < i, j \leqslant n - m$ и $3 \leqslant m \ll n$) вычисления средних арифметических и стандартных отклонений подпоследовательностей $T_{i,m-1}$ и $T_{j,m-1}$ по формуле (1.6) частично повторяются.

Чтобы избежать вышеописанных накладных расходов, в алгоритме PALMAD используются векторы средних арифметических и стандартных отклонения всех подпоследовательностей временного ряда заданной длины

$\bar{\mu}, \bar{\sigma} \in \mathbb{R}^{n-minL+1}$ соответственно. В данных векторах обработке подлежат первые $n - m + 1$ элементов, где m — заданная длина подпоследовательности ($minL \leqslant m \leqslant maxL$), а остальные остаются неизменными.

Для подпоследовательностей длины $m = minL$ векторы $\bar{\mu}$, $\bar{\sigma}$ вычисляются в соответствии с формулой (1.6) один раз перед первым вызовом PD3, тогда как для остальных значений m векторы $\bar{\mu}$ и $\bar{\sigma}$ обновляются перед каждым последующим вызовом PD3. Далее докажем утверждение, в котором выводятся рекуррентные формулы, предназначенные для обновления средних арифметических и стандартных отклонений подпоследовательностей временного ряда.

Утверждение. Пусть имеется временной ряд T, $|T| = n$, и две его подпоследовательности $T_{i,\,m}$ и $T_{i,\,m+1}$, имеющие длины m и $m+1$ соответственно, где $1 \leqslant i \leqslant n - m$ и $3 \leqslant m \leqslant n$. Тогда среднее арифметическое $\mu_{T_{i,\,m+1}}$ и стандартное отклонение $\sigma^2_{T_{i,\,m+1}}$ подпоследовательности $T_{i,\,m+1}$ вычисляются по следующим рекуррентным формулам:

$$\mu_{T_{i,\,m+1}} = \frac{1}{m+1}\big(m\mu_{T_{i,\,m}} + t_{i+m}\big), \tag{3.1}$$

$$\sigma^2_{T_{i,\,m+1}} = \frac{m}{m+1}\Big(\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(\mu_{T_{i,\,m}} - t_{i+m}\big)^2\Big). \tag{3.2}$$

Доказательство. Сначала докажем формулу (3.1) для нахождения среднего арифметического $\mu_{T_{i,\,m+1}}$ подпоследовательности $T_{i,\,m+1}$ на основе ранее вычисленного $\mu_{T_{i,\,m}}$ подпоследовательности $T_{i,\,m}$.

Согласно определению 1.6

$$\mu_{T_{i,\,m}} = \frac{1}{m}\sum_{k=0}^{m-1} t_{i+k}. \tag{3.3}$$

Из (3.3) получаем

$$\sum_{k=0}^{m-1} t_{i+k} = m\mu_{T_{i,\,m}}. \tag{3.4}$$

Далее вычислим $\mu_{T_{i,\,m+1}}$ по формуле (1.6), подставляя в нее правую часть равенства 3.4:

$$\mu_{T_{i,\,m+1}} = \frac{1}{m+1}\sum_{k=0}^{m} t_{i+k} = \frac{1}{m+1}\Big(\sum_{k=0}^{m-1} t_{i+k} + t_{i+m}\Big) = \frac{1}{m+1}(m\mu_{T_{i,\,m}} + t_{i+m}). \tag{3.5}$$

Таким образом, первая формула (3.1) доказана.

Далее приведем доказательство формулы (3.2) для нахождения квадрата стандартного отклонения $\sigma^2_{T_{i,\,m+1}}$ подпоследовательности $T_{i,\,m+1}$, выраженного через $\sigma^2_{T_{i,\,m}}$.

В соответствии с определением 1.6

$$\sigma^2_{T_{i,\,m}} = \frac{1}{m}\sum_{k=0}^{m-1} t^2_{i+k} - \mu^2_{T_{i,\,m}}. \tag{3.6}$$

Тогда

$$\sum_{k=0}^{m-1} t^2_{i+k} = m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}). \tag{3.7}$$

Далее для подпоследовательности $T_{i,\,m+1}$ вычислим $\sigma^2_{T_{i,\,m+1}}$, используя формулу (1.6) и подставляя в нее правую часть равенства (3.7):

$$\sigma^2_{T_{i,\,m+1}} = \frac{1}{m+1}\sum_{k=0}^{m} t^2_{i+k} - \mu^2_{T_{i,\,m+1}} = \frac{1}{m+1}\Big(m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}) + t^2_{i+m}\Big) - \mu^2_{T_{i,\,m}}. \tag{3.8}$$

Воспользовавшись доказанной выше формулой (3.5) для нахождения среднего значения $\mu_{T_{i,\,m+1}}$ подпоследовательности $T_{i,\,m+1}$, получим:

$$\sigma^2_{T_{i,\,m+1}} = \frac{1}{m+1}\Big(m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}) + t^2_{i+m}\Big) - \Big(\frac{1}{m+1}(m\mu_{T_{i,\,m}} + t_{i+m})\Big)^2. \tag{3.9}$$

Далее, раскрывая скобки и приводя подобные слагаемые, получаем

$$\begin{aligned}\sigma^2_{T_{i,\,m+1}} &= \frac{1}{m+1}\Big(m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}) + t^2_{i+m} - \frac{1}{m+1}(m\mu_{T_{i,\,m}} + t_{i+m})^2\Big) = \\ &= \frac{1}{m+1}\Big(m\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(m(m+1)\mu^2_{T_{i,\,m}} - m^2\mu_{T_{i,\,m}} + \\ &\quad + (m+1)t^2_{i+m} - t^2_{i+m} - 2m\mu_{T_{i,\,m}}t_{i+m}\big)\Big) = \\ &= \frac{1}{m+1}\Big(m\sigma^2_{T_{i,\,m}} + \frac{m}{m+1}\big(\mu^2_{T_{i,\,m}} - 2\mu_{T_{i,\,m}}t_{i+m} + t^2_{i+m}\big)\Big) = \\ &= \frac{m}{m+1}\Big(\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(\mu_{T_{i,\,m}} - t_{i+m}\big)^2\Big).\end{aligned}$$

Таким образом, вторая формула (3.2) также доказана. □

Инициализация и обновление векторов $\bar{\mu}$ и $\bar{\sigma}$ реализуются в виде CUDA-ядер, формирующих одномерную сетку из N нитей, в которой каждые *blocksize* нитей составляют отдельный блок, где *blocksize* является параметром алгоритма и устанавливается кратным размеру варпа графического процессора. Каждая нить выполняет вычисление одного элемента векторов $\bar{\mu}$ и $\bar{\sigma}$: инициализация по формуле (1.6), последующее их обновление по рекуррентным формулам (3.1) и (3.2).

При этом указанные выше CUDA-ядра отличаются друг от друга тем, что при инициализации каждая нить блока считывает всю подпоследовательность временного ряда из глобальной памяти графического процессора, в то время как при обновлении нить считывает только один элемент временного ряда вместо m. Сокращение количества обращений к глобальной памяти графического процессора достигается благодаря выведенным рекуррентным формулам.

3.2. Алгоритм ранжирования диссонансов

Визуализация является важной и неотъемлемой частью решения задач интеллектуального анализа данных, поскольку обеспечивает аналитику наглядное представлением исследуемых данных и результатов анализа,

являющееся основой выявления скрытых закономерностей для принятия стратегически важных решений [50].

В данном разделе представлена новая техника визуализации аномалий временного ряда в виде тепловой карты диссонансов, в которой степень аномальности диссонанса отражается посредством интенсивности цвета. Далее в разделах 3.2.1 и 3.2.2 описаны детали построения тепловой карты и способ нахождения наиболее значимых аномалий на основе построенной карты соответственно.

3.2.1. Построение тепловой карты диссонансов

Пусть имеется временной ряд T длины n, в котором с помощью алгоритма PALMAD осуществляется поиск диссонансов длины m, принимающей значения в заданном диапазоне $minL \leqslant m \leqslant maxL$ ($minL \leqslant maxL \ll n$). По завершении работы алгоритма PALMAD получим итоговое множество диссонансов $\mathcal{D} = \bigcup_{m=minL}^{maxL} D_m$, где D_m — подмножество диссонансов, имеющих длину m.

Далее рассмотрим матричный профиль временного ряда T для длины подпоследовательности m. *Матричный профиль (matrix profile)* [142] временного ряда T для длины подпоследовательности m представляет собой временной ряд $MP_m \in \mathbb{R}^{n-m+1}$, элементами которого являются расстояния от соответствующей подпоследовательности ряда до ближайшего соседа:

$$\begin{gathered} MP_m(i) = \mathrm{Dist}(T_{i,\,m},\ Neighbor), \\ Neighbor = \arg \min_{T_{j,\,m} \in D_{T_{i,\,m}}} \mathrm{Dist}(T_{i,\,m},\ T_{j,\,m}). \end{gathered} \tag{3.10}$$

Затем возьмем матричные профили ряда T для всех длин подпоследовательности из диапазона $minL..maxL$ в порядке возрастания длин и построим из них матрицу профилей $MMP \in \mathbb{R}^{(maxL-minL+1)\times(n-minL)}$. В качестве строки указанной матрицы фигурирует отдельный матричный профиль ряда, где обнулены элементы, соответствующие подпоследовательностям ряда, которые не являются диссонансами (индексы строки и столбца мат-

рицы показывают соответственно длину диссонанса и его индекс в ряде):

$$MMP(m,\, i) = \begin{cases} MP_{minL+m-1}(i), & T_{i,\, minL+m-1} \in D_{minL+m-1} \\ 0, & \text{otherwise.} \end{cases} \tag{3.11}$$

Отметим, что матрица *MMP* является результатом работы алгоритма PALMAD и не требует отдельных вычислений матричных профилей ряда для рассматриваемых длин подпоследовательности (например, с помощью алгоритма [147]).

Определим *тепловую карту диссонансов (discord heatmap)* как матрицу $heatmap \in \mathbb{R}^{(maxL-minL+1)\times(n-minL)}$, получаемую нормированием элементов в каждой строке матрицы $MMP(m, \cdot)$ с помощью множителя $\frac{1}{2m}$:

$$heatmap(m,\, i) = \frac{MMP(m,\, i)}{2m}. \tag{3.12}$$

Указанный нормирующий множитель обеспечивает приведение значений элементов тепловой карты к диапазону от 0 до 1, поскольку доказано [98], что положительная корреляция по Пирсону двух векторов $x,\, y \in \mathbb{R}^m$ и нормализованное евклидово расстояние между ними связаны следующим соотношением:

$$PearsonCorr(x, y) = 1 - \frac{\mathrm{ED}^2_{\mathrm{norm}}(x,\, y)}{2m}. \tag{3.13}$$

Таким образом, каждый элемент $heatmap(m,\, i)$ представляет собой оценку аномальности диссонанса $T_{i,\, m} \in D_m$, нормированную по всем диссонансам множества $\mathcal{D}$. В тепловой карте диссонансов используется один цвет (например, красный), и интенсивность цвета в пикселе (m, i) прямо пропорциональна оценке диссонанса $T_{i,\, m} \in D_m$. При этом нормирование обеспечивает на одной тепловой карте корректную визуализацию оценок аномальности диссонансов различной длины.

Алг. 3.2. TopInterestDiscords (in: $heatmap$, K; out: $InterestD$)

1: $Scores \leftarrow \{score_i\}_{i=1}^{n-minL+1}$, где $score_i = \max\limits_{minL \leqslant m \leqslant maxL} heatmap(m, i)$
2: $Lengths \leftarrow \{length_i\}_{i=1}^{n-minL+1}$, где $length_i = \arg\max\limits_{minL \leqslant m \leqslant maxL} heatmap(m, i)$
3: $Indexes \leftarrow \{i\}_{i=1}^{n-minL+1}$
4: Sort($Scores$, $Lengths$, $Indexes$) ▷ Сортировка векторов по убыванию значений в $Scores$
5: $InterestD \leftarrow \{(T_{Indexes(1),\, Lengths(1)},\ Scores(1))\}$; $k \leftarrow 2$
6: **while** $(|InterestD| < K)$ **or** $(k < N)$ **do**
7: $score \leftarrow Scores(k)$; $length \leftarrow Lengths(k)$; $index \leftarrow Indexes(k)$
8: **for each** $T_{i,\,m} \in InterestD$ **do**
9: **if** $\min(length, m) < |i - index| < \max(length, m)$ **then** ▷ Проверка пересечения
10: $InterestD \leftarrow InterestD \cup \{(T_{index,\, length}, score)\}$
11: **else**
12: **break**
13: $k \leftarrow k + 1$
14: **return** $InterestD$

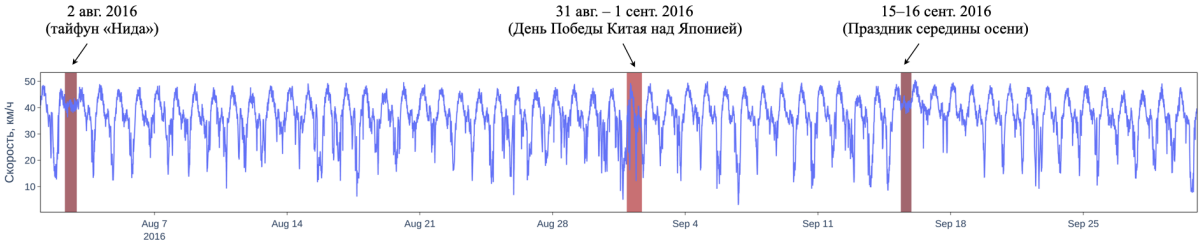
3.2.2. Ранжирование диссонансов различной длины

Тепловая карта диссонансов, описанная выше, представляет собой удобный для аналитика инструмент визуализации диссонансов заданного временного ряда, имеющих различную длину. Тем не менее, для аналитика также важно ранжирование найденных диссонансов по их практической значимости вне зависимости от их длины. Далее представлен алгоритм нахождения top-k наиболее значимых диссонансов, который основан на простой идее отдавать предпочтение диссонансам, которые имеют бóльшую оценку, исключая при этом пересекающиеся диссонансы (см. алг. 3.2).

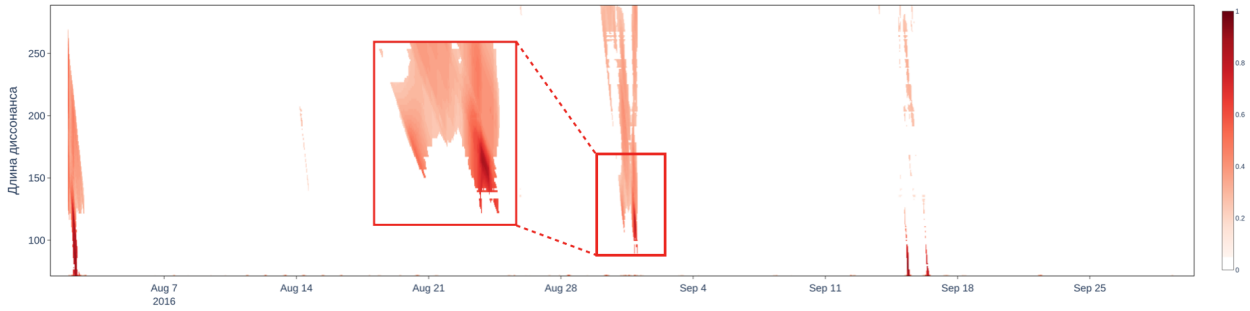
Алгоритм получает на входе тепловую карту диссонансов $heatmap \in \mathbb{R}^{(maxL-minL+1)\times(n-minL)}$ и число K искомых диссонансов, а на выходе формирует множество $InterestD$, элементами которого являются пары вида $(T_{i,\,m}, score)$, где диссонанс $T_{i,\,m} \in D_m$, $score$ — оценка диссонанса. Алгоритм выполняется следующим образом. Сначала формируются векторы $Scores, Lengths \in \mathbb{R}^{n-minL+1}$, в которых i-й элемент хранит соответственно

максимальную оценку диссонанса с индексом i и длину такого диссонанса. Вектор $Indexes \in \mathbb{R}^{n-minL+1}$ хранит индексы диссонансов. Далее вектор $Scores$ сортируется по убыванию, а векторы $Lengths$ и $Indexes$ упорядочиваются на основе результата такой сортировки. В итоге указанные три вектора совместно представляют сведения о диссонансах множества $\mathcal{D}$ в порядке убывания оценок входящих в него диссонансов (строки 1–4 в алг. 3.2). Искомое множество диссонансов инициализируется диссонансом с максимальной оценкой (строка 5 в алг. 3.2). Затем выполняется просмотр диссонансов множества $\mathcal{D}$ в полученном порядке (цикл в строках 6–13 в алг. 3.2). Если рассматриваемый диссонанс не пересекается ни с одним элементом из $Interest\mathcal{D}$, то он добавляется в него. Сканирование продолжается до тех пор, пока не найдено K наиболее значимых диссонансов или исчерпано множество $\mathcal{D}$.

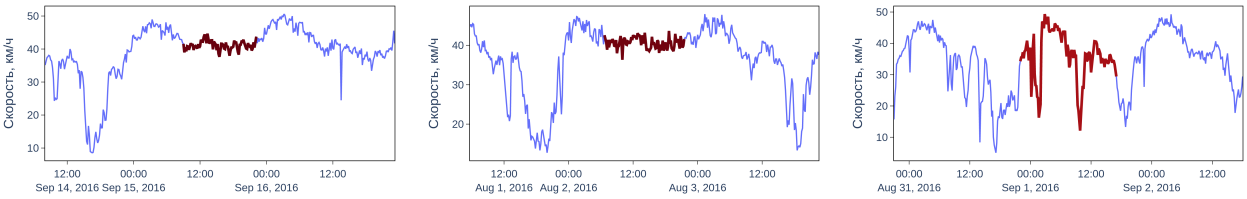
Проиллюстрируем работу описанного алгоритма на следующем реальном примере временного ряда [34], который содержит данные о скорости городского трафика на одном из участков городской автомагистрали Гуанчжоу (Китай), измерявшиеся каждые 10 мин. с 1 августа по 30 сентября 2016 г. (см. рис. 3.1a). В ряде выделены красным цветом диссонансы различной длины, которые были найдены с помощью алгоритма PALMAD. На рис. 3.1b показана тепловая карта найденных диссонансов. Далее, на рис. 3.1c показаны три наиболее важных диссонанса (имеющие различную длину), отобранные с помощью описанного выше алгоритма TopInterestDiscords. Можно видеть, что у двух из указанных диссонансов время возникновения совпадает с нетипичными событиями: тайфун Нида, отмечаемый в Китае Праздник середины осени (фестиваль Луны) и длинные выходные в связи с празднованием Дня Победы Китая над Японией.



(a) Временной ряд



(b) Тепловая карта диссонансов



Top-1 диссонанс, $m = 78$ Top-2 диссонанс, $m = 87$ Top-3 диссонанс, $m = 116$

(c) Найденные top-3 интересных диссонанса различных длин

Рис. 3.1. Тепловая карта диссонансов, построенная алгоритмом Top-InterestDiscords

3.3. Вычислительные эксперименты

Для оценки эффективности разработанного алгоритма были проведены вычислительные эксперименты, направленные на сравнение производительности PALMAD с известными аналогами на реальных и синтетических временных рядах, а также на исследование масштабируемости PALMAD. Ниже в разделе 3.3.1 приводятся технические характеристики аппаратных платформ и описание временных рядов, используемых в экспериментах, в разделе 3.3.2 представлены результаты экспериментов и их обсуждение.

Табл. 3.1. Наборы данных экспериментов с алгоритмом PALMAD

Временной ряд	Длина ряда, n	Длина диссонанса, $minL = maxL$	Предметная область
Space shuttle	50 000	150	Показания датчика, установленного на механизме космического корабля NASA
ECG	45 000	200	Показания ЭКГ пациента с хронической сердечной недостаточностью
ECG-2	21 600	400	
Koski-ECG	100 000	458	
Respiration	24 125	250	Грудное дыхание человека
Power demand	33 220	750	Годовое энергопотребление учреждения
RandomWalk1M	10^7	512	Синтетические временные ряды на основе модели случайного блуждания
RandomWalk2M	$2 \cdot 10^7$	512	

3.3.1. Описание экспериментов

Для проведения экспериментов были использованы временные ряды, резюмированные в табл. 3.1. Данные ряды также были взяты авторами алгоритмов HOTSAX [68], KBF_GPU [127] и алгоритма Жу и др. [143] для экспериментальной оценки их эффективности.

Ряд Space shuttle [41] представляет собой измерения тока соленоида, которым оснащен клапан Marotta MPV-41, циклически включающийся и выключающийся во время проведения лабораторных испытаний в различных условиях. Данный клапан используется для управления подачи топлива на космический корабль NASA.

Временные ряды ECG, ECG2 [47] и Koski-ECG [72] представляют собой электрокардиограммы взрослых пациентов, которые имеют хроническую сердечную недостаточность.

Временной ряд Respiration [68] содержит показания дыхания пациента, снимаемые при расширении грудной клетки, в момент пробуждения человека.

Временной ряд Power demand содержит данные о потреблении электроэнергии исследовательским центром в Нидерландах за 1997 г. [135].

Ряды RandomWalk1M и RandomWalk2M являются синтетическими и сгенерированы с помощью модели случайного блуждания [108].

Эксперименты проведены на графических процессорах GPU-SUSU и GPU-MSU (см. табл. 2.2), которыми оснащены узлы вычислительных си-

стем, установленных в Суперкомпьютерных центрах Южно-Уральского государственного университета [2] и Московского государственного университета [129] соответственно.

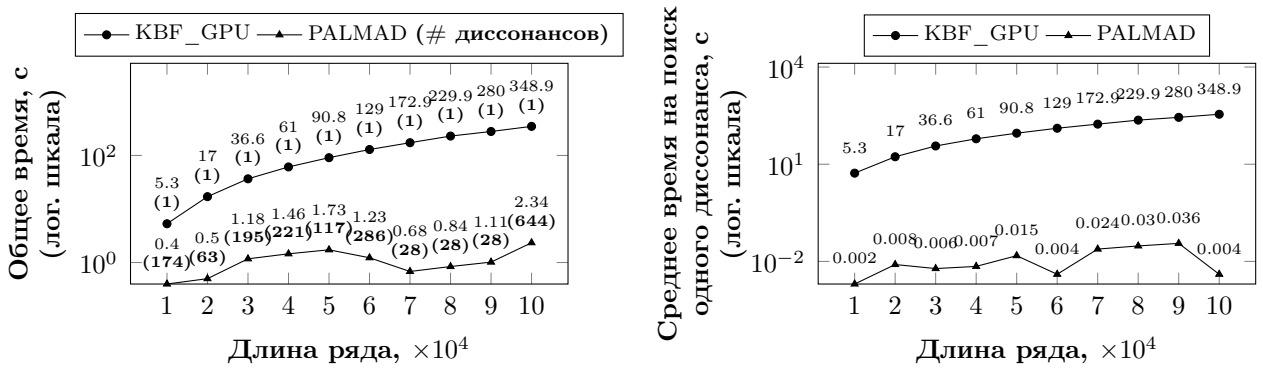
3.3.2. Анализ результатов

Сравнение с аналогами

В первой серии экспериментов выполнялось сравнение PALMAD с известными аналогами. В качестве аналогов были выбраны два алгоритма, KBF_GPU [127] и алгоритм Жу и др. [143], поскольку обзор релевантных работ (см. раздел 1.5.3) не выявил других параллельных алгоритмов, ориентированных на графические процессоры. Поскольку авторы вышеперечисленных алгоритмов-аналогов не предоставляют свои исходные коды, для объективного сравнения в экспериментах используются временные ряды и аппаратные платформы, идентичные тем, которые рассматриваются в [127] и [143] соответственно и сравниваются полученные результаты с результатами, представленными в оригинальных статьях авторов. Временной ряд Koski-ECG (см. табл. 3.1) использовался для сравнения PALMAD с KBF_GPU, запущенными на GPU-SUSU (см. табл. 2.2), а остальные временные ряды использовались для сравнения с алгоритмом Жу и др. на GPU-MSU. Для каждого эксперимента алгоритм PALMAD запускался 10 раз. В качестве окончательного времени выполнения принималось среднее значение, вычисленное по всем запускам без учета затрат на чтение исходных данных и запись полученных результатов.

Поскольку аналоги выполняют поиск только top-1 диссонанса, в то время как PALMAD находит диапазонные диссонансы каждой длины в указанном диапазоне длин, чтобы обеспечить справедливое сравнение, в экспериментах устанавливаются следующие параметры алгоритма PALMAD. Во-первых, для диапазона длин диссонансов значение максимальной длины приравнивается к значению минимальной длины, $minL = maxL$. Во-вторых, помимо измерения общего времени выполнения PALMAD, подсчи-

тывается количество найденных диссонансов, чтобы дополнительно вычислить среднее время, затрачиваемое PALMAD на обнаружение одного диссонанса.



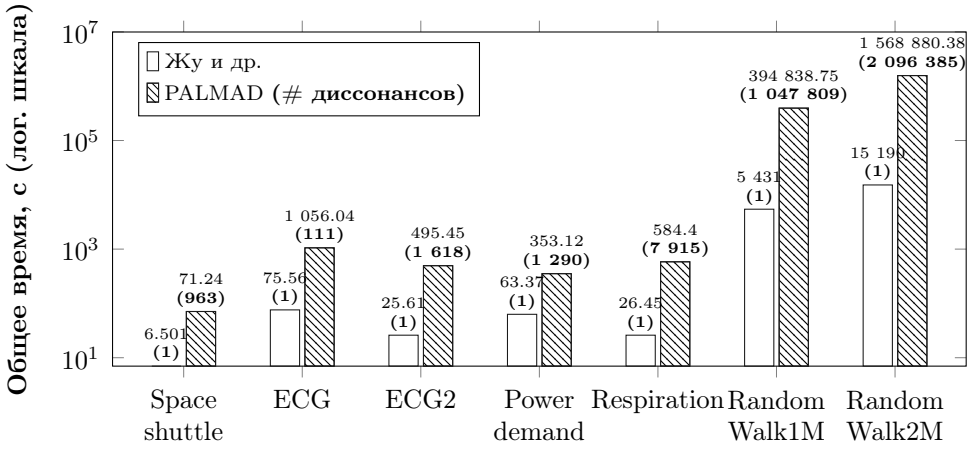
(a) Общее время выполнения

(b) Среднее время выполнения на поиск одного диссонанса

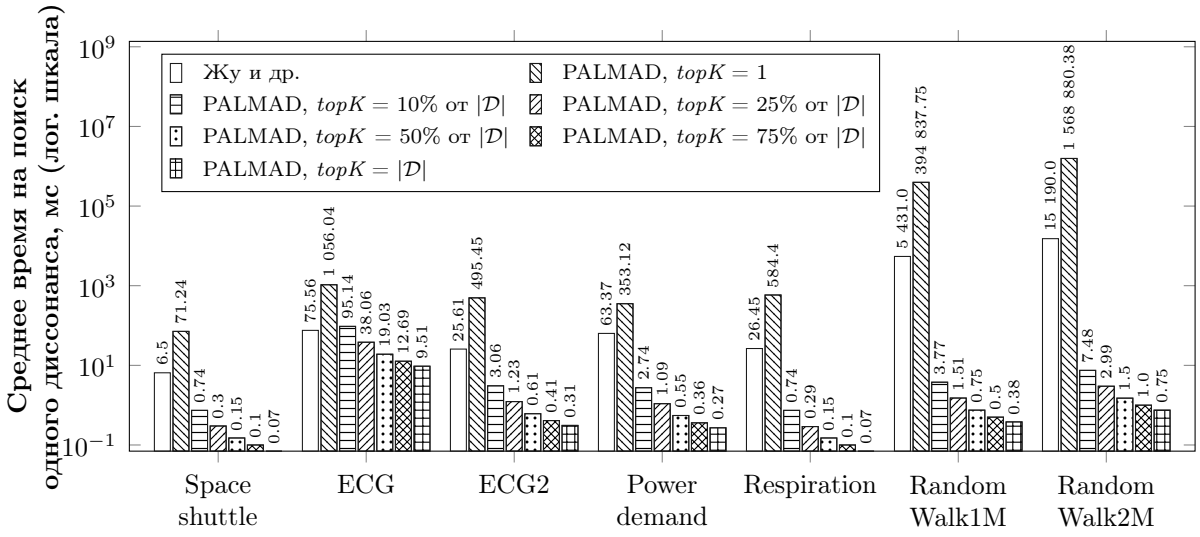
Рис. 3.2. Производительность алгоритма PALMAD в сравнении с KBF_GPU

На рис. 3.2 приведены результаты экспериментов по сравнению производительности алгоритмов PALMAD и KBF_GPU. Можно видеть, что PALMAD существенно опережает конкурента как по общему времени работы, так и по среднему времени выполнения работы на поиск одного диссонанса. Полученные результаты можно объяснить тем, что KBF_GPU реализует метод полный перебор, тогда как PALMAD избегает избыточных вычислений и использует векторно-матричные структуры данных для достижения наибольшего параллелизма.

График, представленный на рис. 3.3, отражает сравнение производительности PALMAD с алгоритмом Жу и др. в проведенных экспериментах. Видно, что алгоритм Жу и др. значительно опережает PALMAD: до 20 раз на реальных данных и на два порядка на синтетических временных рядах при поиске всех возможных диссонансов. Однако в то же время PALMAD находит существенно больше диссонансов: как минимум на два и на семь порядков больше в реальных и синтетических временных рядах соответственно. Таким образом, сравнивая среднее время работы на поиск одного диссонанса, можно видеть, что PALMAD значительно опережает



(a) Общее время выполнения



(b) Среднее время выполнения на поиск одного диссонанса

Рис. 3.3. Производительность алгоритма PALMAD в сравнении с Жу и др.

аналога, начиная с поиска $topK$ диссонансов, когда значение $topK$ равно четверти от фактического числа найденных диссонансов: не менее чем в два раза и на три порядка на реальных и синтетических временных рядах соответственно.

Масштабируемость

Помимо проведения сравнения алгоритма PALMAD с известными аналогами, проведены эксперименты по исследованию его масштабируемости

от различных значений входных параметров алгоритма. Сначала исследуется влияние длины сегмента временного ряда *seglen* на масштабируемость PALMAD. Затем оценивается масштабируемость PALMAD в зависимости от длины временного ряда n и ширины диапазона длин диссонансов $minL..maxL$, которые напрямую влияют на объем вычислений.

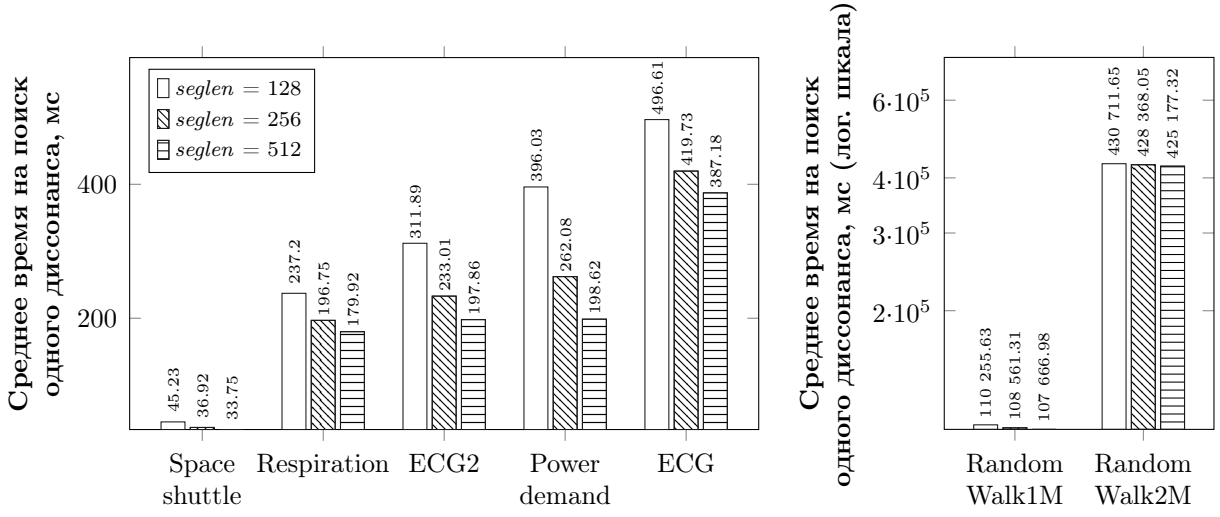
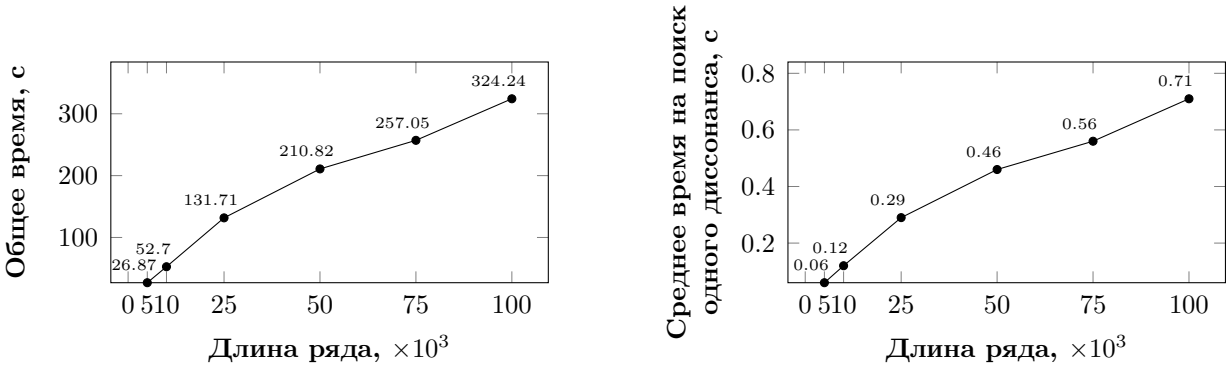


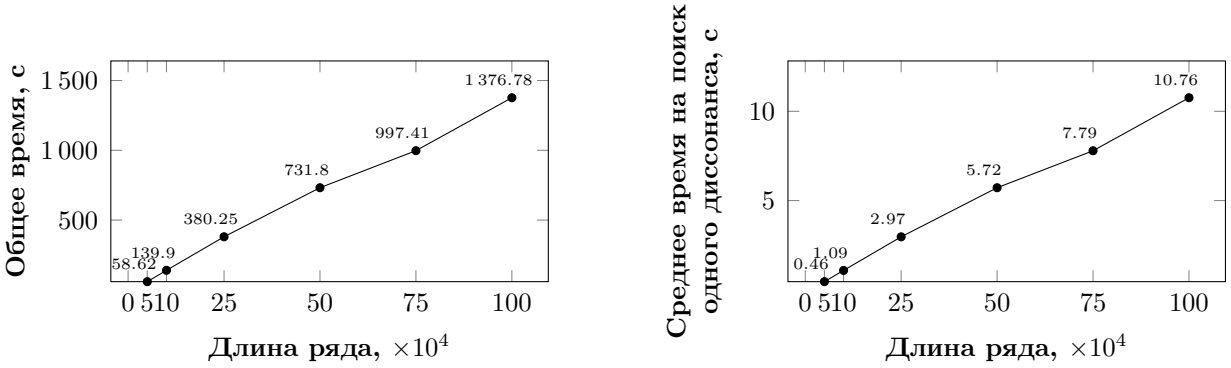
Рис. 3.4. Масштабируемость алгоритма PALMAD в зависимости от длины сегмента

На рис. 3.4 представлены результаты экспериментов, исследующие влияния длины сегмента на производительность PALMAD. Можно заметить, что время работы алгоритма пропорционально длине сегмента как для реального, так и для синтетического временных рядов. При этом при большем значении длины сегмента достигается более высокая производительность алгоритма. Это можно объяснить тем, что при увеличении длины сегмента снижаются накладные расходы на чтение и запись сегментов в разделяемую память графического процессора. Более того, именно по этой причине в описанных выше экспериментах было установлено для параметра длины сегмента временного ряда *seglen* значение 512.

На рис. 3.5 и 3.6 показана производительность алгоритма PALMAD в зависимости от длины временного ряда и ширины диапазона длин диссонансов соответственно для реальных и синтетических рядов. Можно за-



(a) Реальный ряд Koski-ECG (диапазон длин диссонансов 458..916)



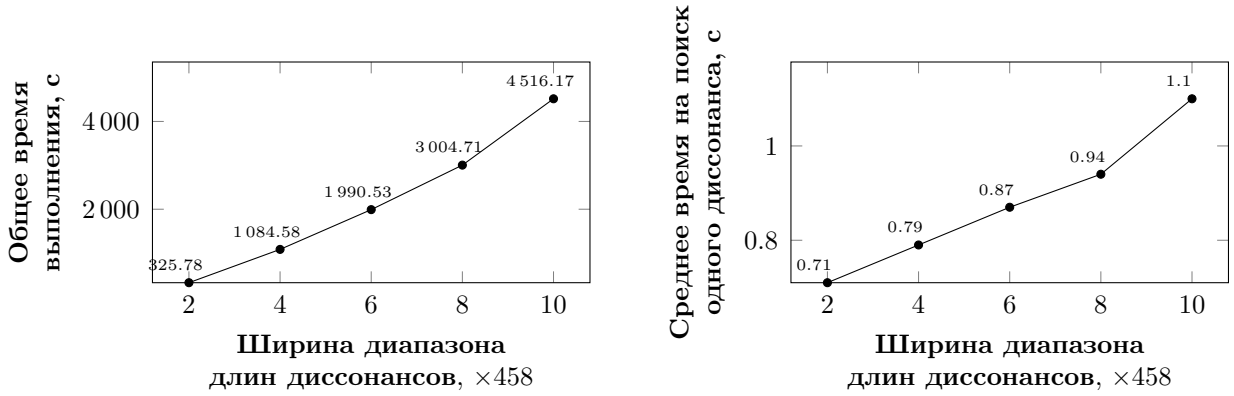
(b) Синтетический ряд RandomWalk1M (диапазон длин диссонансов 128..256)

Рис. 3.5. Масштабируемость алгоритма PALMAD в зависимости от длины временного ряда

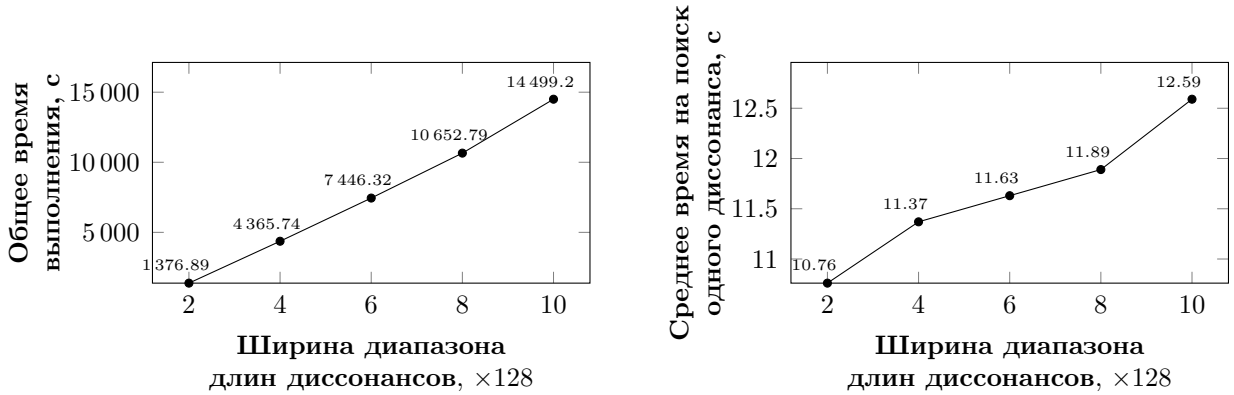
метить, что время работы алгоритма пропорционально вышеупомянутым параметрам как для реальных, так и для синтетических временных рядов.

3.4. Поиск аномалий в сенсорных данных цифровой индустрии

В данном разделе приведены результаты тематических исследований по применению разработанного нами алгоритма для поиска диссонансов в сенсорных данных из различных областей цифровой индустрии. Указанные исследования проведены на оборудовании Лаборатории суперкомпьютерного моделирования ЮУрГУ [2] (платформа GPU-SUSU в табл. 2.2).



(a) Реальный ряд Koski-ECG



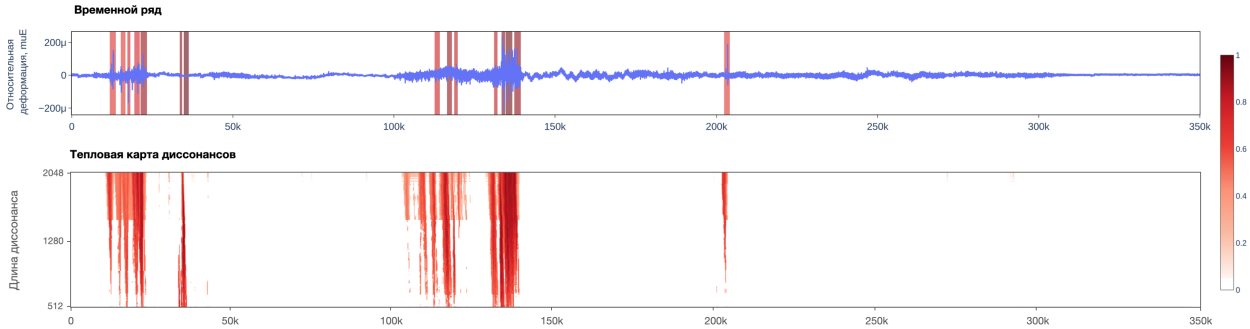
(b) Синтетический ряд RandomWalk1M

Рис. 3.6. Влияние ширины диапазона длин диссонансов на масштабируемость алгоритма PALMAD

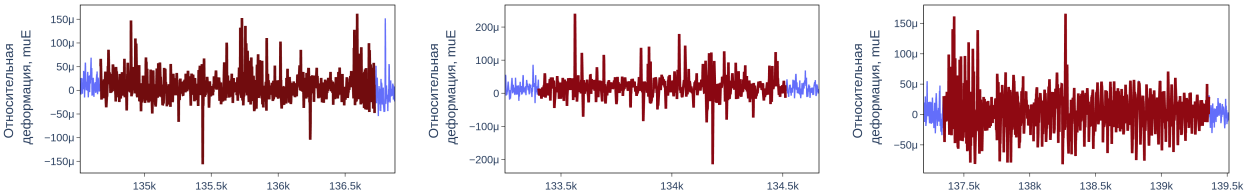
3.4.1. Деформации в механизме сцепки вагонов трамвая

Первое исследование связано с поиском диссонансов в сенсорных данных, полученных с тензометрического датчика, установленного на механизме сцепки вагонов трамвая. Данные указанного датчика представляют собой значения относительной деформации, связанные со значениями напряжений, возникающими в процессе эксплуатации механизма сцепки вагонов. Использованные в исследовании данные сняты в течение одной рабочей смены трамвая, следующего по одному маршруту, с частотой дискретизации 4 096 Гц в течение 1.5 мин. (350 000 точек).

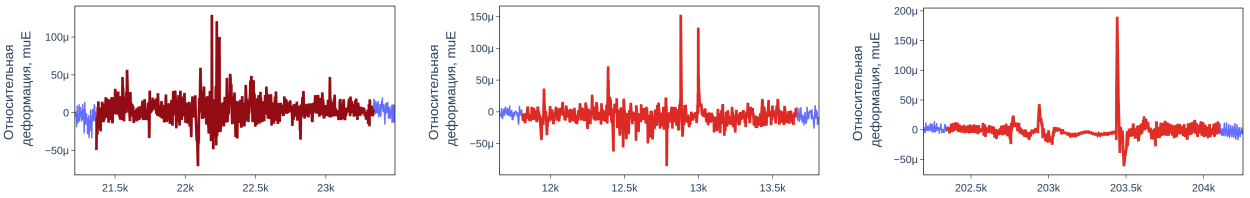
На рис. 3.7а представлены временной ряд и тепловая карта найденных с помощью разработанного нами параллельного алгоритма диссонансов,



(a) Временной ряд и тепловая карта диссонансов



Top-1: $m = 2\,048$ (0.5 с) Top-2: $m = 1\,114$ (0.27 с) Top-3: $m = 2\,015$ (0.49 с)



Top-4: $m = 1\,975$ (0.48 с) Top-5: $m = 1\,852$ (0.45 с) Top-6: $m = 1\,756$ (0.43 с)

(b) Примеры найденных top-k диссонансов различных длин

Рис. 3.7. Аномалии временного ряда относительных деформаций механизма стыковки вагонов трамвая

имеющих длину в диапазоне 512..2 048. На рис. 3.7b представлены примеры шести наиболее значимых диссонансов. Найденные диссонансы позволяют видеть, что при эксплуатации узла сцепки вагонов имеет место работа в экстремальных условиях с высокими амплитудами отклонения знакопеременной нагрузки. Очевидно, что при длительной работе в подобных условиях остаточный ресурс узла сцепки вагонов сокращается.

3.4.2. Разрушение плит системы профилировки валков стана холодной прокатки

Второе исследование связано с поиском диссонансов во временных рядах, полученных с сенсоров, установленных на стане холодной прокатки металлургического завода. Холодная прокатка сопровождается воздействием

больших усилий и напряжений на основные рабочие и вспомогательные элементы стана, поэтому они быстро изнашиваются и часто ломаются, что негативно влияет на качество выпускаемой продукции. Для повышения качества металлопродукции и совершенствования технологии производства холоднокатанных полос каждая клеть стана оснащается системой регулировки профиля полос посредством осевой сдвижки выпукло-вогнутых рабочих валков CVC (Continuously Variable Curvature) [113]. Система CVC позволяет добиться уменьшения разнотолщинности по профилю, по краям и по центру холоднокатанных полос и высокой плоскостности по всей ширине листа. Однако при эксплуатации системы CVC имеют место частые поломки плит, по которым происходит движение системы CVC совместно с рабочими валками клети в горизонтальном направлении. Причиной подобных поломок, предположительно, являются высокие значения знакопеременных нагрузок изгибающих моментов, вызванных изменениями напряженности клети. Указанные нагрузки приводят к появлению в наиболее нагруженных частях плит CVC трещин, которые ведут к разрушению плит.

В исследовании использовались данные сенсоров, установленные на одной из клетей стана холодной прокатки, которые измеряли различные показатели с частотой 1 раз в секунду в течение 6 месяцев, а также сведения о простоях клетей стана вследствие технического обслуживания или ремонта плит CVC за указанный период. В силу ранее сделанного предположения о высокой напряженности клети как о наиболее вероятной причине поломок для исследования нами выбран временной ряд показаний датчика, измеряющего фактическое изгибающее усилие прокатки, в котором исключены периоды простоев стана, не связанных с поломками плит. Полученный ряд представлен на рис. 3.8а. В данном временном ряде выполнялся поиск диссонансов различных длин из диапазона от 30 минут до 1 часа (т.е. $minL = 1\ 800$ и $maxL = 3\ 600$).

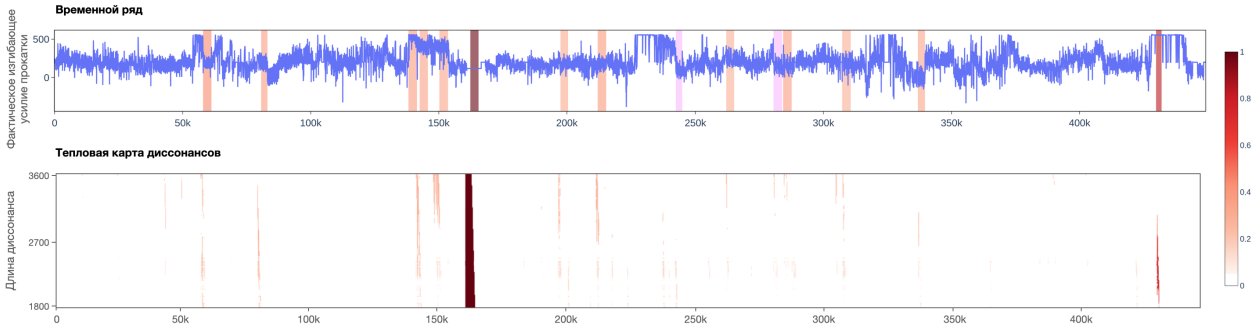
На рис. 3.8а представлены результаты работы алгоритма PALMAD по поиску top-15 диссонансов, которые выделены красным цветом. Далее нами из них были отобраны 4 следующих диссонанса, которые наиболее точ-

но отражают причины возникновения критических ситуаций при работе плит CVC системы (см. рис. 3.8b). Первый диссонанс показывает, что на плиту длительное время действуют напряжения, намного превышающие допустимые значения, вызванные напряженностью клети стана в отсутствии прокатываемого металла. Остальные три диссонанса соответствуют напряженному состоянию в плитах при знакопеременных нагрузках и критических значениях давлений на клеть при обработке проката. Отобранные диссонансы показывают, что в межремонтные периоды плиты, подвергаемые экстремальным знакопеременным нагрузкам, с большой вероятностью могут получить повреждения в виде трещин, приводящих к поломками и выходу плит из строя. Найденные диссонансы могут в дальнейшем быть использованы как паттерны снижения остаточного ресурса плит CVC системы и предсказания их поломок.

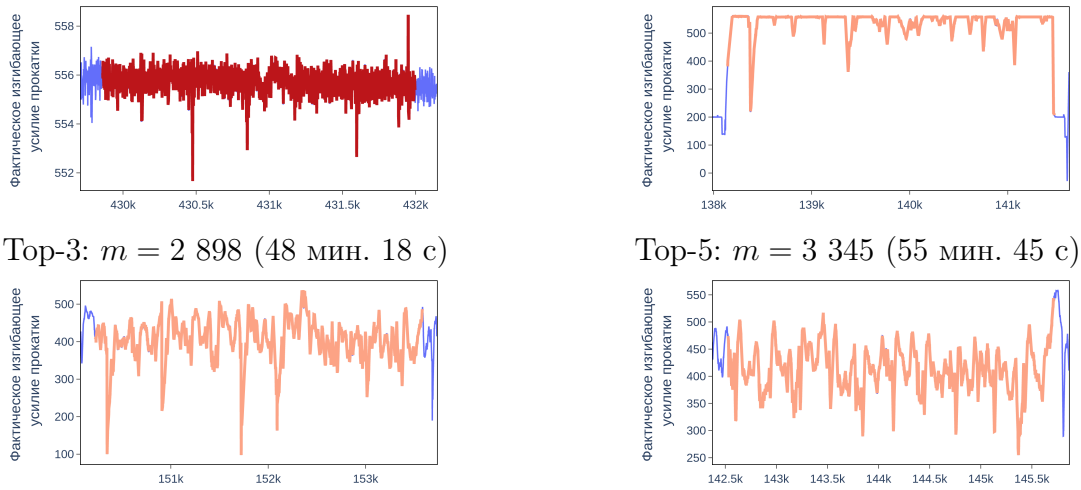
3.4.3. Нештатные ситуации в системе умного управления отоплением зданий

Третье исследование направлено на поиск диссонансов в показаниях температурных датчиков, которые являются частью интеллектуальной системы ПолиТЭР [1] управления теплоснабжением, установленной в Южно-Уральском государственном университете (ЮУрГУ). ПолиТЭР выполняет мониторинг инженерных систем зданий кампуса ЮУрГУ и управляет режимами их работы на основе анализа показаний проводных и беспроводных датчики. Для исследования были взяты показания беспроводного датчика, установленного в одной из учебных аудиторий ЮУрГУ (см. рис. 3.9a), за 2018 г. при частоте снятия показаний 1 раз в 15 мин. В данном временном ряде выполнялся поиск диссонансов различных длин из диапазона от 1 дня до 1 недели (т.е. $minL = 96$ и $maxL = 672$).

Примеры некоторых аномалий из списка top-10, полученного с помощью алгоритма PALMAD, представлены на рис. 3.9b. Первая аномалия показывает, что датчик, скорее всего, вышел из строя, поскольку в течение определенного времени передавалась постоянная температура. В осталь-



(a) Временной ряд фактического изгибающего усилия прокатки



Top-3: $m = 2\,898$ (48 мин. 18 с)

Top-5: $m = 3\,345$ (55 мин. 45 с)

Top-7: $m = 3\,576$ (59 мин. 36 с)

Top-10: $m = 3\,187$ (53 мин. 6 с)

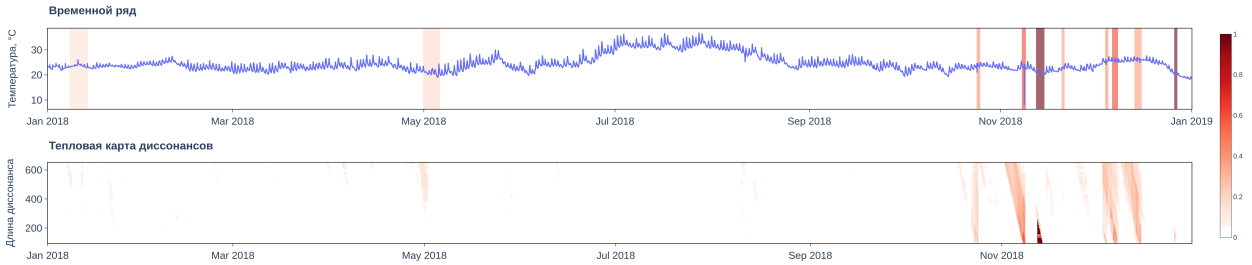
(b) Примеры найденных top-k диссонансов различных длин

Рис. 3.8. Аномалии временного ряда фактического изгибающего усилия прокатки

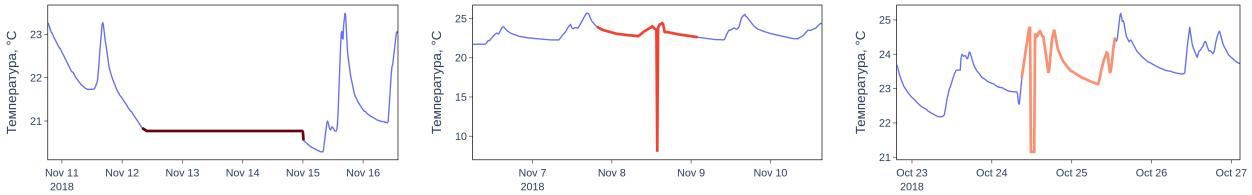
ных случаях аномалии свидетельствуют о влиянии человеческого фактора на энергоэффективность системы теплоснабжения зданий университета: вероятно, резкое изменение температуры вызвано сознательно либо случайно открытыми окнами в помещении.

3.5. Выводы по главе 3

В данной главе представлен новый параллельный алгоритм PALMAD (Parallel Arbitrary Length MERLIN-based Anomaly Discovery), выполняющий поиск диссонансов временного ряда, имеющих длину из заданного диапазона, на графическом процессоре. PALMAD основан на последовательном алгоритме MERLIN. Однако, в отличие от оригинального алгорит-



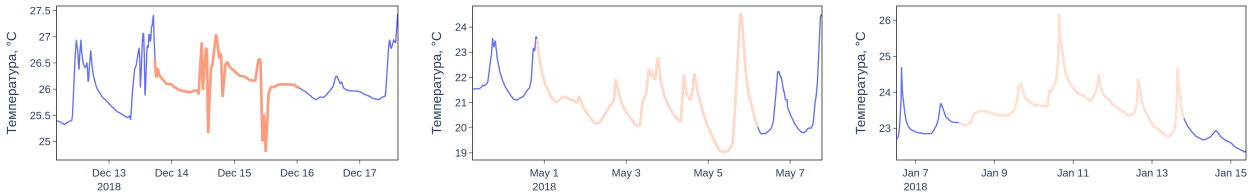
(a) Временной ряд и тепловая карта диссонансов



Top-1: $m = 258$
(2 дня 16 час. 30 мин.)

Top-3: $m = 122$
(1 день 6 час. 30 мин.)

Top-6: $m = 113$
(1 день 4 час. 15 мин.)



Top-7: $m = 219$
(2 дня 6 час. 45 мин.)

Top-9: $m = 504$
(5 дней 6 час.)

Top-10: $m = 650$
(6 дней 18 час. 30 мин.)

(b) Примеры найденных top-k диссонансов различных длин

Рис. 3.9. Аномалии временного ряда показаний температурных датчиков в системе умного управления отоплением зданий

ма, в PALMAD предложены и доказаны рекуррентные формулы для вычисления расстояний между подпоследовательностями ряда, позволяющие существенно сократить объем вычислений. Для нахождения диссонансов фиксированной длины PALMAD применяет параллельный алгоритм PD3 (Parallel DRAG-based Discord Discovery), представленный выше в главе 2.

Проведены вычислительные эксперименты на реальных и синтетических временных рядах, подтверждающие эффективность разработанного алгоритма PALMAD. В экспериментах было выполнено сравнение производительности PALMAD с двумя известными аналогами: KBF_GPU [127] и алгоритм Жу и др. [143], поскольку обзор релевантных работ (см. раздел 1.5.3) не выявил других параллельных алгоритмов, ориентированных

на графические процессоры. Указанные алгоритмы ограничиваются поиском одного (наиболее важного) диссонанса временного ряда.

В экспериментах PALMAD значительно (более чем в 100 раз) опережает алгоритм KBF_GPU по производительности, поскольку KBF_GPU распараллеливает полный перебор подпоследовательностей ряда. В сравнении с алгоритмом Жу и др. PALMAD показывает существенно меньшее время на поиск одного диссонанса: не менее чем в два раза и на три порядка на реальных и синтетических временных рядах соответственно. При этом алгоритм PALMAD ожидаемо проигрывает алгоритму Жу и др. в производительности, поскольку PALMAD, в отличие от конкурента, выполняет поиск всех диссонансов, а не одного, наиболее важного. В соответствии с этим PALMAD обнаруживает значительно большее количество диссонансов: на два и семь порядков больше на реальных и синтетических временных рядах соответственно.

Наконец, в ходе экспериментов также исследовалось влияние длины сегмента на производительность PALMAD. Время выполнения алгоритма пропорционально длине сегмента как для реальных, так и для синтетических временных рядов, так как при увеличении длины сегмента накладные расходы на чтение и запись сегментов в разделяемую память графического процессора уменьшаются.

Представлены результаты исследований по применению алгоритма PALMAD для поиска диссонансов во временных рядах, которые представляют собой показания сенсоров из реальных предметных областей: деформации механизма стыковки вагонов трамвая, деформации плиты системы регулировки профиля полос стана холодной прокатки металлургического завода и показания температурных датчиков интеллектуальной системы управления отоплением зданий. Аномалии, найденные в таких данных, свидетельствуют о нештатной ситуации, отказах, сбоях и износе технологического оборудования.

Для визуализации найденных аномалий предложена техника тепловой карты диссонансов, имеющих различные длины. Представлен алгоритм нахождения наиболее значимых диссонансов независимо от их длин.

Результаты, приведенные в данной главе, опубликованы в работах [4, 149]. Получено свидетельство Роспатента о государственной регистрации программы для ЭВМ [7].

Глава 4. Параллельный алгоритм поиска диссонансов произвольной длины для кластерных вычислительных систем с графическими процессорами

В данной главе представлен новый параллельный алгоритм PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs) поиска диссонансов произвольной длины на высокопроизводительном вычислительном кластере с графическими процессорами. Алгоритм PADDi снимает ограничение длины временного ряда размером памяти графического процессора, имеющееся в алгоритме PALMAD (см. главу 3). Описаны принципы распараллеливания и общая схема вычислений алгоритма PADDi. В отличие от PALMAD, схема вычислений PADDi модифицирована, чтобы обеспечить распараллеливание не только на уровне вычислительных узлов кластера, но и в рамках нескольких графических процессоров, входящих в один узел кластера. Представлены результаты вычислительных экспериментов на реальных и синтетических временных рядах, в которых исследуется производительность и масштабируемость алгоритма PADDi на различных аппаратных платформах.

4.1. Принципы распараллеливания

4.1.1. Сегментация и фрагментация

Алгоритм PADDi обладает двухуровневым параллелизмом по данным (см. рис. 4.1). *Первый уровень параллелизма* представляет распределение нагрузки между вычислительными узлами кластерной системы и реализуется с помощью фрагментации временного ряда. Фрагментация предполагает, что временной ряд T разбивается на фрагменты $T^{(1)}, \ldots, T^{(p)}$ равной длины по числу выделенных алгоритму узлов p вычислительного кластера,

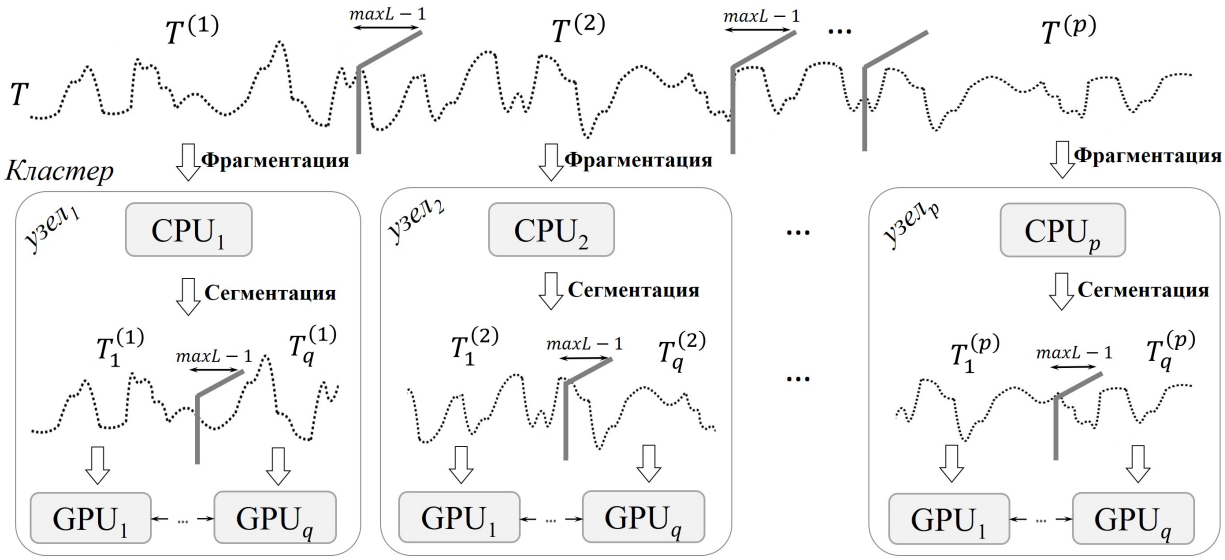


Рис. 4.1. Сегментация и фрагментация в PADDi

каждый из которых оснащен центральным и графическими процессорами. При этом количество фрагментов (узлов кластерной системы) p выбирается таким образом, чтобы фрагмент и соответствующие вспомогательные данные алгоритма могли быть размещены в оперативной памяти вычислительного узла. Если суммарный объем оперативной памяти недостаточен для размещения всех данных, то в кластер добавляется дополнительный узел (узлы).

Фрагментация осуществляется на основе техники разбиения с перекрытием, которая описана в разделе 2.1.2. В конец каждого фрагмента (за исключением последнего) добавляется $maxL-1$ элементов ряда, взятых из начала следующего фрагмента, для предотвращения потери результатов на стыке фрагментов.

В ходе обработки узлы обмениваются данными для выполнения процедуры подбора параметра r. Данные обмены необходимы, поскольку для каждого фиксированного значения r каждому узлу кластера необходимо проверить, что найденные в текущем фрагменте кандидаты в диссонансы являются таковыми для фрагментов всех остальных узлов. Обмены данными между узлами реализуются с помощью технологии MPI [121].

Второй уровень параллелизма заключается в распределении вычислительной нагрузки между графическими процессорами каждого узла и реализуется посредством разбиения фрагмента на сегменты по числу графических процессоров для того, чтобы каждый сегмент обрабатывался отдельным графическим процессором. Сегментация выполняется, как и фрагментация, с помощью описанной выше техники разбиения с перекрытием. В результате применения техники формируются сегменты $T_1^{(i)}, \ldots, T_q^{(i)}$, $1 \leqslant i \leqslant p$, где q — количество графических процессоров, которыми оснащен каждый узел кластера, $q \geqslant 1$.

Далее выполняется выравнивание каждого сегмента для обеспечения баланса загрузки нитей графического процессора, обрабатывающих свой сегмент. Длина сегмента устанавливается кратной размеру варпа графического процессора. Если таковая кратность не обнаруживается, то сегмент дополняется справа фиктивными элементами, имеющими значение $+\infty$. Предполагается, что в итоге описанных преобразований сегмент ряда может быть целиком размещен в оперативной памяти как центрального, так и графического процессоров.

Для обработки всех сегментов фрагмента порождается необходимое количество CUDA потоков (CUDA stream), соответствующее числу графических процессоров узла. В каждом CUDA потоке многократно (для каждой длины диссонанса из указанного диапазона) запускаются CUDA ядра, реализующие алгоритм PD3 и фазу очистки кандидатов от ложноположительных диссонансов фрагмента и ряда в целом. Для организации вычислений на графических процессорах применяется технология CUDA [64].

4.1.2. Общая схема вычислений

Как и в алгоритме PALMAD, распределенный алгоритм PADDi осуществляет поиск диссонансов различных длин из заданного диапазона аналогичным образом. Предполагается разбиение диапазона длин диссонанса на три неравные части и выполнение трех шагов поиска диссонансов соответствующих длин. На каждом шаге выполняются следующие действия:

инициализация порога r, предобработка, поиск диссонансов $\mathcal{D}_m$ текущей длины m с заданным порогом и обновление порога r в случае, если диссонансы не были найдены. По завершении шага найденные диссонансы $\mathcal{D}_m$ добавляются в результирующее множество $\mathcal{D}$. Однако несмотря на схожесть последовательностей действий алгоритмов PALMAD и PADDi, их реализации имеют различия. В отличие от PALMAD, в схему вычислений PADDi внедрены модификации, позволяющие выполнять поиск диссонансов ряда на кластере с multi-GPU узлами.

Организация работы алгоритма, использующего множество графических процессоров на узле, осуществляется следующим образом. Сначала устанавливается количество графических процессоров, между которыми будет равномерно распределяться вся вычислительная нагрузка алгоритма. При этом не допускается, чтобы задаваемое количество графических процессоров превышало количество установленных на узле кластера графических процессоров.

Далее организуется цикл по всем графическим процессорам, где на каждой итерации для текущего графического процессора выделяется закрепленная память (pinned memory) хоста, позволяющая выполнять асинхронную передачу данных между центральным и графическими процессорами, а также глобальная память графического процессора под хранение сегментов временного ряда и вспомогательных структур данных. Чтобы одновременно выполнять вычисления на разных графических процессорах в этом цикле из одного потока центрального процессора также создаются CUDA-потоки (CUDA stream). Число CUDA-потоков совпадает с числом графических процессоров, чтобы один поток был ассоциирован с одним графическим процессором. Функция CUDA-потоков заключается в возможности создания очередей команд запуска CUDA-ядер и копирования данных между центральным и графическим процессорами. При этом в начале каждой итерации цикла устанавливается текущий графический процессор с помощью команды CUDA API cudaSetDevice(), после чего все CUDA-операции выполняются в контексте этого графического процессора.

Далее создается еще один похожий на предыдущий цикл, с тем отличием, что на каждой итерации последовательно переключаясь на соответствующий графический процессор (контекст), в CUDA-потоке производятся асинхронное копирование входных данных алгоритма из памяти центрального процессора в память графического процессора, запуск CUDA-ядер и асинхронные передачи результатов CUDA-ядер из графического на центральный процессор. Поскольку все перечисленные операции асинхронны, управление немедленно возвращается в основной поток центрального процессора. После чего выполняется безопасное переключение на следующий графический процессор и запуск операции на нем в то время, когда операции на предыдущем графическом процессоре еще не завершены. За счет этого механизма достигается параллельность.

По завершении вычислений освобождается память и уничтожаются созданные CUDA-потоки.

Алг. 4.1 отражает общую схему вычислений на каждом шаге подбора параметра r алгоритма PADDi, разработанного для кластера с графическими процессорами. Для поиска множества диссонансов $\mathcal{D}_m$, имеющих длину m ($minL \leqslant m \leqslant maxL$), каждый шаг выполняется по следующей схеме.

Сперва каждый графический процессор узла выполняет параллельную предобработку средних значений и стандартных отклонений подпоследовательностей своего сегмента ряда, и найденные статистические показатели записываются в векторы $\overline{\mu}$ и $\overline{\sigma}$ (см. строки 8–11 в алг. 4.1). После этого осуществляется поиск диссонансов, состоящий их двух фаз: отбор и очистка диссонансов фрагмента и очистка диссонансов ряда. Каждая фаза выполняется на каждом узле кластера.

Первая фаза алгоритма PADDi начинается с отбора и очистки диссонансов сегмента, которые задействуют представленный в главе 2 параллельный алгоритм PD3 (см. строку 13 в алг. 4.1). PD3 применяется к каждому сегменту ряда $T_j^{(i)}$, обрабатываемому на отдельном j-м графическом процессоре i-го узла кластера, $1 \leqslant j \leqslant q$, $1 \leqslant i \leqslant p$. Результатом вы-

Алг. 4.1. PADDI (IN T, $minL$, $maxL$, $topK$; OUT $\mathcal{D}$)

$Node_i$ $(i \in [1, p])$

CPU_i $(i \in [1, p])$	GPU_j $(j \in [1, q])$
$myrank \leftarrow$ MPI_Comm_rank()	
if $myrank = 0$ **then** $\mathcal{D} \leftarrow \varnothing$	
Read fragment $T^{(i)}$ of T	
$\{T_i^{(i)}\}_{j=1}^{q} \leftarrow \text{SEGMENTATE}(T^{(i)})$	
for $m \leftarrow minL$ **to** $maxL$ **do**	
$nnDist_m \leftarrow -\infty$	
$r \leftarrow \text{INITTHRESHOLD}()$	
	▷ Предобработка ряда
	if $m = minL$ **then**
	$\bar{\mu}_j^{(i)}, \bar{\sigma}_j^{(i)} \leftarrow \text{CALCMEANSTD}(T_j^{(i)})$
	else
	$\bar{\mu}_j^{(i)}, \bar{\sigma}_j^{(i)} \leftarrow \text{UPDATEMEANSTD}(T_j^{(i)}, \bar{\mu}_j^{(i)}, \bar{\sigma}_j^{(i)})$
while $nnDist_m < 0$ **and** $\lvert\mathcal{D}\rvert < topK$ **do**	
	▷ Локальный отбор и очистка
	$\mathcal{D}_j^{(i)} \leftarrow \text{PD3}(T_j^{(i)}, \mu_j^{(i)}, \sigma_j^{(i)}, r^2)$
$\mathcal{C}^{(i)} \leftarrow \cup_{j=1}^{q} \mathcal{D}_j^{(i)}$	
	$\mathcal{D}_j^{(i)} \leftarrow \text{LOCALREFINE}(T_j^{(i)}, \mathcal{C}^{(i)})$
$\mathcal{C}_m^{(i)} \leftarrow \cap_{j=1}^{q} \mathcal{D}_j^{(i)}$	
$\mathcal{C}_m \leftarrow$ MPI_Allgatherv($\mathcal{C}_m^{(i)}$, MPI_FLOAT)	
	▷ Глобальная очистка
	$\mathcal{D}_j^{(i)} \leftarrow \text{GLOBALREFINE}(T_j^{(i)}, \mathcal{C}_m)$
$\widetilde{\mathcal{C}}_m^{(i)} \leftarrow \cap_{j=1}^{q} \mathcal{D}_j^{(i)}$	
if $myrank = 0$ **then**	
$\mathcal{D}_m \leftarrow \cap_{j=1}^{p}$MPI_Recv($\widetilde{\mathcal{C}}_m^{(i)}$)	
$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_m$	
$nnDist_m \leftarrow \min_{d \in \mathcal{D}_m} d.nnDist$	
else	
MPI_Send($\widetilde{\mathcal{C}}_m^{(i)}$)	
if $myrank = 0$ **then**	
MPI_Send($nnDist_m$)	
else MPI_Recv($nnDist_m$)	
$r \leftarrow \text{UPDATETHRESHOLD}(r)$	
return $\mathcal{D}$	

полнения на каждом GPU алгоритма PD3 является множество диссонансов сегмента $\mathcal{D}_j^{(i)}$. Далее формируется множество подпоследовательностей-кандидатов в диссонансы $\mathcal{C}^{(i)}$ как объединение множеств диссонансов всех сегментов $\mathcal{C}^{(i)} = \cup_{j=1}^{q} \mathcal{D}_j^{(i)}$ (см. строку 14 в алг. 4.1).

Однако полученное множество $\mathcal{C}^{(i)}$ может содержать ложноположительные диссонансы, поскольку диссонансы, найденные в одном сегменте, не сравниваются с подпоследовательностями других сегментов, принадлежащих одному и тому же фрагменту, и, тем самым, могут не быть диссонансами в рамках всего фрагмента. Для отбрасывания ложноположительных диссонансов из $\mathcal{C}^{(i)}$ каждый графический процессор выполняет очистку диссонансов фрагмента, формируя множество $\mathcal{D}_j^{(i)}$ (см. строку 15 в алг. 4.1). Подробное описание данной процедуры представлено в следующем разделе 4.2 данной главы. Далее узел получает все $\mathcal{D}_j^{(i)}$. Заканчивается первая фаза тем, что на стороне центрального процессора создается множество кандидатов фрагмента $\mathcal{C}_m^{(i)}$, являющееся результатом от применения операции пересечения над множествами $\mathcal{D}_j^{(i)}$, т.е. $\mathcal{C}_m^{(i)} = \cap_{j=1}^{q} \mathcal{D}_j^{(i)}$ (см. строку 16 в алг. 4.1).

Необходимо отметить, что в данной фазе все обмены данными выполняются через память центрального процессора узла кластера. Кроме того, если задействуется только один графический процессор на каждом узле кластера, то при выполнении первой фазы очистка диссонансов фрагмента пропускается.

Далее узлы кластера осуществляют обмен найденными кандидатами фрагментов по принципу «каждый с каждым», после чего на каждом узле формируется множество кандидатов ряда в диссонансы $\mathcal{C}_m = \cup_{i=1}^{p} \mathcal{C}_m^{(i)}$ (см. строку 17 в алг. 4.1).

Вторая фаза заключается в очистке диссонансов ряда, реализация которой схожа с очисткой диссонансов фрагмента из предыдущей фазы (см. строку 18 в алг. 4.1, ниже раздел 4.2). Данная процедура предполагает удаление ложноположительных экземпляров, которые не являются диссонансами в ряде в целом, из множества $\mathcal{C}_m$, объединившего в себе диссонан-

сы всех фрагментов ряда. После этого центральный процессор формирует множество кандидатов в диссонансы ряда $\tilde{\mathcal{C}}_m^{(i)}$ из не отброшенных графическими процессорами диссонансов $\mathcal{D}_j^{(i)}$ как $\tilde{\mathcal{C}}_m^{(i)} = \cap_{j=1}^{q} \mathcal{D}_j^{(i)}$ (см. строку 19 в алг. 4.1).

После этого один из узлов вычислительного кластера (например, нулевой узел) объявляется мастером, остальные — рабочими. Каждый узел-рабочий отправляет узлу-мастеру свое множество локальных диссонансов, после чего мастер формирует результирующее множество диссонансов заданной длины как $\mathcal{D}_m = \cap_{i=1}^{p} \tilde{\mathcal{C}}_m^{(i)}$ (см. строки 20–21, 24–25 в алг. 4.1). Пороговое расстояние вычисляется как $r = \max\limits_{c \in \mathcal{D}_m} c.nnDist$, где запись $c.nnDist$ означает расстояние от кандидата в диссонансы c до его ближайшего соседа.

По завершении шага подбора параметра r узел-мастер пополняет итоговое множество диссонансов исходного ряда $\mathcal{D}$, добавляя в него только что найденное множество $\mathcal{D}_m$. Таким образом, итоговое множество диссонансов вычисляется как $\mathcal{D} = \cup_{m=minL}^{maxL} \mathcal{D}_m$ (см. строку 22 в алг. 4.1).

В описанной выше схеме используются функции `MPI_Allgatherv` и `MPI_Gatherv` стандарта MPI для реализации приема-передачи локальных кандидатов и локальных диссонансов соответственно.

4.2. Распараллеливание фазы очистки диссонансов

Принцип очистки диссонансов фрагмента и очистки диссонансов ряда схож между собой и осуществляется следующим образом (см. алг. 4.2). Очистка диссонансов заключается в проведении каждым графическим процессором узлов кластера очистки множества кандидатов для множества подпоследовательностей соответствующего сегмента $S^m_{T_j^{(i)}}$ в соответствии с процедурой, предусмотренной в алгоритме DRAG (см. раздел 1.5.2). Для простоты дальнейшего изложения выполним обобщение множества кандидатов в диссонансы фрагмента $\mathcal{C}_m$ и множества кандидатов в диссонансы ряда $\tilde{\mathcal{C}}_m$, введя для них одно обозначение $\mathcal{C}$.

Алг. 4.2. GLOBAL-/LOCAL- REFINE (IN S, C, $\overline{\mu}$, $\overline{\sigma}$, m, r; OUT $\mathcal{D}$)

1: $Bitmap \leftarrow \overline{\texttt{TRUE}}$; $nnDist \leftarrow \overline{+\infty}$
2: $start\_x_{Tile_\text{S}} \leftarrow blockid.y \cdot blocklen$; $end\_x_{Tile_\text{S}} \leftarrow (blockid.y + 1) \cdot blocklen$;
3: $start\_y_{Tile_\text{C}} \leftarrow blockid.x \cdot blocklen$; $end\_y_{Tile_\text{C}} \leftarrow (blockid.x + 1) \cdot blocklen$;
▷ Обработка тайлов
4: **for** $i \leftarrow 0$ **to** m **step** $blocklen$ **do**
▷ Копирование тайлов из глобальной в разделяемую память
5: $Tile_\text{S} \leftarrow S(start\_x_{Tile_\text{S}} : end\_x_{Tile_\text{S}}, i : i + blocklen)$
6: $Tile_\text{C} \leftarrow C(i : i + blocklen, start\_y_{Tile_\text{C}} : end\_y_{Tile_\text{C}})$
▷ Вычисление скалярных произведений
7: $Tile_\text{P} \leftarrow Tile_\text{P} + \text{MATRIXMULTIPLY}(Tile_\text{S}, Tile_\text{C})$
8: $\{\overline{\mu}_\text{C}, \overline{\sigma}_\text{C}\} \leftarrow \{\overline{\mu}_\text{C}, \overline{\sigma}_\text{C}\} + \text{CALCMEANSTD}(Tile_\text{C}, m)$
▷ Вычисление расстояний
9: $dist \leftarrow \text{CALCDIST}(Tile_\text{P}, \overline{\mu}, \overline{\sigma}, \overline{\mu}_\text{C}, \overline{\sigma}_\text{C}, m)$
10: **if** $dist < r$ **then**
11: $nnDist(start\_y_{Tile_\text{C}} + tid.x) \leftarrow \min\big(dist, nnDist(start\_y_{Tile_\text{C}} + tid.x)\big)$
12: **else**
13: $Bitmap(start\_y_{Tile_\text{C}} + tid.x) \leftarrow \texttt{FALSE}$
▷ Формирование множества диссонансов
14: $\mathcal{D} \leftarrow \big\{\{C_i \in C;\ nnDist(i)\} \mid 1 \leqslant i \leqslant |C|, Bitmap(i) = \texttt{TRUE}\big\}$
15: **return** $\mathcal{D}$

Для очистки диссонансов необходимо вычислить расстояния от каждого кандидата $c \in \mathcal{C}$ до каждой подпоследовательности $s \in S^m_{T_j^{(i)}}$, используя формулу (1.8). В оперативной памяти графического процессора каждого узла кластера множества подпоследовательностей-кандидатов $\mathcal{C}$ и подпоследовательностей сегмента $S^m_{T_j^{(i)}}$ представляются в виде матриц $C \in \mathbb{R}^{|\mathcal{C}| \times m}$ и $S \in \mathbb{R}^{N \times m}$ соответственно. Для хранения средних арифметических и стандартных отклонений подпоследовательностей-кандидатов предусматриваются векторы $\overline{\mu}$, $\overline{\sigma} \in \mathbb{R}^{|\mathcal{C}|}$ соответственно. Для хранения расстояний между кандидатами и их ближайшими соседями используется массив $nnDist \in \mathbb{R}^{|\mathcal{C}|}$.

Результатом умножения матриц S и C является матрица $P \in \mathbb{R}^{N \times |\mathcal{C}|}$, каждый элемент которой представляет скалярное произведение соответствующих строк матрицы S (подпоследовательностей) и столбцов транспо-

нированной матрицы $C^{\intercal}$ (кандидатов). Для эффективного умножение матриц S и C используется блочный алгоритм, предложенный в работе [103].

Для выполнения указанного умножения на графическом процессоре формируется двумерная сетка нитей, состоящая из блоков нитей размера $blocklen \times blocklen$, где параметр $blocklen$ (количество нитей в блоке) выбирается кратным размеру варпа и не превышает максимальное количество нитей в блоке. Каждый блок нитей вычисляет одну подматрицу (тайл) $Tile_{\mathrm{P}} \in \mathbb{R}^{blocklen \times blocklen}$, а нить блока отвечает за вычисление одного элемента тайла.

Для нахождения элементов тайла матрицы P блок нитей обращается к двум полосам матриц подпоследовательностей S и кандидатов C. Каждая из полос разбивается на квадратные подматрицы (тайлы) $Tile_{\mathrm{S}} \in \mathbb{R}^{blocklen \times blocklen}$ и $Tile_{\mathrm{C}} \in \mathbb{R}^{blocklen \times blocklen}$. Обработка этих тайлов осуществляется следующим образом. Сначала организуется цикл по количеству тайлов в полосах, где на каждой итерации блоки нитей загружают в разделяемую память графического процессора два тайла, один из которых берется из полосы матрицы S, другой — из C (см. строки 5–6 в алг. 4.2). После каждой загрузки нити блока выполняют считывание соответствующей строки в $Tile_{\mathrm{S}}$ и столбца в $Tile_{\mathrm{C}}$ и вычисляют скалярное произведение между ними. Найденное частичное значение суммируется с текущим результатом скалярного произведения, полученным на предыдущих итерациях этого цикла. Таким образом, после обработки всех тайлов, взятых из соответствующих полос, каждая нить находит скалярное произведение между подпоследовательностью сегмента и кандидатом в диссонансы (см. строку 7 в алг. 4.2).

Далее каждая нить блока, имеющая координаты $(i,\, j)$, используя формулу (1.8), вычисляет расстояние между кандидатом $C(i,\, \cdot)$ и подпоследовательностью $S(j,\, \cdot)$ (см. строку 9 в алг. 4.2) на основе уже найденного скалярного произведения как

$$dist(C(i,\, \cdot),\, S(j,\, \cdot)) = 2m \left(1 - \frac{P(i,\, j) - m \cdot \overline{\mu}(i) \cdot \overline{\mu}(j)}{m \cdot \overline{\sigma}(i) \cdot \overline{\sigma}(j)}\right). \qquad (4.1)$$

Затем, используя операцию атомарного минимума (выполняемого над данными в приватной памяти текущей нити), блок нитей вычисляет расстояние от кандидата $C(i, \cdot)$ до его ближайшего соседа как минимум всех вышеуказанных расстояний, помещая результат в $nnDist(i)$ (см. строку 11 в алг. 4.2).

На финальном шаге очистки, используя подсчитанные выше расстояния до ближайших соседей кандидатов, множество диссонансов формируется как $\mathcal{D}_j^{(i)} = \{c \in \mathcal{C} \mid c.nnDist \geqslant r\}$ (см. строку 14 в алг. 4.2).

4.3. Вычислительные эксперименты

4.3.1. Описание экспериментов

Для исследования эффективности разработанного алгоритма PADDi были проведены вычислительные эксперименты. В экспериментах изучалась и анализировалась масштабируемость алгоритма, понимаемую как способность пропорционально увеличивать ускорение и производительность при добавлении аппаратных ресурсов (графических процессоров узлов кластера), определяемые следующим образом.

Производительность представляет собой среднее арифметическое по времени работы алгоритма за 10 запусков, исключая затраты на ввод исходных данных и вывод полученных результатов. *Ускорение* алгоритма $s(p)$ на узлах кластера, которые суммарно оснащены p графическими процессорами, вычисляется как $s(p) = \frac{t_1}{t_p}$, где t_1 и t_p — время работы алгоритма PADDi на одном и на p графических процессорах узлов кластера соответственно.

В экспериментах использовались следующие три временных ряда, резюмированные в табл. 3.1. Каждый временной ряд имеет длину 2 000 000 точек. Для всех рядов поиск диссонансов осуществлялся в диапазоне длин 64..128. Ряд ECG [47] содержит показания электрокардиограммы взрослого пациента. Ряд GAP [53] представляет собой поминутные показатели общего энергопотребления частного дома во Франции в период 2006–2010 гг. Ряд

Табл. 4.1. Наборы данных экспериментов с алгоритмом PADDi

Временной ряд	Длина ряда, n	Диапазон длин диссонансов, $minL..maxL$	Предметная область
ECG	$2 \cdot 10^6$	64..128	ЭКГ взрослого человека
GAP			Энергопотребление частного дома во Франции
MGAB			Синтетический ряд на основе уравнения Макки—Гласса

MGAB [125] синтезирован на основе уравнения Макки—Гласса (нелинейное дифференциальное уравнение с временной задержкой) [93].

Табл. 4.2. Аппаратная платформа экспериментов с алгоритмом PADDi

Характеристика	GPU-MSU		GPU-UNN
Производитель, семейство	NVIDIA Tesla		NVIDIA Kepler
Модель	K40	P100	K20X
Количество CUDA-ядер	2 880	3 584	2 688
Частота ядра, ГГц	0.745	1.19	0.732
Оперативная память, Гб	11.56	16	6
Пиковая производительность (двойная точность), TFLOPS	1.682	4	1.31

Вычислительные эксперименты были проведены на платформе суперкомпьютеров Ломоносов-2 [129] и Лобачевский (НОЦ «СКТ-Приволжье» Университет Лобачевского, Нижний Новгород) для следующих трех конфигураций вычислительного кластера с узлами на базе графических процессоров. Характеристики графических процессоров приведены в табл. 4.2.

Конфигурация 48×K40 задействовала 48 графических процессоров узлов суперкомпьютера Ломоносов-2 из раздела Compute. При этом каждый узел оснащен одним графическим процессором NVIDIA Tesla K40.

Конфигурация 64×K20 использовала 64 графических процессора узлов сегмента суперкомпьютера Лобачевский. На всех узлах установлено по три графических процессора NVIDIA Kepler K20X.

Конфигурация 64×P100 задействует 64 графических процессора узлов суперкомпьютера Ломоносов-2 из раздела Pascal. Каждый узел обладает двумя графическими процессорами NVIDIA Tesla P100.

4.3.2. Анализ результатов

Графики, представленные на рис. 4.2–4.4, отражают масштабируемость алгоритма в проведенных экспериментах. Результаты экспериментов показывают картину, общую для всех рядов и всех конфигураций всех кластеров. Можно видеть, что имеет место уменьшение ускорения и производительности по сравнению с линейным при переносе алгоритма на конфигурацию с бóльшим количеством графических процессоров. Причина заключается в увеличении накладных расходов на обмены кандидатами в диссонансы между узлами. Тем не менее, масштабируемость алгоритма сохраняет линейный характер и ее стагнация и деградация отсутствуют.

Важно отметить, что для решения задачи поиска диссонансов на параллельных системах с распределенной памятью обмены кандидатами неизбежны, поскольку механическое применение парадигмы MapReduce [38] приведет к некорректному результату: кандидаты в диссонансы, найденные в одном фрагменте ряда, но очищенные в рамках того же фрагмента без привлечения кандидатов других фрагментов ряда, очевидно, не обязаны являться диссонансами ряда в целом в смысле определения (1.4) [86].

Выполнены эксперименты по сравнению масштабируемости алгоритма PADDi при запуске на разном количестве узлов, обладающих одинаковым суммарным количеством графических процессоров. В этой серии экспериментов участвовали конфигурации *64×K20* и *64×P100*. Из рис. 4.2–4.4 можно сделать вывод, что когда алгоритм выполняется на узлах кластера, где задействовано максимально возможное количество графических процессоров, масштабируемость больше по сравнению с тем, когда алгоритм запускается на большем числе узлов, но с одним графическим процессором. Это можно объяснить тем, что один графический процессор отбрасывает меньше кандидатов в диссонансы в своем фрагменте, что, в свою очередь,

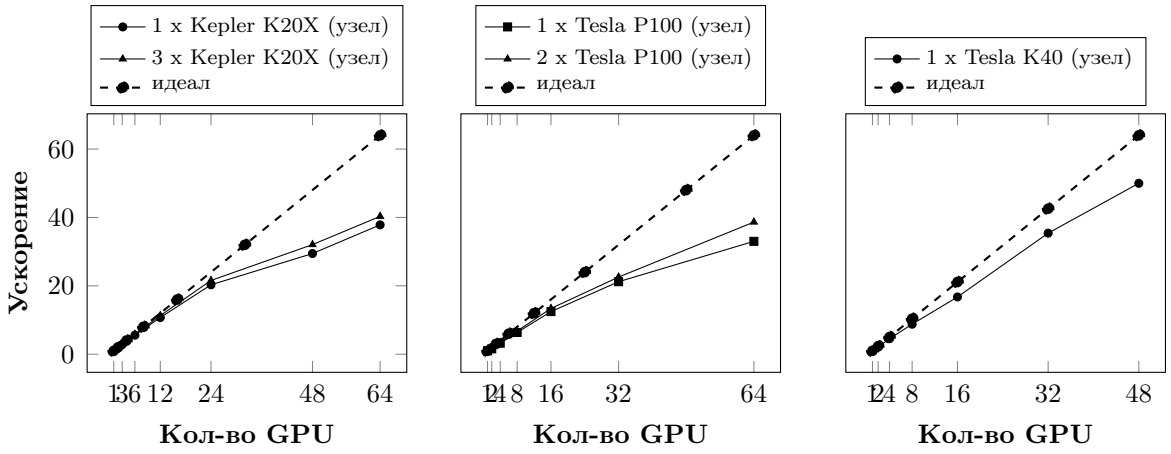
приводит к увеличению объема пересылок и времени, затрачиваемому на фазу очистки диссонансов ряда, из-за большего объема вычислений.

Сравнив результаты экспериментов, полученных для трех временных рядов, было замечено, что для синтетического ряда MGAB получено наибольшее ускорение (см. рис. 4.4a), поскольку в нем оказалось существенно меньше кандидатов, что снизило объем их пересылок.

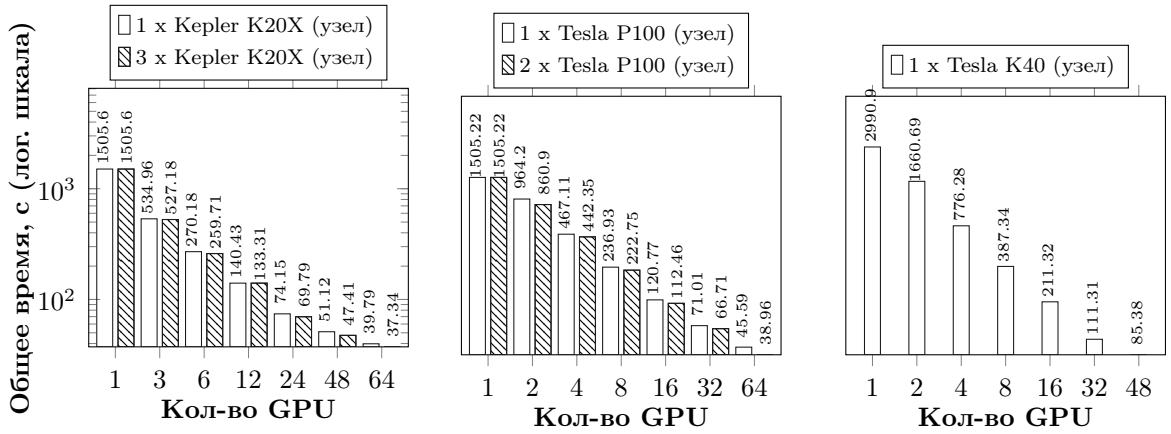
Статистические итоги поиска диссонансов представлены на рис. 4.5 и рис. 4.6 и показывают следующее. В обоих рядах распределение найденных диссонансов по длинам примерно одинаковое (см. рис. 4.5a), а общее количество найденных диссонансов не превышает 0.08% от общего числа подпоследовательностей ряда с искомыми длинами, что согласуется с интуитивным представлением об аномалиях (аномалия — то, что встречается редко).

Размеры кандидатов, пересылаемых между узлами кластера, для всех исследуемых временных рядов представлены на рис. 4.2c–4.4c) и определяют распределение времени работы алгоритма по его фазам. Первая фаза подразумевает вычисления на графических процессорах узлов, необходимые для отбора кандидатов в диссонансы сегмента и очистки диссонансов сегмента и фрагмента. Вторая фаза включает в себя пересылки кандидатов между узлами, очистки диссонансов ряда и вычисления на узле-мастере, необходимые для получения результирующего множества диссонансов. Большее время выполнение второй фазы соответствует меньшему ускорению алгоритма (см. рис. 4.2a–4.4a и рис. 4.6).

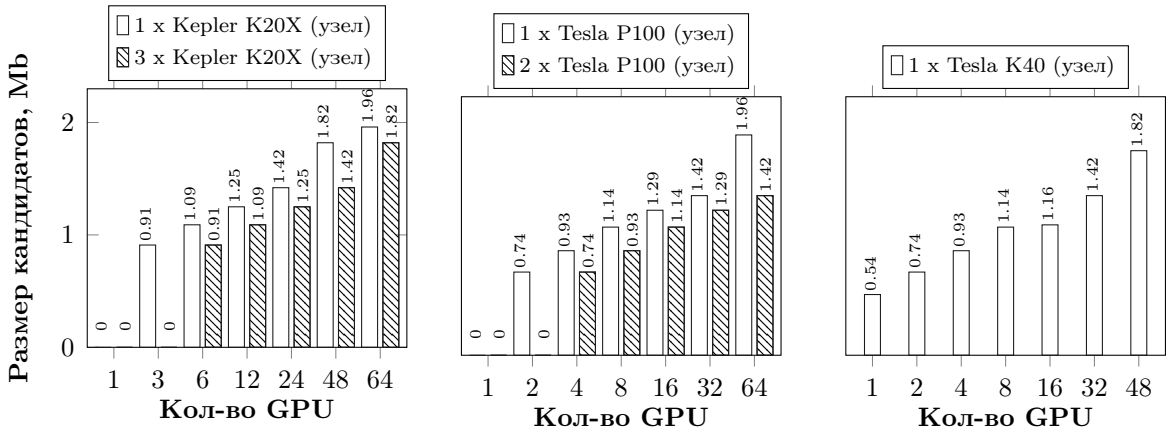
Помимо описанных выше исследований, также сравнивалось количество найденных диссонансов у разработанного алгоритма и двух алгоритмов аналогичного назначения в описанных выше временных рядах. Отметим, что провести оценку качества соответствия найденных диссонансов аномалиям соответствующих реальных предметных областей не представляется возможным. Такая оценка требует ручной разметки аномалий во временных рядах, предварительно выполненной экспертом в предметной области, но в нашем случае разметка задействованных в экспериментах



(a) Ускорение



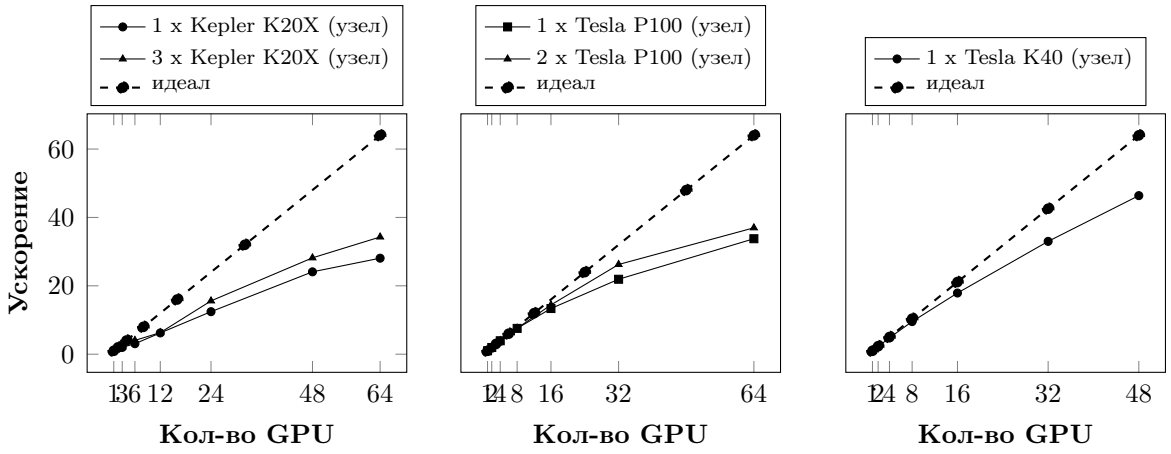
(b) Производительность



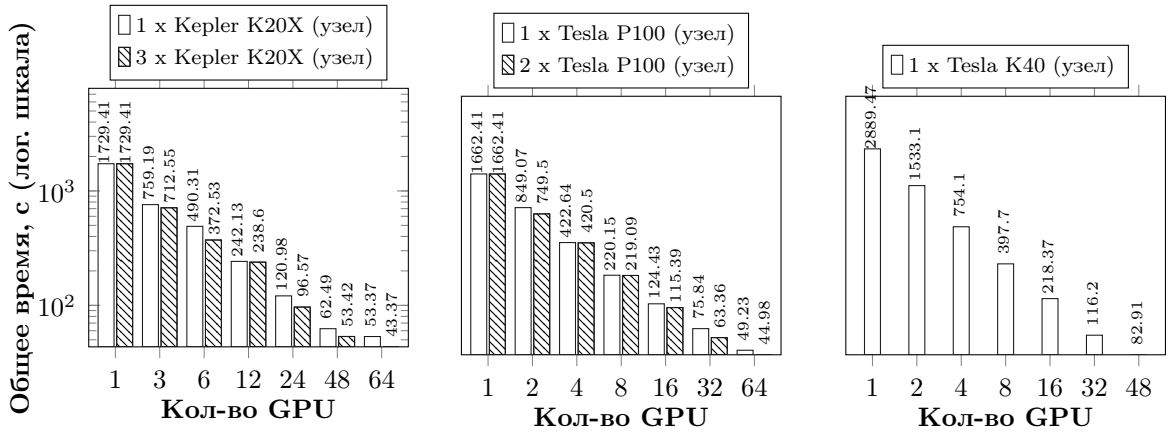
(c) Размер найденных кандидатов

Рис. 4.2. Масштабируемость алгоритма PADDi (временной ряд ECG)

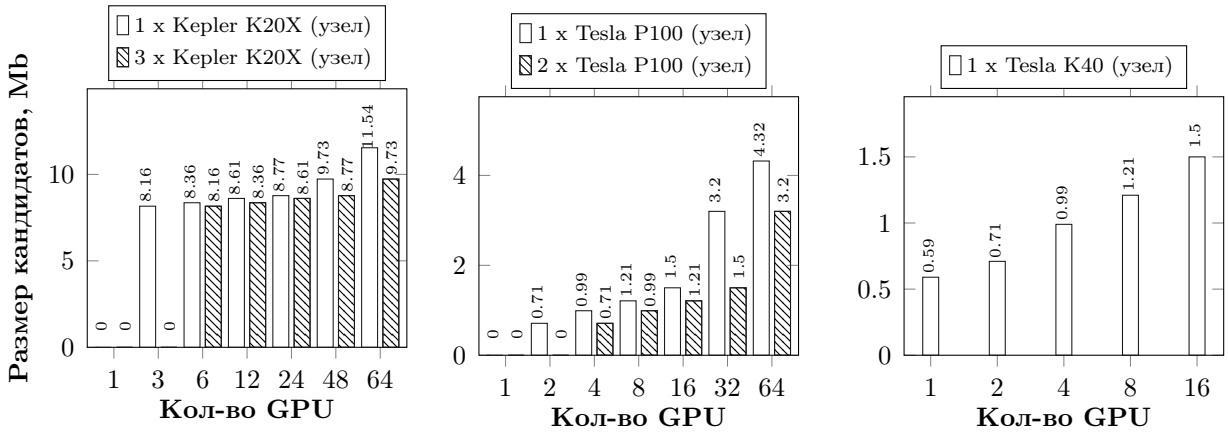
временных рядов отсутствует. Первый из указанных выше алгоритмов — это последовательный алгоритм MERLIN [101], распределенной версией ко-



(a) Ускорение



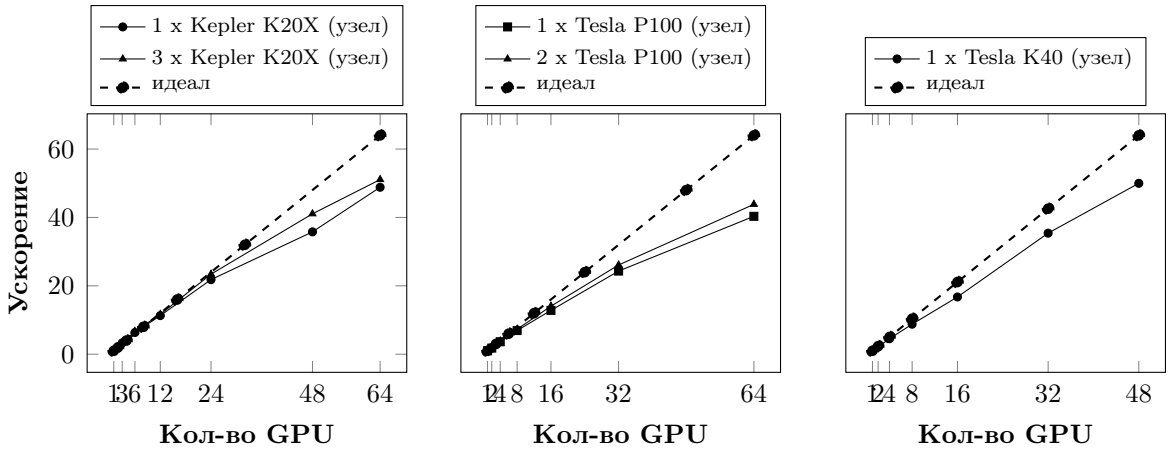
(b) Производительность



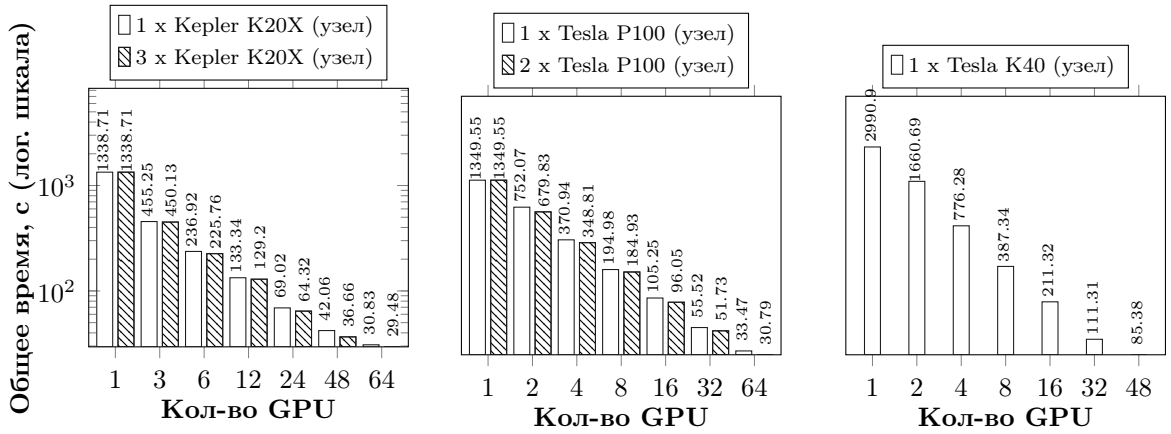
(c) Размер найденных кандидатов

Рис. 4.3. Масштабируемость алгоритма PADDi (временной ряд GAP)

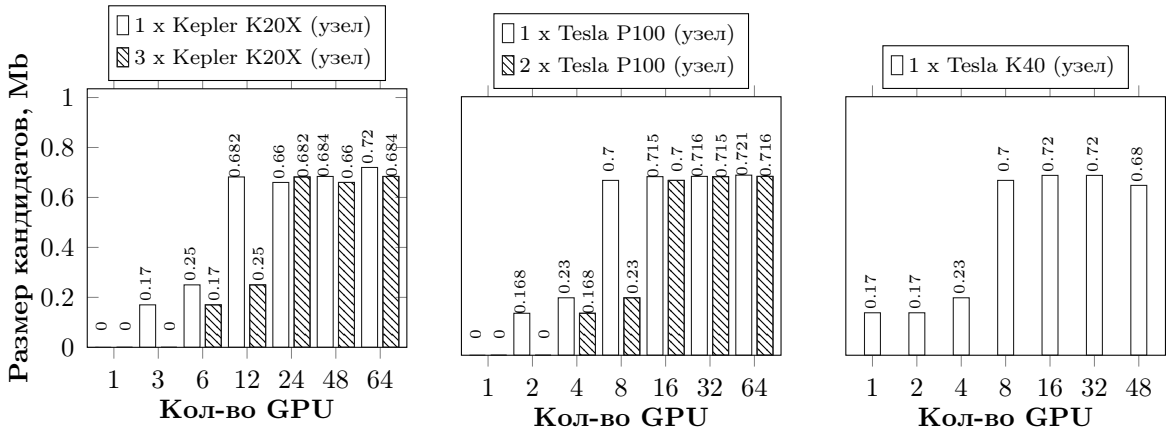
торого является алгоритм PADDi. Второй алгоритм, SCAMP [147], предназначен для вычисления матричного профиля [142] временного ряда на



(a) Ускорение



(b) Производительность



(c) Размер найденных кандидатов

Рис. 4.4. Масштабируемость алгоритма PADDi (временной ряд MGAB)

графическом процессоре. Диссонансы, имеющие длину m, могут быть най-

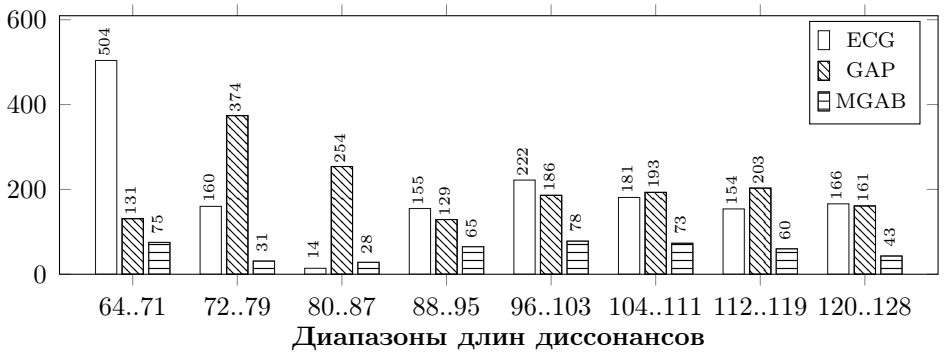


Рис. 4.5. Количество диссонансов, найденных алгоритмом PADDi

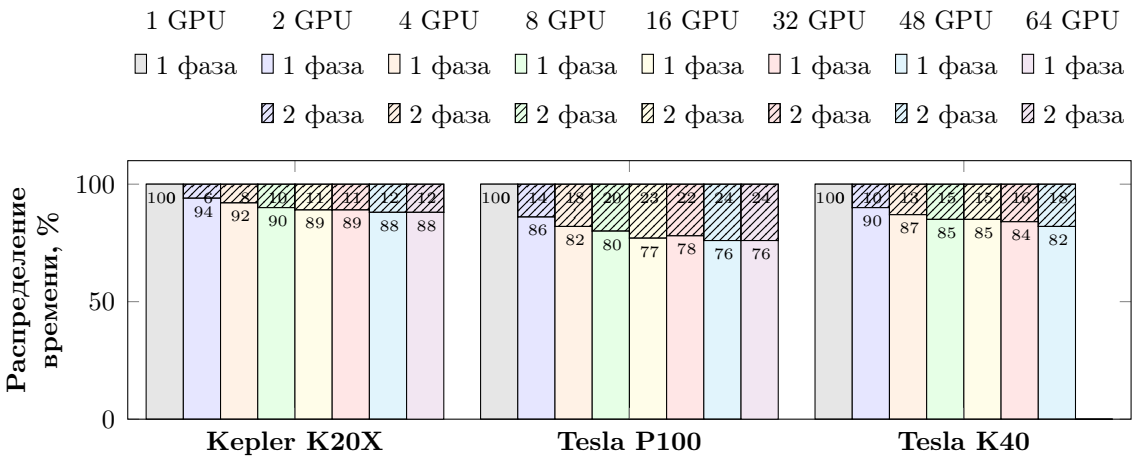
дены как подпоследовательности ряда, которым соответствуют локальные максимумы MP_m.

Результаты сравнения показали, что алгоритм PADDi и вышеупомянутые алгоритмы находят одинаковое количество диссонансов. Можно заключить, что предложенный алгоритм адекватно реализует поиск диссонансов.

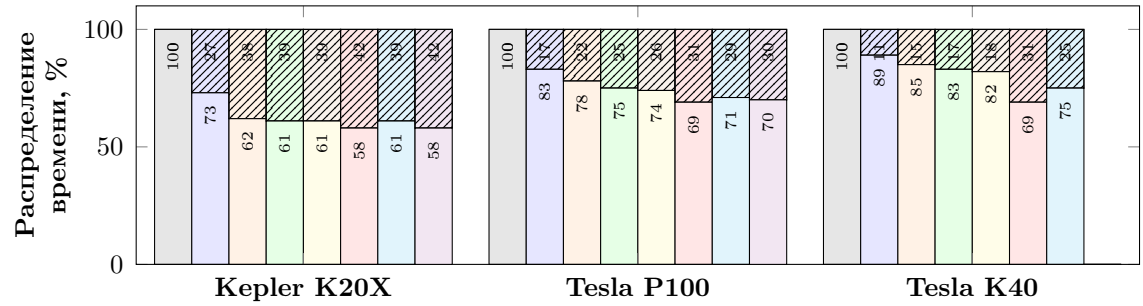
4.4. Выводы по главе 4

В данной главе представлен новый параллельный алгоритм поиска диссонансов временного ряда, имеющих длину в заданном диапазоне, на вычислительном кластере, каждый узел которого оснащен графическим процессором. Новый алгоритм получил название PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs). Благодаря этой разработке решается проблема, имеющаяся у алгоритма PALMAD, который не способен обрабатывать временные ряды, не помещающиеся в оперативную память графического процессора.

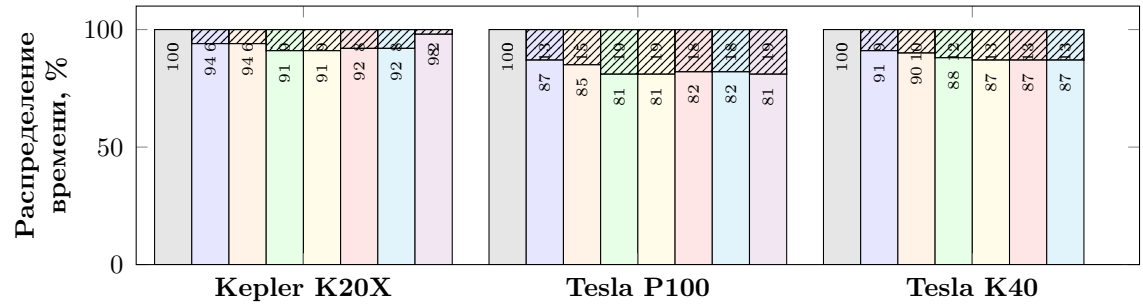
Предложенный алгоритм PADDi модифицирует схему вычислений PALMAD следующим образом. Алгоритм предполагает двухуровневое распараллеливание вычислений: на уровне всех узлов кластерной системы используется технология MPI, в рамках одного графического процессора вычислительного узла кластера используется технология CUDA. Временной ряд разбивается на фрагменты, распределяемые по узлам вычислительно-



(a) Ряд ECG



(b) Ряд GAP



(c) Ряд MGAB

Рис. 4.6. Распределение времени по фазам алгоритма PADDi

го кластера. В свою очередь, каждый фрагмент разбивается на сегменты по числу графических процессоров в узле. При этом фрагменты и сегменты имеют нахлест для предотвращения потерь результатов на стыках.

Алгоритм предусматривает цикл по диапазону длин диссонансов, где на каждой итерации осуществляется параллельная предобработка и поиск диссонансов текущей длины. Поиск диссонансов предполагает две фазы, выполняемые на каждом узле кластера. Сначала в каждом фрагменте выполняется отбор кандидатов в диссонансы и отбрасывание ложноположительных диссонансов. Отбор и очистка диссонансов фрагмента задействует алгоритм PD3, который применяется к каждому сегменту ряда, обрабатываемому на отдельном графическом процессоре. Затем полученные множества кандидатов всех фрагментов объединяются и рассылаются каждому узлу с помощью технологии MPI по принципу «каждый с каждым». После этого полученные кандидаты проходят очистку в рамках всех фрагментов на основе параллельного блочного перемножения матриц кандидатов и подпоследовательностей сегмента. Результирующие множества передаются узлу-мастеру, который выдает итоговый результат как пересечение этих множеств.

Проведены вычислительные эксперименты, на платформе суперкомпьютеров Ломоносов-2 и Лобачевский, оснащенных 48–64 графическими процессорами, над синтетическими и реальными временными рядами. В экспериментах исследовались ускорение и производительность алгоритма при добавлении в кластер графических процессоров. Результаты экспериментов показали картину, общую для всех рядов и всех конфигураций всех кластеров. При добавлении графических процессоров ускорение уменьшается, но сохраняет линейный характер и его стагнация или деградация отсутствуют. При этом ускорение больше, если на одном узле кластера задействовано большее количество графических процессоров.

Результаты, приведенные в данной главе, опубликованы в работе [6].

Глава 5. Метод поиска аномалий в потоковых данных

В данной главе предлагается новый метод поиска аномальных подпоследовательностей в потоковом временном ряде, получивший название DiSSiD (Discord, Snippet, and Siamese Neural Network-based Detector of anomalies). Описаны компоненты метода DiSSiD: нейросетевая модель, определяющая степень аномальности входной подпоследовательности ряда, и алгоритм формирования обучающей выборки для этой модели, основанный на концепции диссонансов и сниппетов (типичных подпоследовательностей временного ряда). Для обучения нейросетевой модели предложена модифицированная функция контрастных потерь. Также описана модификация алгоритма поиска сниппетов, позволяющая более точно по сравнению с оригинальным алгоритмом находить указанные подпоследовательности заданного временного ряда. Предлагается эвристика, на основе которой выполняется подбор гиперпараметров для модифицированного алгоритма поиска сниппетов. Представлены результаты вычислительных экспериментов над временными рядами из различных предметных областей.

5.1. Нейросетевая модель поиска аномалий

5.1.1. Архитектура модели

Разработанная нейросетевая модель представлена на рис. 5.1. DiSSiD представляет собой сиамскую нейронную сеть (Siamese Neural Network, SNN) [35]. SNN объединяет в себе две подсети, которые имеют одинаковые архитектуру, конфигурацию (количество слоев, число нейронов в каждом слое, размерность входного и выходного слоев, функции активации и др.), а также наборы весов и смещений, полученных в результате обучения. Каждая из указанных подсетей формирует векторное представление (embedding) поданной на вход подпоследовательности, а на выходе

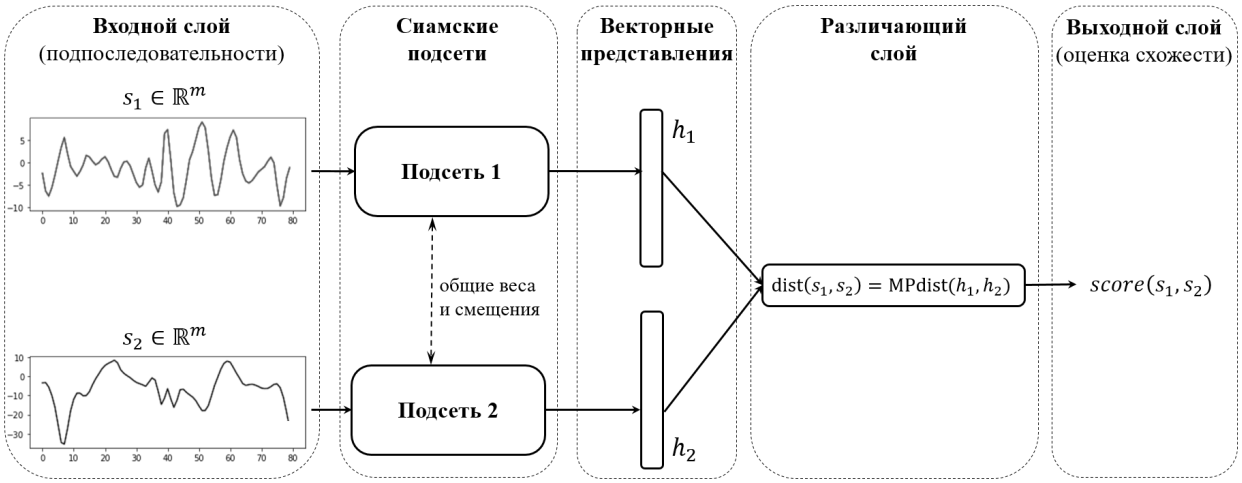


Рис. 5.1. Нейросетевая модель DiSSiD

модель выдает MPdist-расстояние между сформированными векторными представлениями. В качестве подсети фигурирует модификация нейросетевой модели ResNet [52]. Архитектура SNN по сравнению с традиционными нейросетевыми моделями лучше приспособлена к обучению в случае дисбаланса классов и позволяет добавить новый класс в уже развернутую модель без ее повторного обучения [35]. Архитектура ResNet в настоящее время является одним из наиболее эффективных средств решения проблемы затухания градиента (vanishing gradient) при обучении нейросетевой модели [52].

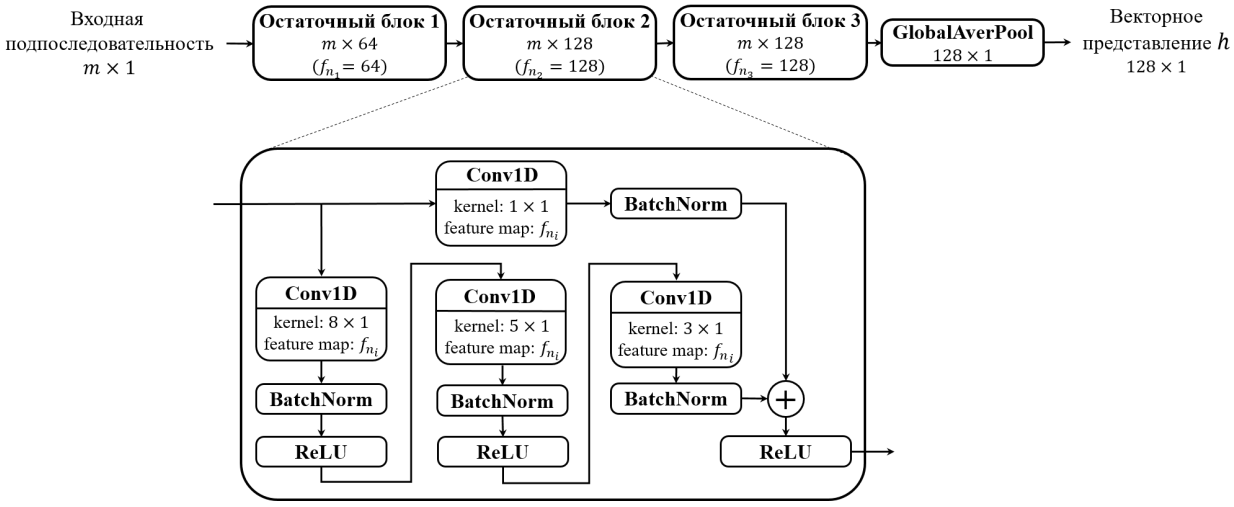


Рис. 5.2. Архитектура подсети ResNet

Подсеть на основе ResNet имеет следующую архитектуру (см. рис. 5.2). На входной слой поступает подпоследовательность временного ряда, имеющая длину m. Далее подсеть включает в себя три одинаковых остаточных блока (residual block) и за ними слой глобальной усредняющей агрегации (GlobalAveragePooling), формирующий векторное представление. Размерность итогового векторного представления определяется количеством карт признаков последнего слоя в последнем остаточном блоке.

Каждый остаточный блок включает в себя три сверточных слоя, на которых применяются фильтры (ядра) с размерами 8×1, 5×1 и 3×1 соответственно. Каждый сверточный слой чередуется со слоем пакетной нормализации (batch normalization) [61], к которому применяется функция активации Линейный выпрямитель (ReLU, Rectified linear unit). Пакетная нормализация преобразует набор входных данных таким образом, что его математическое ожидание обращается в ноль, а дисперсия — в единицу, и предназначена для ускорения сходимости обучения.

После прохождения трех сверточных слоев остаточный блок выдает карты признаков (feature maps): первый блок — 64 карты, остальные два блока — по 128 карт. Далее выполняется сложение входа остаточного блока, пропущенного через сверточный слой с ядром размера 1×1, с выходом этого остаточного блока. Однако входы не могут добавляться напрямую к выходам остаточного блока, поскольку они не имеют одинаковых размеров. Данный прием применяется как средство преодоления проблемы затухающего градиента (vanishing gradient) [54] между внутренними слоями нейронной сети.

5.1.2. Обучение модели

Для обучения модели DiSSiD из заданного временного ряда формируется обучающая выборка, определяемая следующим образом:

$$\begin{aligned}
\mathcal{L} =& \mathcal{L}_{\text{true}} \cup \mathcal{L}_{\text{false}} \setminus \mathcal{L}_{\text{anomaly}} \\
\mathcal{L}_{\text{true}} =& \{< s_1; s_2; 1 > \mid \; s_1, s_2 \in C_i.NN, \quad 1 \leqslant i \leqslant K\} \\
\mathcal{L}_{\text{false}} =& \{< s_1; s_2; 0 > \mid \; s_1 \in C_i.NN,\; s_2 \in C_j.NN, \quad i \neq j,\; 1 \leqslant i, j \leqslant K\},
\end{aligned} \tag{5.1}$$

где множество $\mathcal{L}_{\text{true}}$ включает в себя пары подпоследовательностей, входящих в множество ближайших соседей одного и того же сниппета, множество $\mathcal{L}_{\text{false}}$ — пары подпоследовательностей из множеств ближайших соседей разных сниппетов, а множество $\mathcal{L}_{\text{anomaly}}$ содержит аномальные подпоследовательности, не включаемые в обучающую выборку. Алгоритм очистки исходного временного ряда от аномальных подпоследовательностей и формирования обучающей выборки рассмотрен далее в разделе 5.2.

Для обучения модели DiSSiD предлагается следующая модифицированная функция контрастных потерь (contrastive loss) [49]:

$$\begin{aligned}
L(s_1, s_2) &= \delta_{s_1 s_2} \cdot \text{MPdist}(h_1, h_2) + \\
&\quad + (1 - \delta_{s_1 s_2}) \big(\max(\tau - \text{MPdist}(h_1, h_2), 0)\big)^2 \qquad (5.2) \\
\delta_{s_1 s_2} &= \begin{cases} 1, & s_1, s_2 \in C_i.NN, 1 \leqslant i \leqslant K \\ 0, & s_1 \in C_i.NN, s_2 \in C_j.NN, 1 \leqslant i \neq j \leqslant K \end{cases} \qquad (5.3) \\
\tau &= \min_{\{s_1, s_2 \mid \delta_{s_1 s_2} = 0\}} \text{MPdist}_\ell(h_1, h_2), \qquad (5.4)
\end{aligned}$$

где s_1 и s_2, h_1 и h_2 — исходные подпоследовательности и их векторные представления соответственно; кронекериан $\delta_{s_1 s_2}$ принимает значение 1, если исходные подпоследовательности являются ближайшими соседями одного и того же сниппета, и 0 в противном случае; τ — минимальное расстояние MPdist между векторными представлениями исходных подпоследователь-

ностей, являющихся ближайшими соседями разных сниппетов (параметр модели). Указанная функция потерь обеспечивает обучение модели, в результате которого похожие подпоследовательности исходного ряда получат векторные представления, отстоящие друг от друга в смысле расстояния MPdist не более чем на τ, а непохожие — более чем на τ соответственно.

Перед обучением элементы множества $\mathcal{L}$ случайным образом разделяются на два не пересекающихся подмножества: обучающую и валидационную выборки $\mathcal{L}_{\text{train}}$ и $\mathcal{L}_{\text{valid}}$, используемые для обучения модели и настройки ее гиперпараметров соответственно. Мощности указанных выборок находятся в традиционном соотношении 80% и 20% соответственно.

5.1.3. Применение модели

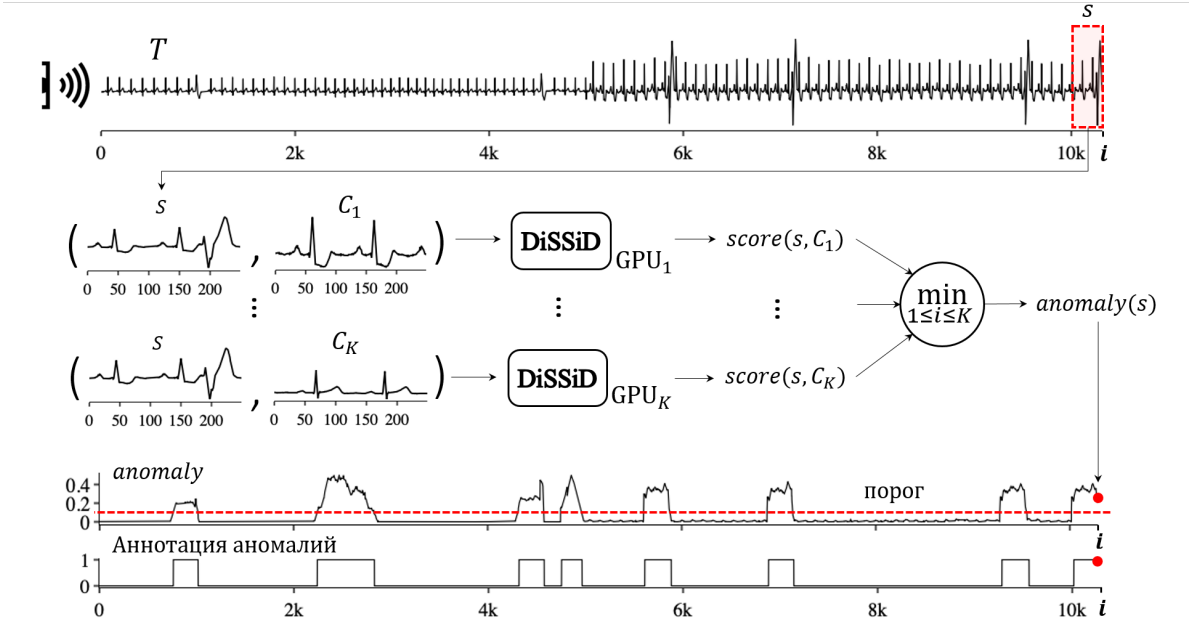


Рис. 5.3. Применение модели DiSSiD

Пусть имеется обученная модель DiSSiD, обучающая выборка которой содержит набор сниппетов $\{C_i\}_{i=1}^{K}$ и с помощью модели требуется определить, является ли входная подпоследовательность s аномальной. Предполагается, что K экземпляров модели запускаются каждый на отдельном

графическом процессоре. Применение обученной модели выглядит следующим образом (см. рис. 5.3).

Сначала формируется набор пар $\{< s; C_i >\}_{i=1}^{K}$ «входная подпоследовательность, сниппет». Затем элементы данного набора параллельно подаются на вход экземплярам модели, каждая из которых выдает $score(s, C_i)$, соответствующую оценку схожести векторных представлений элементов входной пары в смысле расстояния MPdist. Далее оценка аномальности входной подпоследовательности получается как выполнение редукции по операции минимума $anomaly(s) = \min\limits_{1 \leqslant i \leqslant K} score(s, C_i)$. Подпоследовательность s считается аномальной, если $anomaly(s)$ превышает значение наперед заданного аналитиком порога.

В качестве порога используется значение k-го процентиля по набору оценок схожести, которые выдает модель DiSSiD на кортежах валидационной выборки $\mathcal{L}_{\text{valid}}$, имеющих вид $< s_1; s_2; 1 >$ (иными словами, порог — это k-й процентиль подпоследовательностей валидационной выборки, входящих в множество ближайших соседей одного и того же сниппета). В данном исследовании в качестве порога применяется $k = 95$.

5.2. Алгоритм формирования обучающей выборки

Для формирования обучающей выборки $\mathcal{L}$ для представленной выше модели DiSSiD аналитик выбирает репрезентативный временной ряд, адекватно отражающий типичную деятельность субъекта (противоположную аномалиям, которые предполагается обнаруживать с помощью модели). Автоматизированное формирование обучающей выборки включает в себя два шага: очистка и генерация. Очистка подразумевает формирование множества подпоследовательностей ряда, имеющих заданную аналитиком длину, и удаление из указанного множества аномальных подпоследовательностей, которые не должны попасть в обучающую выборку. На шаге генерации из очищенного множества подпоследовательностей тривиальным образом формируются описанные выше множества $\mathcal{L}_{\text{true}}$ и $\mathcal{L}_{\text{false}}$.

Алг. 5.1. CLEANDATA (IN: T, m, α, φ, K; OUT: $\mathcal{L}$)

1: $C_T^m \leftarrow \text{SFA}(T, m, K)$ ▷ Поиск типичной активности
2: $C_{\text{weak}} \leftarrow \{C_i \in C_T^m \mid C_i.frac \leqslant \varphi, 1 \leqslant i \leqslant K\}$ ▷ Поиск нетипичной активности
3: $C_T^m \leftarrow C_T^m \setminus C_{\text{weak}}$
4: **for** $i \leftarrow 1$ **to** $|C_T^m|$ **do**
5: $\quad O \leftarrow \text{ISOLATIONFOREST}(C_i.profile) \cup O$ ▷ Поиск шумов
6: $n_{\text{discord}} \leftarrow \lceil \alpha \cdot (n - m + 1) \rceil$
7: $D \leftarrow \text{PALMAD}(T, m, m, n_{\text{discord}})$ ▷ Поиск аномальной активности
8: $\mathcal{L}_{\text{anomaly}} \leftarrow C_{\text{weak}} \cup C_{\text{weak}}.NN \cup D \cup O$
9: $\mathcal{L} \leftarrow S_T^m \setminus \mathcal{L}_{\text{anomaly}}$ ▷ Очистка
10: **return** $\mathcal{L}$

Псевдокод шага очистки представлен в алг. 5.1. Данный алгоритм имеет следующие задаваемые аналитиком параметры: репрезентативный ряд T ($|T| = n$), длина подпоследовательности m ($m \ll n$), предполагаемая доля аномальных подпоследовательностей в ряде α ($0 < \alpha \ll 1$), предполагаемое количество сниппетов K ($K > 1$), пороговая мощность сниппета φ ($0 < \varphi < 1/K$).

Для очистки обучающей выборки из исходного ряда удаляются подпоследовательности, которые соответствуют аномальной и нетипичной активности субъекта, а также подпоследовательности-шумы. Подпоследовательности, отражающие аномальную активность, трактуются как диссонансы. Подпоследовательности, которые отражают нетипичную активность субъекта, трактуются как сниппеты, имеющие мощность меньше заданного порога φ, и множества их ближайших соседей. Подпоследовательности-шумы трактуются как выбросы в рамках каждого сниппета.

Поиск сниппетов выполняется с помощью алгоритма SFA (Snippet-Finder Advanced) (см. строку 1 в алг. 5.1), который представляет собой улучшенную версию алгоритма Snippet-Finder [60] и описаны ниже в разделе 5.3. Затем из найденного множества сниппетов исключаются маломощные сниппеты (см. строки 2, 3 в алг. 5.1). Далее в множестве ближайших соседей каждого сниппета выполняется нахождение подпоследовательностей-

шумов, реализуемое как поиск выбросов в профиле данного сниппета с помощью метода изолирующего леса (Isolation Forest) [88] (см. строки 4, 5 в алг. 5.1). Наконец, с помощью разработанного автором параллельного алгоритма PALMAD (см. главу 3) выполняется поиск диссонансов, реализующий нахождение подпоследовательностей аномальной активности (см. строки 6, 7 в алг. 5.1). Мощность множества диссонансов вычисляется на основе параметра α, долей аномальных подпоследовательностей в исходном временном ряде (типичным значением данного параметра является $\alpha = 0.05$). В завершении очистки из множества подпоследовательностей ряда исключаются найденные ранее маломощные сниппеты и их ближайшие соседи, а также выбросы и диссонансы (см. строки 8, 9 в алг. 5.1), формируя тем самым множество, используемое для генерации обучающей выборки модели.

5.3. Улучшение алгоритма поиска сниппетов

Поиск сниппетов в алгоритме формирования обучающей выборки выполняется с помощью модифицированного алгоритма SnippetFinder [60], который получил название $\text{SnippetFinder}_{\text{Advanced}}$ (сокращенно SFA). SFA используется для разметки временного ряда и участвует в очистке обучающей выборки. В данной алгоритм было внедрено две модификации.

Первая модификация заключается в поиске MPdist-профилей, которые наиболее адекватно определяют сниппеты ряда и соответствующие им типичные активности. Вторая модификация подстраивает алгоритм SnippetFinder под задачу поиска аномалий. С помощью добавления данной модификации автоматически подбирается гиперпараметр, использующийся при вычислении расстояния MPdist между подпоследовательностями сегментов и подпоследовательностями временного ряда. В целом, все модификации направлены на формирование качественно размеченной обучающей выборки. Далее более подробно разбирается каждая из этих модификаций.

5.3.1. Отбор MPdist-профилей сниппетов

Модификация заключается в усовершенствовании процедуры выбора набора MPdist-профилей сегментов в качестве сниппетов, которые наилучшим образом определяют типичные активности.

Под MPdist-профилем временного ряда T длины n и сегмента ряда $Seg_i \in Seg_T^m$ меньшей длины m ($m \ll n$) понимается вектор из $n - m + 1$ элементов, каждый из которых представляет собой величину схожести соответствующей подпоследовательности ряда T, имеющей длину m, с сегментом Seg_i, также имеющим длину m, в смысле меры MPdist, и обозначается как MPD:

$$MPD(Seg_i, T, \ell) = \{d_j\}_{j=1}^{n-m+1}, \quad d_j = \text{MPdist}_\ell(Seg_i, T_{j,m}). \tag{5.5}$$

В оригинальном алгоритме предлагается целевая функция O, с помощью которой производится оценка всевозможных наборов D_{subset}, составленных из k MPdist-профилей, выбранных из множества всех профилей D, где k — количество сниппетов. Для нахождения целевой функции для каждого набора MPdist-профилей сначала выполняется построение кривой репрезентативности M, где каждый элемент равен минимуму среди соответствующих элементов этих MPdist-профилей. Более формально, кривая репрезентативности M представляет собой ряд, в котором i-я точка показывает схожесть по мере MPdist между i-й подпоследовательностью ряда, имеющей длину m, и наиболее похожим на нее сегментом ряда, взятым из заданного подмножества сегментов D_{subset}:

$$M(D_{\text{subset}}) = \{M_i\}_{i=1}^{n-m+1}, \quad M_i = \min_{D_j \in D_{\text{subset}}} \{d_i \mid d_i \in D_j\}, \quad D_{\text{subset}} \subset D. \tag{5.6}$$

Далее под кривой репрезентативности M вычисляется площадь $ProfileArea$, которая определяет значение целевой функции O. Для выбора результирующего набора MPdist-профилей сегментов выполняется минимизация целевой функции O. Данная функция O достигает своего минимума

при наименьшей площади $ProfileArea$ под кривой репрезентативности M:

$$O(D_{\text{subset}}) = ProfileArea(D_{\text{subset}}) = \sum_{i=1}^{n-m} M_i(D_{\text{subset}}) \to \min\,. \qquad (5.7)$$

Тем самым среди всех наборов выбирается тот, который имеет наименьшую площадь $ProfileArea$ под кривой репрезентативности M.

Для поиска сниппетов в оригинальном алгоритме SnippetFinder вместо полного перебора всех наборов MPdist-профилей и вычисления для них оценок используется жадный подход, который сокращает время достижения глобального экстремума. Идея жадного подхода состоит в том, что глобальное оптимальное решение можно получить, делая локальный оптимальный выбор. В данной задаче для составления результирующего набора из k MPdist-профилей на каждой итерации добавляется в текущий результирующий набор тот MPdist-профиль, который совместно с профилями, выбранными на предыдущих итерациях, образуют кривую репрезентативности с минимальной локальной площадью.

Однако способ отбора MPdist-профилей сегментов в оригинальном алгоритме имеет имеет следующий недостаток. Алгоритм может ошибочно в качестве сниппета выбрать сегмент ряда, который не отражает типичную активность ряда, а является аномалией или шумом в рамках своей активности. Это, в свою очередь, приведет к неадекватным результатам как поиска ближайших соседей такого сниппета в рамках соответствующей активности, так и поиска типичных активностей во временном ряде в целом. В соответствии с этим сформулируем условие, соблюдение которого поможет избежать описанных выше неадекватных результатов поиска сниппетов. Сниппет должен находиться как можно дальше от подпоследовательностей других активностей, которые содержит ряд, и как можно ближе к своим ближайшим соседям.

Однако в оригинальном алгоритме SnippetFinder указанное условие выполняется не всегда. Причина заключается в том, что в оригинальном алгоритме удаленность подпоследовательностей друг от друга определяется с

помощью расстояния MPdist, значение которого пропорционально количеству непрерывных промежутков в указанных подпоследовательностях, похожих друг на друга в смысле нормированной евклидовой метрики. В связи с этим возможна ситуация, когда алгоритм SnippetFinder в качестве сниппета может выбрать подпоследовательность, которая является аномальной. Далее приведен синтетический пример указанной ситуации, который иллюстрирует неадекватную работу оригинального алгоритма SnippetFinder в подобных случаях.

На рис. 5.4 представлен временной ряд, полученный путем конкатенации двух временных рядов-электрокардиограмм, взятых из открытого репозитория [47]. Соединенные ряды представляют собой ЭКГ разных пациентов (мужчина и женщина разных возрастов), символизируя таким образом две активности некоего субъекта в результирующем ряде. В полученном ряде с помощью оригинального алгоритма SnippetFinder найдены два сниппета, которые выделены в ряде на рис. 5.4. Для поиска сниппетов установлены следующие входные параметры алгоритма: $m = 250$, $\ell = 75$, $k = \lceil 0.1m \rceil$. Ниже временного ряда приведена разметка ЭКГ, выполненная кардиологом, фиксирующая наличие аритмии у пациента. Подпоследовательности ряда, которые отмечены как проявление аритмии, считаются аномалиями.

Рассматривая MPdist-профили двух сниппетов, которые представлены на рис. 5.4, можно видеть следующее. Первый найденный алгоритмом SnippetFinder сниппет относится к аномалии согласно разметке кардиолога, которая не отражает типичную активность, присутствующую в ряде. Однако, несмотря на неправильный выбор сниппета, полученная разметка ряда точно выделяет две активности, заданные указанным временным рядом.

В соответствии с этим далее предлагается улучшенный алгоритм поиска сниппетов, получивший название $\text{SnippetFinder}_{\text{Advanced}}$, в котором отбор MPdist-профилей гарантированно не приведет к неадекватному результату, аналогичному вышеописанному примеру.

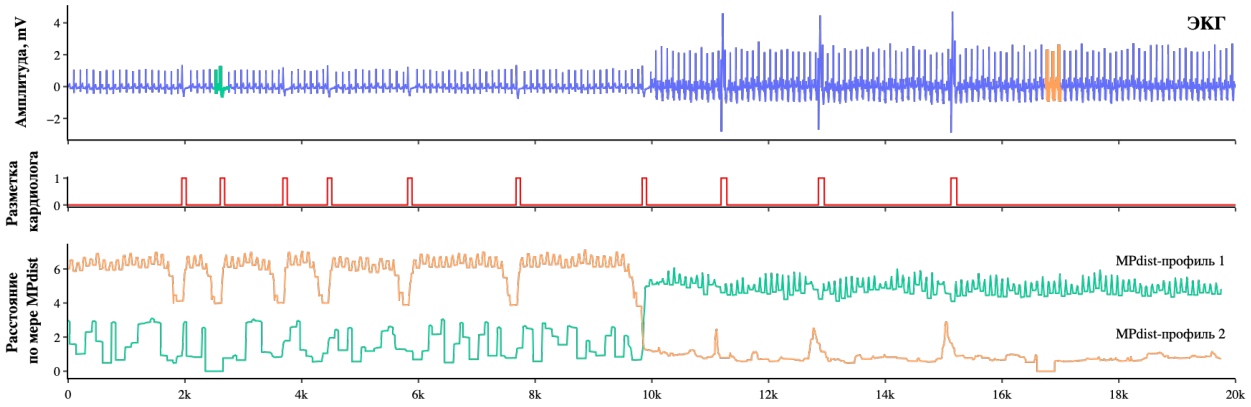


Рис. 5.4. Пример некорректных результатов работы алгоритма SnippetFinder

Алг. 5.2. SnippetFinderAdvanced (in T, m, k, K; out $C^m_{T\,\text{best}}$)

$C^m_{T\,\text{init}} \leftarrow \emptyset$; $C^m_{T\,\text{best}} \leftarrow \emptyset$; $D_{\text{subset}} \leftarrow \emptyset$; $MPD_{\text{diff}} \leftarrow [0]_{k\times k}$
$C^m_{T\,\text{init}} \leftarrow \text{SnippetFinder}(T, m, k)$ ▷ Оригинальный алгоритм [60]
$D_{\text{subset}} \leftarrow C^m_{T\,\text{init}}.profile$
for $i \leftarrow 1$ **to** n/m **do**
    for $j \leftarrow i+1$ **to** n/m **do**
        $MPD_{\text{diff}}(i, j) \leftarrow \sum_{u=1}^{n-m} |D_{\text{subset}}(i, u) - D_{\text{subset}}(j, u)|$
        $MPD_{\text{diff}}(j, i) \leftarrow MPD_{\text{diff}}(i, j)$
$Labels \leftarrow \text{KMeans}(MPD_{\text{diff}}, K)$
$Clusters \leftarrow \{Cluster_i \mid Cluster_i \leftarrow \emptyset,\ 1 \leqslant i \leqslant K\}$
for $i \leftarrow 1$ **to** k **do**
    $label \leftarrow Labels_i$
    $Cluster_{label} \leftarrow Cluster_{label} \cup i$
$CartProduct \leftarrow \text{CartesianProduct}(Clusters)$
$minProfile \leftarrow +\infty$
for $i \leftarrow 1$ **to** $|CartProduct|$ **do**
    $ProfileArea \leftarrow \sum_{u=1}^{n-m} \min\limits_{1\leqslant j\leqslant K} \left(D_{\text{subset}}(CartProduct_i(j), u)\right)$
    if $ProfileArea < minArea$ **then**
        $minProfile \leftarrow ProfileArea$
        $idx \leftarrow i$
$C^m_{T\,\text{best}} \leftarrow \{C_i \mid C_i \in C^m_{T\,\text{init}},\ i \in CartProduct_{idx}\}$
return $C^m_{T\,\text{best}}$

Псевдокод предлагаемого алгоритма представлен в алг. 5.2 и выполняется следующим образом. Сперва выполняется вызов оригинального алгоритма SnippetFinder [60], который находит множество $C^m_{T\,\text{init}}$, состоящее

из k сниппетов (см. строку 2 в алг. 5.2). Данный входной параметр k алгоритма $\text{SnippetFinder}_{\text{Advanced}}$ изначально задается аналитиком и должен превышать количество сниппетов K, реально отражающих все присутствующие в ряде активности.

Нахождение $C^m_{T\,\text{init}}$ выполняется следующим образом. Сначала формируется множество D, включающий все MPdist-профили сегментов временного ряда T:

$$D = \{D_i\}_{i=1}^{n/m}, \quad D_i = MPD(Seg_i, T, \ell). \tag{5.8}$$

Далее выполняется поиск k сниппетов, реализующийся как подбор набора MPdist-профилей из множества D, площадь под кривой репрезентативности M которых наименьшая. Нахождение такого набора организуется с помощью жадного подхода, использование которого было предложено в оригинальном алгоритме. Затем формируется подмножество D_{subset}, состоящее из MPdist-профилей найденных сниппетов $C^m_{T\,\text{init}}$ (см. строку 3 в алг. 5.2).

На следующем шаге вычисляется матрица $MPD_{\text{diff}} \in \mathbb{R}^{k\times k}$, в которой хранятся L1-расстояния между каждым MPdist-профилем и всеми остальными элементами подмножества D_{subset} (см. строки 4–9 в алг. 5.2). Элемент матрицы MPD_{diff} вычисляется следующим образом:

$$\begin{aligned} &MPD_{\text{diff}}(i, j) = dist_{L1}\big(D_{\text{subset}}(i, \cdot), D_{\text{subset}}(j, \cdot)\big) = \\ &= \sum_{u=1}^{n-m+1} |D_{\text{subset}}(i, u) - D_{\text{subset}}(j, u)|, \quad 1 \leqslant i, j \leqslant k. \end{aligned} \tag{5.9}$$

Матрица MPD_{diff} является квадратной и симметричной относительно главной диагонали. Для ускорения вычислений достаточно найти ее верхний треугольник, а нижний треугольник матрицы заполнить значениями верхнего треугольника следующим образом:

$$\forall i < j \quad MPD_{\text{diff}}(j, i) = MPD_{\text{diff}}(i, j). \tag{5.10}$$

После того, как вычислена матрица MPD_{diff}, выполняется кластеризация ее строк, например, с помощью алгоритма k-Means [90] или k-Medoids [65]. В результате кластеризации строки матрицы MPD_{diff} разбиваются на непустые подмножества (кластеры) $C = \{c_i\}_{i=1}^{K}$. При этом каждый кластер c_i будет состоять из объектов, близких по некоторой заданной метрике ρ, в то время как объекты из разных кластеров будут существенно отличаться. Типичной метрикой ρ, используемой для определения схожести объектов, является евклидово расстояние. В данном исследовании количество кластеров устанавливается равным количеству сниппетов K. В результате выполнения алгоритма кластеризации формируется вектор $Labels \in \mathbb{R}^k$, хранящий в себе метки кластеров $label$, $1 \leqslant label \leqslant K$, к которым принадлежат строки матрицы MPD_{diff} (см. строку 10 в алг. 5.2).

Далее на основе вектора меток *Labels*, полученного на предыдущем шаге, формируется множество кластеров $Clusters = \{Cluster_i\}_{i=1}^{K}$, где каждый кластер содержит порядковые номера строк матрицы MPD_{diff}, похожих друг на друга в смысле метрики ρ (см. строки 11–15 в алг. 5.2). Эти номера соответствуют номерам сниппетов из множества $C^m_{T\,\text{init}}$. Тем самым, в один кластер $Cluster_i \in Clusters$ попадут индексы сниппетов, которые выражают одну активность.

Затем с помощью выполнения операции декартова произведения над кластерами *Clusters* конструируется множество *CartProduct*. Элементами данного множества являются кортежи, построенные из K порядковых номеров сниппетов $C^m_{T\,\text{init}}$, принадлежащих различным кластерам (см. строку 16 в алг. 5.2). Мощность данного множества представляет собой число $\prod_{i=1}^{K} |Cluster_i|$. Более формально множество *CartProduct* можно определить следующим образом:

$$\begin{gathered} CartProduct = Cluster_1 \times \ldots \times Cluster_K = \\ = \{(cluster_1, \ldots, cluster_K) \,|\, \forall i \in [1, ..., K]\; cluster_i \in Cluster_i\}. \end{gathered} \tag{5.11}$$

После чего среди всех наборов MPdist-профилей, индексы которых записаны в соответствующем кортеже множества *CartProduct*, находится тот,

который имеет наименьшую площадью под кривой репрезентативности (см. строки 17–24 в алг. 5.2). В итоге на основе найденного набора MPdist-профилей формируется результирующее множество сниппетов $C^m_{T\,\text{best}}$ (см. строку 25 в алг. 5.2).

На рис. 5.5 представлены результаты выполнения модифицированного алгоритма $\text{SnippetFinder}_{\text{Advanced}}$ для поиска сниппетов во временном ряде ЭКГ. По результатам модифицированного алгоритма видно, что внедрение нового подхода в алгоритм SnippetFinder позволило найти сниппеты, являющиеся нормальными подпоследовательностями этого ряда и соответствующие типичным активностям.

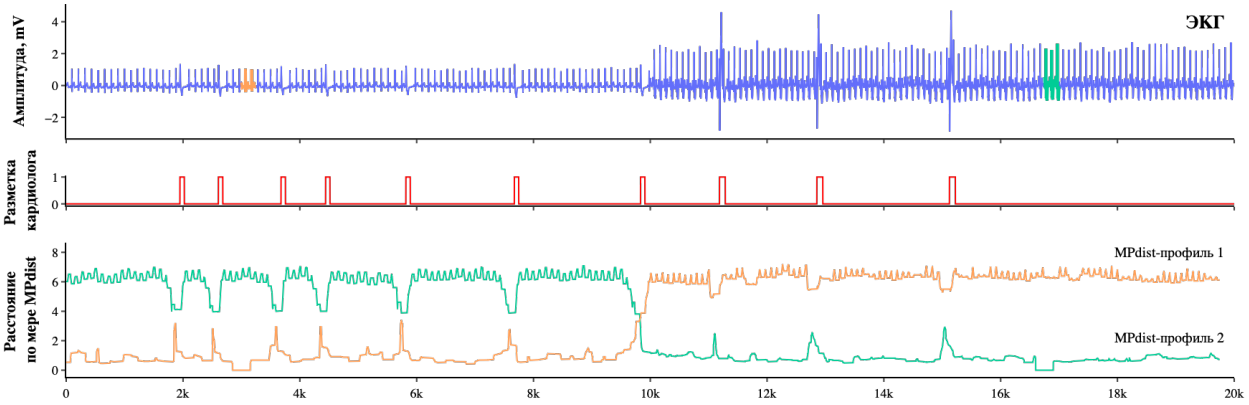


Рис. 5.5. Пример корректных результатов работы алгоритма SFA

5.3.2. Подбор гиперпараметров для расстояния MPdist

Одним из важных параметров алгоритма поиска сниппетов SnippetFinder, влияющего на точность разметки (аннотирования) временного ряда, является параметр k. Данный параметр используется для определения индекса элемента отсортированного по возрастанию ряда P_{ABBA}, полученного путем конкатенации двух матричных профилей P_{AB} и P_{BA}, вычисленных по формуле (1.16). k-й элемент ряда P_{ABBA} выступает в качестве MPdist расстояния между каждым сегментом ряда $Seg_i \in Seg^m_T$ и всеми его подпоследовательностями S^m_T.

В оригинальном алгоритме SnippetFinder [60] авторы рекомендуют для параметра k устанавливать значение, равное 5% от удвоенной длины сегмента и подпоследовательности [46], $k_{\text{default}} = \lceil 0.1m \rceil$. Выбор такого значения был установлен экспериментальным путем и делает меру MPdist инвариантной к шумам и искажениям, которые могут присутствовать в исходных данных.

В разработанном алгоритме формирования обучающей выборки предусматривается шаг очистки множества подпоследовательностей ряда от подпоследовательностей-шумов, которые могут содержаться в каждой активности. Под шумами понимаются ближайшие соседи сниппета, которые наиболее не похожи на него. Согласно алгоритму для нахождения шумов берутся MPdist-профили и далее извлекаются те подпоследовательности ряда, которые имеют наибольшие значения MPdist расстояний в соответствующем профиле сниппета. Для нахождения MPdist расстояний между сегментами и подпоследовательностями ряда в модифицированном алгоритме SnippetFinder предлагается параметру k присваивать значение, вычисленное по следующей выведенной формуле:

$$k = \max\left(1 + \frac{1 - 2\ell}{2(m - \ell + 1)},\ k_{\text{default}}\right). \tag{5.12}$$

При этом, если величина значимой длины подпоследовательности ℓ существенно близка к длине сегмента m, то параметру k присваивается типичное значение, равное 5% от $2m$, суммарной длины сегментов.

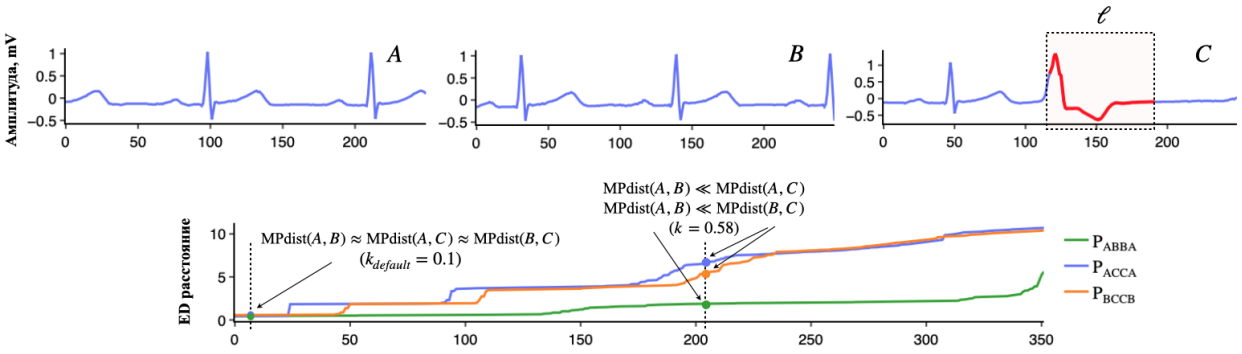


Рис. 5.6. Пример временных рядов для подбора гиперпараметра k

Предлагаемая формула (5.12) для нахождения параметра k получена на основе следующей эвристики. Рассмотрим три временных ряда A, B и C длины m, отражающих одинаковую активность (см. рис. 5.6). Предположим, что ряд C содержит некоторый аномальных участок длины ℓ. Тогда для того, чтобы определить, что C является шумом среди трех рассматриваемых рядов, необходимо установить такое значение параметра k, при котором выполняются следующие неравенства:

$$\text{MPdist}(A,\, B) \ll \text{MPdist}(A,\, C), \tag{5.13}$$

$$\text{MPdist}(A,\, B) \ll \text{MPdist}(B,\, C). \tag{5.14}$$

На основе формулы (1.15) построим для пар рядов A и B, A и C, B и C временные ряды P_{ABBA}, P_{ACCA} и P_{BCCB} соответственно, отсортировав их значения в порядке возрастания. Полученные ряды представлены на рис. 5.6. Далее проанализируем влияние параметра k на MPdist расстояние. Можно заметить, что при малых значениях параметра k (в том числе и при значении по умолчанию) MPdist расстояния между подпоследовательностями указанных выше пар имеют небольшую разницу, в то время как при больших значениях k существенно увеличивается разрыв между парами, где одна из подпоследовательностей является шумом, и парой подпоследовательностей, в которых отсутствует аномальный участок. Такая разница во втором случае объясняется тем, что выбор большого значения k увеличивает влияние аномального участка, присутствующего в C, на MPdist расстояние между A и C, B и C, несмотря на наличие в них подобных участков. Далее найдем такое значение k, которое зависит от длины значимого участка ℓ.

Рассмотрим в качестве примера временные ряды A и C. Обозначим за N количество подпоследовательностей длины ℓ, которые содержатся в каждом ряде A и C длины m, тогда $N = m - \ell + 1$. Далее определим количество нормальных и аномальных подпоследовательностей в A и C.

Поскольку по начальному предположению ряд A не содержит аномального участка, то количество нормальных подпоследовательностей в нем совпадает с общим числом подпоследовательностей и количество аномальных подпоследовательностей равно нулю, т.е. $N^A_{\text{norm}} = N$ и $N^A_{\text{anom}} = 0$ соответственно. Для вычисления количества аномальных подпоследовательностей в ряде C будем считать, что подпоследовательности, которые пересекаются с аномальным участком, то есть отстоят от нее не более чем на ℓ точек, также отмечаются как аномалии. С учетом этого, в C содержится нормальных и аномальных подпоследовательностей в количестве, равном $N^C_{\text{norm}} = m - 3\ell + 2$ и $N^C_{\text{anom}} = 2\ell - 1$ соответственно.

Далее, построив ряд P_{ACCA} для нахождения MPdist расстояния между A и C, вычислим долю элементов в ряде P_{ACCA}, которые являются расстояниями между всеми нормальными подпоследовательностями рядов A и C и их ближайшими соседями:

$$\begin{aligned} frac_{\text{norm}} = \frac{N^A_{\text{norm}} + N^C_{\text{norm}}}{|P_{ACCA}|} = \frac{(m - \ell + 1) + (m - 3\ell + 2)}{2(m - \ell + 1)} = \\ = \frac{2m - 4\ell + 3}{2(m - \ell + 1)} = 1 + \frac{1 - 2\ell}{2(m - \ell + 1)}. \end{aligned} \tag{5.15}$$

Таким образом, найденная доля нормальных подпоследовательностей $frac_{\text{norm}}$ по формуле (5.15) определяет нижнюю границу параметра k.

5.4. Вычислительные эксперименты

В данном разделе представлены результаты вычислительных экспериментов, проведенных на реальных временных рядах, для которых имеется предварительно выполненная разметка аномалий. В экспериментах выполняется сравнение точности предлагаемого метода DiSSiD с рассмотренными в обзоре известными аналогами (см. главу 1), относящихся к методам с частичным привлечением учителя. Помимо этого, исследуется влияние функции расстояния между векторными представлениями входных подпоследовательностей (метрики L1 и предложенным в данной работе исполь-

зованием меры MPdist [46]) на эффективность поиска аномалий с помощью метода DiSSiD. В заключении данного раздела выполняется оценка времени обучения и тестирования у модели DiSSiD и известных аналогов.

5.4.1. Описание экспериментов

Табл. 5.1. Наборы данных экспериментов с методом DiSSiD

№ п/п	Временной ряд	Предметная область	Длина ряда n	Параметры метода DiSSiD			
				Длина сниппета m	Длина значимого участка ℓ	Кол-во сниппетов K	Доля аномалий α, $\times 10^{-4}$
1	SMD	Мониторинг серверов	23 706	100	20	2	5
2	OPP	Распознавание физических активностей	26 204	200	50	2	5
3	Daphnet	Медицина	19 200	216	72	2	5
4	ECG-2 (803, 805)	Медицина	200 000	250	75	2	8
5	ECG-2 (803, 806)	Медицина	200 000	250	75	2	5
6	ECG-3 (803, 805, 806)	Медицина	300 000	250	75	3	10
7	MITDB	Медицина	200 000	520	75	2	2
8	IOPS	Мониторинг серверов	129 010	1000	500	2	1
9	YAHOO	Мониторинг серверов	1422	60	30	2	10

Наборы данных

Для проведения экспериментов по исследованию эффективности разработанного метода DiSSiD взяты временные ряды из различных предметных областей. Таблица 5.1 резюмирует характеристики исследуемых временных рядов и входные параметры, задаваемые для метода DiSSiD. Данные взяты из общедоступного фреймворка TSB-UAD [107], в котором собрано множество реализаций методов обнаружения аномалий во временных рядах для сравнительного анализа их точности. В данном фреймворке для каждого временного ряда имеется истинная разметка аномалий (каждая точка ряда имеет метку «норма» или «аномалия»).

Ряд Server Machine Dataset (SMD) [123] содержит показания потребления оперативной памяти сервером интернет-провайдера, собираемые в течение 5 недель работы сервера.

Ряд OPPORTUNITY (OPP) [112] представляет собой показания виброакселерометра, закрепленного на человеке, выполняющего в течение 2 час. различные виды утренней повседневной активности: подъем, перемещение по комнате, приготовление завтрака, уборка, отдых.

Ряд Daphnet [16] содержит показания виброакселерометра, закрепленного на пациенте с болезнью Паркинсона, для обнаружения у него симптомов замирания походки. Пациент выполнял следующие виды физической активности: ходьба по прямой, ходьба с поворотами, перемещения между разными комнатами с открыванием дверей и др.

Каждый из рядов ECG-2 (803, 805), ECG-2 (803, 806), ECG-3 (803, 805, 806) [47] и MITDB [97] представляет собой конкатенацию ЭКГ реальных пациентов, страдающих соответственно синдромом преждевременного сокращения желудочков и нарушениями сердечного ритма. Каждая из сцепленных ЭКГ отражает одну активность пациента.

Ряды IOPS [73] и YAHOO [81] представляют собой показания производительности серверов (количество операций ввода-вывода и обращений к памяти соответственно), которые установлены в компаниях Alibaba и Yahoo соответственно.

Аналоги

В экспериментах разработанная модель DiSSiD сравнивалась с известными аналогами, принадлежащими к группам методов обнаружения аномалий без учителя и с частичными привлечением учителя, рассмотренные в главе 1. В качестве методов без учителя выбраны IForest [88], LOF [27], MP [142], DAMP [92], NormA [23], PCA [10] и POLY [85], в качестве методов с частичным привлечением учителя — AE [115], Bagel [84], DeepAnT [99], IE-CAE [45], LSTM-AD [94], OceanWNN [132], OCSVM [117], TAnoGAN [19]. Реализация указанных методов взята из фреймворков TSB-UAD [107] и TimeEval [134]. При проведении экспериментов для каждого метода были установлены оптимальные значения гиперпараметров, подобранные авторами фреймворка TimeEval и представленные в работе [116]. Гиперпара-

метры настраивались автоматически таким образом, чтобы метод обнаружения аномалий достигал наибольшей средней точности по метрике AUC-ROC на всех синтетических временных рядах, сгенерированных с помощью фреймворка GutenTAG [134].

Кроме того, в сравнении участвовала версия DiSSiD, в которой модель выдает оценку схожести векторных представлений входных подпоследовательностей в виде евклидова расстояния вместо MPdist-расстояния.

Метрики сравнения

Для оценки точности метода DiSSiD используются специализированные метрики, разработанные для задачи поиска аномалий во временных рядах, когда аномалия представляет собой некоторый непрерывный промежуток ряда (подпоследовательность). К таким метрикам относятся R-AUC-ROC, R-AUC-PR, VUS-ROC, VUS-PR, предложенные в работе [106].

Метрики R-AUC-ROC и R-AUC-PR являются модификациями общепринятых метрик которые определяются как площадь под ROC-кривой (Receiver Operating Characteristics curve) и PR-кривой (Precision-Recall curve) соответственно. ROC-кривая отображает соотношение между долей неверно распознанных объектов (False Positive Rate, FPR) и долей верно распознанных объектов (True Positive Rate, TPR). PR-кривая определяется аналогично ROC-кривой, однако для построения PR-кривой вместо FPR и TPR используются точность (Precision) и полнота (Recall) [18]. Это позволяет получить более адекватную оценку качества работы модели, когда в наборе данных присутствует несбалансированность классов. Кроме того, метрики семейства AUC не требуют от аналитика установления порога аномальности, поскольку точность модели может существенно варьироваться при различных пороговых значениях.

Модификация метрик R-AUC-ROC и R-AUC-PR заключается в том, что границы каждой исходной аномальной подпоследовательности ряда расширяются с помощью добавления так называемой буферной зоны некоторой длины. Таком образом, все подпоследовательности, индексы которых

попадают в буферную зону, также считаются аномалиями. Таким подпоследовательностям назначается новая метка, принимающая значение в диапазоне от 0 до 1, в соответствии с их удаленностью от границы аномалии, вместо метки, обозначающей норму, из исходной разметки.

Чтобы устранить влияние ширины буферной зоны на точность обнаружения аномалий, определяемую описанными выше Range-AUC метриками, были предложены другие метрики VUS-ROC и VUS-PR [106]. Они позволяют оценить точность модели не только при разнообразных пороговых значениях оценки аномальности, но и при различных размерах буферной зоны. Таким образом, VUS-ROC и VUS-PR являются непараметризованными метриками, не зависящими от порога и устойчивыми к доле аномалий в ряде, лагам, шуму, которые могут присутствовать в векторе оценок аномальности, полученных для каждой подпоследовательности ряда с помощью модели.

Для нахождения метрик VUS-ROC и VUS-PR вычисляется объем под поверхностью VUS (Volume Under the Surface) в трехмерном пространстве: ROC-поверхность для VUS-ROC и PR-поверхность для VUS-PR. Первыми двумя измерениями пространства являются FPR, TPR для VUS-ROC и Precision, Recall для VUS-PR, третьим измерением — ширина буферной зоны.

Все описанные выше метрики точности принимают значения из отрезка $[0, 1]$, где большему значению соответствует лучшее качество, т.е. установленный порог точно отделяет аномальные подпоследовательности от нормальных на основе их оценок аномальности.

Аппаратная платформа

Вычислительные эксперименты выполнялись на вычислительном узле комплекса «Нейрокомпьютер» Суперкомпьютерного центра ЮУрГУ [2], который оснащен графическим процессором NVIDIA Tesla V100 SXM2, характеристики которого представлены в табл. 2.2.

5.4.2. Анализ результатов

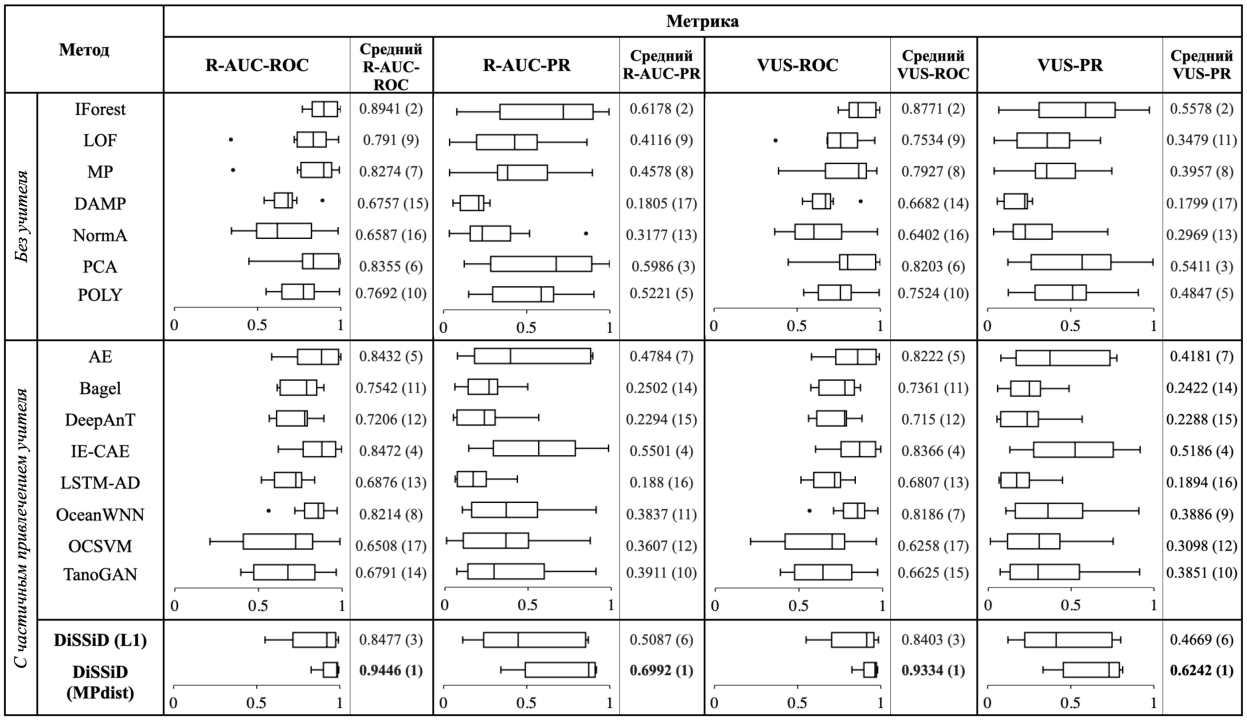


Рис. 5.7. Точность метода DiSSiD в сравнении с аналогами

Обобщенные результаты сравнения точности метода DiSSiD с известными аналогами представлены на рис. 5.7. Детализированные результаты этих экспериментов вынесены в Приложение А (см. табл. А.1, А.2, А.3, А.4), в которых приведены результаты сравнения по метрикам R-AUC-ROC, R-AUC-PR, VUS-ROC и VUS-PR соответственно.

Для компактного представления результатов экспериментов для каждого метода по каждой метрике была построена диаграмма размаха («ящики с усами») [39]. Диаграмма размаха представляет собой способ визуализации, изображающий одномерное распределение данных и показывающий нижний и верхний квартили, минимальное и максимальное значения в данных и выбросы. Линия внутри каждой диаграммы обозначает среднее значение в данных. Все диаграммы размаха представлены на рис. 5.7. Для их построений была выполнена группировка значений точности по метрике и методу, полученных по всем временным рядам. Для сравнения методов по

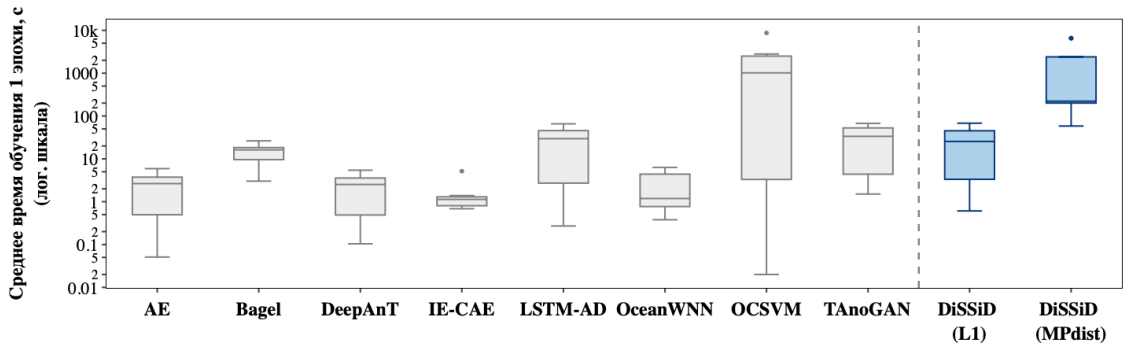
каждой метрике независимо выполняется сравнительный анализ распределений значений точности на основе соотвествующих диаграмм размаха.

Помимо диаграмм размаха, для каждого метода приведены средние значения точности по каждой метрике, вычисленные по всем временным рядам, взятых для экспериментов. В скобках указан ранг метода, определяющий его позицию среди всех аналогов по среднему значению той или иной метрики. Полужирным шрифтом обозначен метод, являющийся лучшим по среднему показателю метрики.

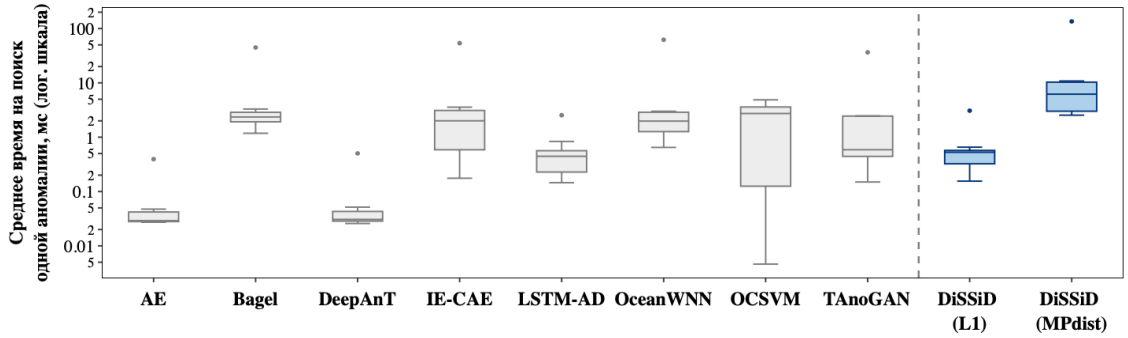
Можно видеть, что при применении евклидова расстояния в функции контрастных потерь метод DiSSiD в среднем входит в тройку лучших методов по точности обнаружения аномалий, вычисленной по метрикам R-AUC-ROC и VUS-ROC, и занимает 6 позицию по метрикам R-AUC-PR и VUS-PR. Однако использование модифицированной функции контрастных потерь с расстоянием MPdist в формуле (5.2) позволяет добиться лучшей в среднем точности обнаружения аномалий по всем метрикам. Тем самым, метод DiSSiD с функцией MPdist опережает как методы без учителя, так и с частичным привлечением учителя.

При этом можно заметить, что дисперсия распределения значений точности по всем метрикам, достигнутых методом DiSSiD, наименьшая по сравнению с дисперсиями распределений у известных аналогов. Это свидетельствует о том, что разброс значений точности относительно среднего значения небольшой и метод DiSSiD показывает близкую по значению точность при поиске аномалий во всех исследуемых временных рядах. В дополнение к этому отметим, что большинство значений у распределения метода DiSSiD превосходят значения точности известных аналогов.

Рисунок 5.8 показывает время работы метода DiSSiD по сравнению с известными аналогами. Можно видеть, что применение расстояния MPdist в функции контрастных потерь делает предложенную модель наиболее медленной как по времени ее обучения, так и по времени поиска аномалий. Причиной этого является применение в DiSSiD вычислительно затратной функции MPdist для нахождения расстояния между векторными представ-



а) обучение



б) тестирование

Рис. 5.8. Производительность метода DiSSiD в сравнении с аналогами

лениями входных подпоследовательностей (временная сложность указанной функции является кубической по отношению к длине подпоследовательности [60]). Низкое быстродействие является в данном случае обратной стороной высокой точности обнаружения аномалий (см. рис. 5.7).

Рассмотрим отдельно возможность применения модели DiSSiD в режиме реального времени. Из рис. 5.8б видно, что в экспериментах предложенная модель показывает среднее время поиска аномалий 0.1 с. Такое быстродействие модели допускает ее применение в режиме реального времени, что подтверждают следующие примеры. В системах автоматизации управления сеть передачи данных, обслуживающая датчики измерения температуры, влажности и давления, имеет цикл передачи данных до 100 мс [8]. Компания Emerson, один из ведущих мировых производителей измерительных систем, поставляет беспроводные температурные датчики, имеющие период обновления данных не менее 1 с [30].

5.5. Выводы по главе 5

Предложен новый метод поиска аномальных подпоследовательностей временного ряда с частичным привлечением учителя DiSSiD (Discord, Snippet, and Siamese Neural Network-based Detector of anomalies). Теоретическую основу метода составляют концепции диссонанса и сниппета, которые формализуют понятия аномальных и типичных подпоследовательностей временного ряда соответственно. Предложенный метод включает в себя нейросетевую модель, которая определяет степень аномальности входной подпоследовательности ряда, и алгоритм автоматизированного построения обучающей выборки для этой модели.

Нейросетевая модель представляет собой сиамскую нейронную сеть, где в качестве подсети фигурирует модификация нейросетевой модели ResNet. Для обучения модели предложена модифицированная функция контрастных потерь.

Входными данными для формирования обучающей выборки является репрезентативный временной ряд, адекватно отражающего типичную деятельность субъекта, противоположную аномалиям, которые предполагается обнаруживать с помощью модели. Формирование обучающей выборки включает в себя два шага: очистка и генерация. Очистка подразумевает формирование множества подпоследовательностей ряда, имеющих заданную аналитиком длину, и удаление из указанного множества диссонансов, маломощных сниппетов со своими ближайшими соседями и выбросов в рамках каждого сниппета, трактуемых соответственно как аномальная, нетипичная деятельность субъекта и шумы. Поиск сниппетов выполняется с помощью модифицированного параллельного алгоритма PSF. Модификация заключается в усовершенствовании процедуры выбора набора MPdist-профилей сегментов в качестве сниппетов, которые наилучшим образом определяют типичные активности. Также предлагается эвристика, на основе которой выполняется подбор гиперпараметров для модифицированного алгоритма поиска сниппетов.

Вычислительные эксперименты на реальных временных рядах из различных предметных областей показывают, что предлагаемый метод DiSSiD по сравнению с аналогами показывает в среднем наиболее высокую точность поиска аномалий по метрикам R-AUC-ROC, R-AUC-PR, VUS-ROC, VUS-PR. Обратной стороной высокой точности метода является большее по сравнению с известными аналогами время, которое затрачивается на обучение модели и распознавание аномалии. Тем не менее, метод DiSSiD обеспечивает быстродействие, достаточное для поиска аномальных подпоследовательностей в режиме реального времени.

Результаты, приведенные в этой главе, опубликованы в статье [3] и получено свидетельство Роспатента о государственной регистрации программы для ЭВМ [5].

Заключение

В заключение излагаются итоги диссертационной работы и ее основные отличия от предыдущих работ, даются рекомендации по использованию полученных результатов, а также рассмотрены направления дальнейших исследований в данной области.

Итоги исследования

Данная диссертационная работа посвящена разработке и исследованию новых методов и алгоритмов поиска аномальных подпоследовательностей временного ряда на платформе современных высокопроизводительных вычислительных систем. В настоящее время обеспечение эффективного поиска аномалий во временных рядах является актуальной проблемой, решение которой востребовано в широком спектре научных и практических приложений. Проведенное исследование базируется на концепции диссонанса временного ряда.

В результате исследования разработаны параллельные алгоритмы поиска диссонансов временного ряда PD3, PALMAD, PADDi и метод обнаружения аномалий в потоковом временном ряде DiSSiD. Алгоритм PD3 (Parallel DRAG-based Discord Discovery) обеспечивает поиск всех диссонансов временного ряда, имеющих заданную длину, на графическом процессоре. Алгоритм PALMAD (Parallel Arbitrary Length MERLIN-based Anomaly Discovery) обеспечивает поиск всех диссонансов временного ряда, имеющих длину в заданном диапазоне, на графическом процессоре. Алгоритм PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs) обеспечивает поиск всех диссонансов временного ряда, который не может быть целиком размещен в оперативной памяти, имеющих длину в заданном диапазоне, на высокопроизводительном вычислительном кластере с графическими процессорами. Метод DiSSiD (Discord, Snippet, and Siamese Neural Network-based Detector of anomalies) обеспечивает поиск аномалий потокового временного ряда и включает в себя нейросетевую модель и алго-

ритм построения обучающей выборки для указанной модели. Проведены вычислительные эксперименты с синтетическими и реальными временными рядами, подтверждающие эффективность алгоритмов PD3, PALMAD, PADDi и метода DiSSiD.

Применение полученных результатов

Результаты исследования могут быть применены для эффективного поиска аномалий во временных рядах в широком спектре предметных областей, поскольку разработанные параллельные алгоритмы поиска диссонансов временных рядов и метод обнаружения аномальных подпоследовательностей в потоковом временном ряде являются агностическими (не зависящими от предметной области) и требуют минимального участия аналитика-эксперта, от которого требуется задать длины искомых аномальных подпоследовательностей.

Отличия исследования от предыдущих работ

Основные результаты, полученные в ходе выполнения диссертационного исследования, являются новыми и не покрываются ранее опубликованными научными работами других авторов, обзор которых был дан выше в главе 1. Основные отличия заключаются в следующем.

1. Алгоритм PD3 поиска диссонансов фиксированной длины на графическом процессоре, в отличие от известных аналогов (алгоритма Жу и др. [143] и алгоритма KBF_GPU [127]) позволяет находить все диссонансы заданного временного ряда, имеющие заданную длину (а не один, наиболее важный из них). При этом PD3 опережает известные аналоги в разы по среднему времени поиска одного диссонанса. В соответствии с этим алгоритм PD3 будет более ценен, чем известные аналоги, в приложениях, где требуется поиск всех аномалий, имеющих фиксированную длину, в заданном временном ряде.

2. Алгоритмы PALMAD и PADDi поиска диссонансов произвольной длины соответственно на графическом процессоре и высокопроизводительном вычислительном кластере с графическими процессорами не имеют аналогов среди параллельных алгоритмов. Упомянутые выше алгоритм Жу и др. [143] и алгоритм KBF_GPU [127] являются относительно похожими, поскольку предназначены для поиска одного наиболее важного диссонанса. PALMAD и PADDi в сравнении с алгоритмами Жу и др. и KBF_GPU быстрее в разы по среднему времени поиска одного диссонанса. Рассмотренные ранее в обзоре в главе 1 алгоритмы DDD [136] и PDD [56] поиска диссонансов фиксированной длины на высокопроизводительных кластерах существенно (более чем на порядок) уступают алгоритму PADDi по быстродействию, поскольку предполагают интенсивные обмены данными между узлами кластера. В соответствии с этим алгоритмы PALMAD и PADDi будут более ценны, чем известные аналоги, в приложениях, где требуется поиск всех аномалий, имеющих произвольную длину, в заданном временном ряде.

3. Метод DiSSiD обнаружения аномалий потокового временного ряда применяет концепции диссонанса и сниппета [60], которые формализуют понятия аномальных и типичных подпоследовательностей временного ряда соответственно. Метод включает в себя нейросетевую модель распознавания аномальной подпоследовательности, представляющую собой сиамскую нейронную сеть [35], для обучения которой предложена модифицированная функция контрастных потерь, и в качестве подсети которой предложена модификация нейросетевой модели ResNet [52]. В соответствии с этим, в отличие от известных аналогов — методов с частичным привлечения учителя [55, 116], DiSSID позволяет выявлять аномальные подпоследовательности ряда, отражающие нетипичную и редко встречающуюся активности исследуемого субъекта.

Репродуцируемость результатов исследования

Исходные тексты программ, реализующих разработанные в рамках диссертационного исследования алгоритмы PD3, PALMAD и PADDi, метод DiSSiD, а также наборы данных, с которыми проведены вычислительные эксперименты, подтверждающие эффективность указанных разработок, размещены в сети Интернет в свободно доступных репозиториях [74–77].

Направления будущих исследований

Теоретические исследования и практические разработки, выполненные в рамках этой диссертационной работы, предполагается продолжить по следующим направлениям:

— обобщение разработанных моделей, методов и подходов на многомерные временные ряды;

— разработка версий предложенных параллельных алгоритмов для многоядерных центральных процессоров и программируемых логических интегральных схем.

Финансовая поддержка исследования

Диссертационное исследование выполнено при финансовой поддержке Российского научного фонда (грант № 23-21-00465).

Список литературы

1. Басалаев А.А. Автоматизированный энергоменеджмент теплоэнергетического комплекса университетского городка // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. 2015. Т. 15, № 4. С. 26–32. DOI: 10.14529/ctcr150403.

2. Биленко Р.В., Долганина Н.Ю., Иванова Е.В., Рекачинский А.И. Высокопроизводительные вычислительные ресурсы Южно-Уральского государственного университета // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. Т. 11, № 1. С. 15–30. DOI: 10.14529/cmse220102.

3. Краева Я.А. Обнаружение аномалий временного ряда на основе технологий интеллектуального анализа данных и нейронных сетей // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 3. С. 50–71. DOI: 10.14529/cmse230304.

4. Краева Я.А. Поиск аномалий в сенсорных данных цифровой индустрии с помощью параллельных вычислений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 2. С. 47–61. DOI: 10.14529/cmse230202.

5. Краева Я.А. Свидетельство Роспатента о государственной регистрации программы для ЭВМ «DiSSiD: детектор аномалий временного ряда в режиме реального времени» 2023685034 от 22.11.2023. 2023. URL: https://new.fips.ru/registers-doc-view/fips_servlet?DB=EVM&DocNumber=2023685034.

6. Краева Я.А., Цымблер М.Л. Поиск аномалий в больших временных рядах на кластере с GPU узлами // Вычислительные методы и программирование. 2023. Т. 24, № 3. С. 291–304. DOI: 10.26089/NumMet.v24r321.

7. Краева Я.А., Цымблер М.Л. Свидетельство Роспатента о государственной регистрации программы для ЭВМ «PALMAD: детектор аномалий различной длины во временном ряде на графическом процессоре» 2022667716 от 23.09.2022. 2022. URL: https://new.fips.ru/registers-doc-view/fips_servlet?DB=EVM&DocNumber=2022667716.

8. Лопухов И. Сети Real-Time Ethernet: от теории к практической реализации // СТА: Современные технологии автоматизациии. 2010. Т. 10, № 3. С. 8–15.

9. Цымблер М.Л. Параллельный алгоритм поиска диссонансов временного ряда для многоядерных ускорителей // Вычислительные методы и программирование. 2019. Т. 20, № 3. С. 211–223. DOI: 10.26089/NumMet.v20r320.

10. Aggarwal C.C. Outlier Analysis. 2st. Springer Cham, 2017. DOI: 10.1007/978-3-319-47578-3.

11. Akesson B., Nasri M., Nelissen G., *et al.* A comprehensive survey of industry practice in real-time systems // Real Time Syst. 2022. Vol. 58, no. 3. P. 358–398. DOI: 10.1007/S11241-021-09376-1.

12. Almeida A., Brás S., Sargento S., Pinto F.C. Time series big data: a survey on data stream frameworks, analysis and algorithms // J. Big Data. 2023. Vol. 10, no. 1. P. 83. DOI: 10.1186/S40537-023-00760-1.

13. Angiulli F., Pizzuti C. Fast Outlier Detection in High Dimensional Spaces // Principles of Data Mining and Knowledge Discovery, 6th European Conference, PKDD 2002, Helsinki, Finland, August 19-23, 2002, Proceedings. Vol. 2431 / ed. by T. Elomaa, H. Mannila, H. Toivonen. Springer, 2002. P. 15–26. (Lecture Notes in Computer Science). DOI: 10.1007/3-540-45681-3_2.

14. Arruda H.M., Bavaresco R.S., Kunst R., *et al.* Data Science Methods and Tools for Industry 4.0: A Systematic Literature Review and Taxonomy // Sensors. 2023. Vol. 23, no. 11. P. 5010. DOI: 10.3390/S23115010.

15. Avogadro P., Dominoni M.A. A fast algorithm for complex discord searches in time series: HOT SAX Time // Applied Intelligence. 2022. DOI: 10.1007/s10489-021-02897-z.

16. Bächlin M., Plotnik M., Roggen D., *et al.* Wearable assistant for Parkinson's disease patients with the freezing of gait symptom // IEEE Trans. Inf. Technol. Biomed. 2010. Vol. 14, no. 2. P. 436–446. DOI: 10.1109/TITB.2009.2036165.

17. Bacon D.F., Graham S.L., Sharp O.J. Compiler Transformations for High-Performance Computing // ACM Comput. Surv. 1994. Vol. 26, no. 4. P. 345–420. DOI: 10.1145/197405.197406.

18. Baeza-Yates R.A., Ribeiro-Neto B.A. Modern Information Retrieval. ACM Press / Addison-Wesley, 1999. URL: http://www.dcc.ufmg.br/irbook/.

19. Bashar M.A., Nayak R. TAnoGAN: Time Series Anomaly Detection with Generative Adversarial Networks // 2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020, Canberra, Australia, December 1-4, 2020. IEEE, 2020. P. 1778–1785. DOI: 10.1109/SSCI47803.2020.9308512.

20. Blázquez-Garcıéa A., Conde A., Mori U., Lozano J.A. A Review on Outlier/Anomaly Detection in Time Series Data // ACM Comput. Surv. 2021. Vol. 54, no. 3. 56:1–56:33. DOI: 10.1145/3444690.

21. Boniol P., Linardi M., Roncallo F., Palpanas T. Automated Anomaly Detection in Large Sequences // 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020. IEEE, 2020. P. 1834–1837. DOI: 10.1109/ICDE48307.2020.00182.

22. Boniol P., Linardi M., Roncallo F., Palpanas T. SAD: An Unsupervised System for Subsequence Anomaly Detection // 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020. IEEE, 2020. P. 1778–1781. DOI: 10.1109/ICDE48307.2020.00168.

23. Boniol P., Linardi M., Roncallo F., *et al.* Unsupervised and scalable subsequence anomaly detection in large data series // VLDB J. 2021. Vol. 30, no. 6. P. 909–931. DOI: 10.1007/S00778-021-00655-8.

24. Boniol P., Palpanas T. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series // Proc. VLDB Endow. 2020. Vol. 13, no. 11. P. 1821–1834. DOI: 10.14778/3384345.3384350.

25. Boniol P., Palpanas T., Meftah M., Remy E. GraphAn: Graph-based Subsequence Anomaly Detection // Proc. VLDB Endow. 2020. Vol. 13, no. 12. P. 2941–2944. DOI: 10.14778/3415478.3415514.

26. Boniol P., Paparrizos J., Palpanas T. New Trends in Time Series Anomaly Detection // Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023 / ed. by J. Stoyanovich, J. Teubner, N. Mamoulis, *et al.* OpenProceedings.org, 2023. P. 847–850. DOI: 10.48786/EDBT.2023.80.

27. Breunig M.M., Kriegel H., Ng R.T., Sander J. LOF: Identifying Density-Based Local Outliers // Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA / ed. by W. Chen, J.F. Naughton, P.A. Bernstein. ACM, 2000. P. 93–104. DOI: 10.1145/342009.335388.

28. Bu Y., Leung O.T., Fu A.W., *et al.* WAT: Finding top-k discords in time series database // Proceedings of the 7th SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA. 2007. P. 449–454. DOI: 10.1137/1.9781611972771.43.

29. Buu H.T.Q., Anh D.T. Time series discord discovery based on *i*SAX symbolic representation // 3rd International Conference on Knowledge and Systems Engineering, KSE 2011, Hanoi, Vietnam, October 14-17, 2011. 2011. P. 11–18. DOI: 10.1109/KSE.2011.11.

30. Catalogue 2021. Emerson temperature sensors. URL: https://www.c-o-k.ru/library/catalogs/emerson/110477.pdf (accessed: 3.9.2021).

31. Chandola V., Banerjee A., Kumar V. Anomaly detection: A survey // ACM Comput. Surv. 2009. Vol. 41, no. 3. 15:1–15:58. DOI: 10.1145/1541880.1541882.

32. Chandola V., Cheboli D., Kumar V. Detecting anomalies in a time series database. Retrieved from the University of Minnesota Digital Conservancy. 2009. accessed: 2022-04-12. https://hdl.handle.net/11299/215791.

33. Chau P.M., Duc B.M., Anh D.T. Discord detection in streaming time series with the support of R-tree // 2018 International Conference on Advanced Computing and Applications (ACOMP), 27-29 November 2018, Ho Chi Minh City, Vietnam. 2018. P. 96–103. DOI: 10.1109/ACOMP.2018.00023.

34. Chen X., Chen Y., He Z. Urban Traffic Speed Dataset of Guangzhou, China. 2018. DOI: 10.5281/zenodo.1205229.

35. Chicco D. Siamese Neural Networks: An Overview // Artificial Neural Networks / ed. by H. Cartwright. New York, NY: Springer US, 2021. P. 73–94. DOI: 10.1007/978-1-0716-0826-5_3.

36. Choi K., Yi J., Park C., Yoon S. Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines // IEEE Access. 2021. Vol. 9. P. 120043–120065. DOI: 10.1109/ACCESS.2021.3107975.

37. Dau H.A., Keogh E., Kamgar K., *et al.* The UCR Time Series Classification Archive. 2018. Accessed: 2022-03-19. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

38. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004. 2004. P. 137–150. URL: http://www.usenix.org/events/osdi04/tech/dean.html.

39. Du Toit S.H.C., Steyn A.G.W., Stumpf R.H. Graphical Exploratory Data Analysis. Berlin, Heidelberg: Springer-Verlag, 1986. DOI: 10.5555/18752.

40. Esling P., Agon C. Time-series Data Mining // ACM Comput. Surv. 2012. Vol. 45, no. 1. 12:1–12:34. DOI: 10.1145/2379776.2379788.

41. Ferrell B., Santuro S. NASA Shuttle Valve Data. 2005. Accessed: 2022-03-19. http://www.cs.fit.edu/~pkc/nasa/data/.

42. Fredkin E. Trie memory // Commun. ACM. 1960. Vol. 3, no. 9. P. 490–499. DOI: 10.1145/367390.367400.

43. Fu A.W., Leung O.T., Keogh E.J., Lin J. Finding time series discords based on Haar transform // Advanced Data Mining and Applications, 2nd International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006, Proceedings. 2006. P. 31–41. DOI: 10.1007/11811305_3.

44. Fu T.-C. A review on time series data mining // Eng. Appl. of AI. 2011. Vol. 24, no. 1. P. 164–181. DOI: 10.1016/j.engappai.2010.09.007.

45. Garcia G.R., Michau G., Ducoffe M., *et al.* Temporal signals to images: Monitoring the condition of industrial assets with deep learning image processing algorithms // Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability. 2022. Vol. 236, no. 4. P. 617–627. DOI: 10.1177/1748006X21994446.

46. Gharghabi S., Imani S., Bagnall A.J., *et al.* An ultra-fast time series distance measure to allow data mining in more complex real-world deployments // Data Min. Knowl. Discov. 2020. Vol. 34, no. 4. P. 1104–1135. DOI: 10.1007/s10618-020-00695-8.

47. Goldberger A.L., Amaral L.A.N., Glass L., *et al.* PhysioBank, PhysioToolkit, and PhysioNet components of a new research resource for complex physiologic signals // Circulation. 2000. Vol. 101, no. 23. P. 215–220. DOI: 10.1161/01.CIR.101.23.e215.

48. Guttman A. R-Trees: A Dynamic Index Structure for Spatial Searching // SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984 / ed. by B. Yormark. ACM Press, 1984. P. 47–57. DOI: 10.1145/602259.602266.

49. Hadsell R., Chopra S., LeCun Y. Dimensionality Reduction by Learning an Invariant Mapping // 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA. IEEE Computer Society, 2006. P. 1735–1742. DOI: 10.1109/CVPR.2006.100.

50. Han Z., Gao P., Wan F. Research on Data Mining and Visualization Technology // CONF-CDS 2021: The 2nd International Conference on Computing and Data Science, Stanford, CA, USA, January 28-30, 2021. 2021. 71:1–71:4. DOI: 10.1145/3448734.3450801.

51. Hawkins D.M. Identification of Outliers. Springer, 1980. (Monographs on Applied Probability and Statistics). DOI: 10.1007/978-94-015-3994-4.

52. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. IEEE Computer Society, 2016. P. 770–778. DOI: 10.1109/CVPR.2016.90.

53. Hebrail G., Berard A. Individual household electric power consumption. 2012. Accessed: 2023-04-18 DOI: 10.24432/C58K54.

54. Hochreiter S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions // Int. J. Uncertain. Fuzziness

Knowl. Based Syst. 1998. Vol. 6, no. 2. P. 107–116. DOI: 10.1142/S0218488598000094.

55. Hodge V.J., Austin J. A Survey of Outlier Detection Methodologies // Artif. Intell. Rev. 2004. Vol. 22, no. 2. P. 85–126. DOI: 10.1023/B:AIRE.0000045502.10941.a9.

56. Huang T., Zhu Y., Mao Y., *et al.* Parallel discord discovery // Advances in Knowledge Discovery and Data Mining - 20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, April 19-22, 2016, Proceedings, Part II. 2016. P. 233–244. DOI: 10.1007/978-3-319-31750-2_19.

57. Huang T., Zhu Y., Wu Y., Shi W. J-distance discord: An improved time series discord definition and discovery method // 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015. P. 303–310. DOI: 10.1109/ICDMW.2015.120.

58. Imani S., Keogh E.J. Matrix profile XIX: Time series semantic motifs: A new primitive for finding higher-level structure in time series // 2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019. 2019. P. 329–338. DOI: 10.1109/ICDM.2019.00043.

59. Imani S., Madrid F., Ding W., *et al.* Introducing time series snippets: A new primitive for summarizing long time series // Data Min. Knowl. Discov. 2020. Vol. 34, no. 6. P. 1713–1743. DOI: 10.1007/s10618-020-00702-y.

60. Imani S., Madrid F., Ding W., *et al.* Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining // 2018 IEEE International Conference on Big Knowledge, ICBK 2018, Singapore, November 17-18, 2018 / ed. by X. Wu, Y. Ong, C.C. Aggarwal, H. Chen. IEEE Computer Society, 2018. P. 382–389. DOI: 10.1109/ICBK.2018.00058.

61. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. Vol. 37 / ed. by F.R. Bach, D.M. Blei. JMLR.org, 2015. P. 448–456. (JMLR Workshop and Conference Proceedings). URL: http://proceedings.mlr.press/v37/ioffe15.html.

62. Ivanov S., Nikolskaya K., Radchenko G., *et al.* Digital twin of city: Concept overview // Proceedings of 2020 Global Smart Industry Conference, GloSIC 2020, Chelyabinsk, Russia, November 17-19, 2020. 2020. P. 178–186. DOI: 10.1109/GloSIC50886.2020.9267879.

63. Jagadish H.V., Koudas N., Muthukrishnan S. Mining Deviants in a Time Series Database // VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK / ed. by M.P. Atkinson, M.E. Orlowska, P. Valduriez, *et al.* Morgan Kaufmann, 1999. P. 102–113. URL: http://www.vldb.org/conf/1999/P9.pdf.

64. John C., Grossman M., McKercher T. Professional CUDA C Programming. 1st. GBR: Wrox Press Ltd., 2014. DOI: 10.5555/2935593.

65. Kaufman L., Rousseeuw P.J. Partitioning Around Medoids (Program PAM) // Finding Groups in Data. John Wiley & Sons, Inc., 2008. P. 68–125. DOI: 10.1002/9780470316801.ch2.

66. Kbaier Ben Ismail D., Lazure P., Puillat I. Statistical properties and time-frequency analysis of temperature, salinity and turbidity measured by the MAREL Carnot station in the coastal waters of Boulogne-sur-Mer (France) // Journal of Marine Systems. 2016. Vol. 162. P. 137–153. Progress in marine science supported by European joint coastal observation systems: The JERICO-RI research infrastructure DOI: 10.1016/j.jmarsys.2016.03.010.

67. Keogh E.J., Chakrabarti K., Pazzani M.J., Mehrotra S. Dimensionality reduction for fast similarity search in large time series databases // Knowl. Inf. Syst. 2001. Vol. 3, no. 3. P. 263–286. DOI: 10.1007/PL00011669.

68. Keogh E.J., Lin J., Fu A.W. HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence // Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA. IEEE Computer Society, 2005. P. 226–233. DOI: 10.1109/ICDM.2005.79.

69. Kirk D.B. NVIDIA CUDA software and GPU parallel computing architecture // Proceedings of the 6th International Symposium on Memory Management, ISMM 2007, Montreal, Quebec, Canada, October 21-22, 2007. 2007. P. 103–104. DOI: 10.1145/1296907.1296909.

70. Knorr E.M., Ng R.T. Finding Intensional Knowledge of Distance-Based Outliers // VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK / ed. by M.P. Atkinson, M.E. Orlowska, P. Valduriez, *et al.* Morgan Kaufmann, 1999. P. 211–222. URL: http://www.vldb.org/conf/1999/P21.pdf.

71. Komitova R., Raabe D., Rein R., Memmert D. Time Series Data Mining for Sport Data: a Review // Int. J. Comput. Sci. Sport. 2022. Vol. 21, no. 2. P. 17–31. DOI: 10.2478/IJCSS-2022-0008.

72. Koski A. Primitive coding of structural ECG features // Pattern Recognit. Lett. 1996. Vol. 17, no. 11. P. 1215–1222. DOI: 10.1016/0167-8655(96)00079-7.

73. KPI Anomaly Detection Dataset. 2018. URL: http://iops.ai/dataset_detail/?id=10 (accessed: 15.8.2023).

74. Kraeva Y. DiSSiD: Discord, Snippet, and Siamese Neural Network-based Detector of Anomalies. 2022. Accessed: 2023-12-10. https://github.com/kraevaya/DiSSiD.

75. Kraeva Y. PADDi: PALMAD-based Anomaly Discovery on Distributed GPUs. 2022. Accessed: 2023-12-10. https://github.com/kraevaya/PADDi.

76. Kraeva Y. PALMAD: Parallel MERLIN-based Anomaly Discovery algorithm for GPU. 2022. Accessed: 2023-12-10. https://github.com/kraevaya/PALMAD.

77. Kraeva Y. PD3: Parallel DRAG-based Discord Discovery algorithm for GPU. 2022. Accessed: 2023-12-10. https://github.com/kraevaya/PD3.

78. Kraeva Y., Zymbler M. A parallel discord discovery algorithm for a graphics processor // Pattern Recognition and Image Analysis. 2023. Vol. 33, no. 2. P. 101–112. DOI: 10.1134/S1054661823020062.

79. Kumar S., Tiwari P., Zymbler M.L. Internet of Things is a revolutionary approach for future technology enhancement: a review // J. Big Data. 2019. Vol. 6. P. 111. DOI: 10.1186/s40537-019-0268-2.

80. Laña I., Olabarrieta I., Vélez M., Del Ser J. On the imputation of missing data for road traffic forecasting: New insights and novel techniques // Transportation Research Part C: Emerging Technologies. 2018. Vol. 90. P. 18–33. DOI: 10.1016/j.trc.2018.02.021.

81. Laptev N., Amizadeh S., Billawala Y. S5 - A Labeled Anomaly Detection Dataset, version 1.0(16M). 2015. URL: https://webscope.sandbox.yahoo.com/catalog.php?%20datatype=s&did=70 (accessed: 15.8.2023).

82. Li G., Bräysy O., Jiang L., *et al.* Finding time series discord based on bit representation clustering // Knowl.-Based Syst. 2013. Vol. 54. P. 243–254. DOI: 10.1016/j.knosys.2013.09.015.

83. Li J., Pedrycz W., Jamal I. Multivariate time series anomaly detection: A framework of Hidden Markov Models // Appl. Soft Comput. 2017. Vol. 60. P. 229–240. DOI: 10.1016/j.asoc.2017.06.035.

84. Li Z., Chen W., Pei D. Robust and Unsupervised KPI Anomaly Detection Based on Conditional Variational Autoencoder // 37th IEEE International Performance Computing and Communications Conference, IPCCC 2018, Orlando, FL, USA, November 17-19, 2018. IEEE, 2018. P. 1–9. DOI: 10.1109/PCCC.2018.8710885.

85. Li Z., Ma H., Mei Y. A Unifying Method for Outlier and Change Detection from Data Streams Based on Local Polynomial Fitting // Advances in Knowledge Discovery and Data Mining, 11th Pacific-Asia Conference, PAKDD 2007, Nanjing, China, May 22-25, 2007, Proceedings. Vol. 4426 / ed. by Z. Zhou, H. Li, Q. Yang. Springer, 2007. P. 150–161. (Lecture Notes in Computer Science). DOI: 10.1007/978-3-540-71701-0_17.

86. Lin J., Keogh E.J., Fu A.W., Herle H.V. Approximations to magic: Finding unusual medical time series // 18th IEEE Symposium on Computer-Based Medical Systems (CBMS 2005), 23-24 June 2005, Dublin, Ireland. 2005. P. 329–334. DOI: 10.1109/CBMS.2005.34.

87. Lin J., Keogh E.J., Lonardi S., Chiu B.Y. A symbolic representation of time series, with implications for streaming algorithms // Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD 2003, San Diego, California, USA, June 13, 2003. 2003. P. 2–11. DOI: 10.1145/882082.882086.

88. Liu F.T., Ting K.M., Zhou Z. Isolation Forest // Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy. IEEE Computer Society, 2008. P. 413–422. DOI: 10.1109/ICDM.2008.17.

89. Liu Y. Scalable Multivariate Time-Series Models for Climate Informatics // Comput. Sci. Eng. 2015. Vol. 17, no. 6. P. 19–26. DOI: 10.1109/MCSE.2015.126.

90. Lloyd S.P. Least squares quantization in PCM // IEEE Trans. Inf. Theory. 1982. Vol. 28, no. 2. P. 129–136. DOI: 10.1109/TIT.1982.1056489.

91. Lök S., Karabatak M. Earthquake Prediction by Using Time Series Analysis // 9th International Symposium on Digital Forensics and Security, ISDFS 2021, Elazig, Turkey, June 28-29, 2021 / ed. by A. Varol, M. Karabatak, I. Varol. IEEE, 2021. P. 1–6. DOI: 10.1109/ISDFS52919.2021.9486358.

92. Lu Y., Wu R., Mueen A., *et al.* DAMP: accurate time series anomaly detection on trillions of datapoints and ultra-fast arriving data streams // Data Min. Knowl. Discov. 2023. Vol. 37, no. 2. P. 627–669. DOI: 10.1007/S10618-022-00911-7.

93. Mackey M.C., Glass L. Oscillation and chaos in physiological control systems // Science. 1977. Vol. 197, no. 4300. P. 287–289. DOI: 10.1126/science.267326.

94. Malhotra P., Vig L., Shroff G.M., Agarwal P. Long Short Term Memory Networks for Anomaly Detection in Time Series // 23rd European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015. 2015. URL: http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf.

95. Marteau P., Soheily-Khah S., Béchet N. Hybrid Isolation Forest - Application to Intrusion Detection // CoRR. 2017. Vol. abs/1705.03800. arXiv: 1705.03800. URL: http://arxiv.org/abs/1705.03800.

96. Mittal S., Vaishay S. A survey of techniques for optimizing deep learning on GPUs // J. Syst. Archit. 2019. Vol. 99. DOI: 10.1016/J.SYSARC.2019.101635.

97. Moody G., Mark R. The impact of the MIT-BIH Arrhythmia Database // IEEE Engineering in Medicine and Biology Magazine. 2001. Vol. 20, no. 3. P. 45–50. DOI: 10.1109/51.932724.

98. Mueen A., Nath S., Liu J. Fast approximate correlation for massive time-series data // Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana,

USA, June 6-10, 2010 / ed. by A.K. Elmagarmid, D. Agrawal. ACM, 2010. P. 171–182. DOI: 10.1145/1807167.1807188.

99. Munir M., Siddiqui S.A., Dengel A., Ahmed S. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series // IEEE Access. 2019. Vol. 7. P. 1991–2005. DOI: 10.1109/ACCESS.2018.2886457.

100. Mutschler C., Ziekow H., Jerzak Z. The DEBS 2013 grand challenge // The 7th ACM International Conference on Distributed Event-Based Systems, DEBS'13, Arlington, TX, USA, June 29 - July 03, 2013 / ed. by S. Chakravarthy, S.D. Urban, P.R. Pietzuch, E.A. Rundensteiner. ACM, 2013. P. 289–294. DOI: 10.1145/2488222.2488283.

101. Nakamura T., Imamura M., Mercer R., Keogh E.J. MERLIN: Parameter-free discovery of arbitrary length anomalies in massive time series archives // 20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020. 2020. P. 1190–1195. DOI: 10.1109/ICDM50108.2020.00147.

102. Nakamura T., Mercer R., Imamura M., Keogh E.J. MERLIN++: Parameter-free discovery of time series anomalies // Data Min. Knowl. Discov. 2023. Vol. 37, no. 2. P. 670–709. DOI: 10.1007/s10618-022-00876-7.

103. NVIDIA. CUDA C++ Best Practices Guide. Release 12.3. 2023. URL: https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf.

104. Orchard M.T. A fast nearest-neighbor search algorithm // 1991 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '91, Toronto, Ontario, Canada, May 14-17, 1991. IEEE Computer Society, 1991. P. 2297–2300. DOI: 10.1109/ICASSP.1991.150755.

105. Page E.S. On Problems in which a Change in a Parameter Occurs at an Unknown Point // Biometrika. 1957. Vol. 44, no. 1/2. P. 248–252. DOI: 10.2307/2333258.

106. Paparrizos J., Boniol P., Palpanas T., *et al.* Volume Under the Surface: A New Accuracy Evaluation Measure for Time-Series Anomaly Detection // Proc. VLDB Endow. 2022. Vol. 15, no. 11. P. 2774–2787. DOI: 10.14778/3551793.3551830.

107. Paparrizos J., Kang Y., Boniol P., *et al.* TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection // Proc. VLDB Endow. 2022. Vol. 15, no. 8. P. 1697–1711. DOI: 10.14778/3529337.3529354.

108. Pearson K. The problem of the random walk // Nature. 1905. Vol. 72, no. 1865. P. 294. DOI: 10.1038/072342a0.

109. Pfeilschifter G. Time Series Analysis with Matrix Profile on HPC Systems // Master thesis, Department of Informatics, Technical University of Munich, Germany. 2019. URL: http://mediatum.ub.tum.de/doc/1471292/1471292.pdf Accessed: 2022-03-17.

110. Ramaswamy S., Rastogi R., Shim K. Efficient Algorithms for Mining Outliers from Large Data Sets // Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA / ed. by W. Chen, J.F. Naughton, P.A. Bernstein. ACM, 2000. P. 427–438. DOI: 10.1145/342009.335437.

111. Reyes R., López-Rodrıéguez I., Fumero J.J., Sande F. de. A preliminary evaluation of OpenACC implementations // J. Supercomput. 2013. Vol. 65, no. 3. P. 1063–1075. DOI: 10.1007/s11227-012-0853-z.

112. Roggen D., Calatroni A., Rossi M., *et al.* Collecting complex activity datasets in highly rich networked sensor environments // Seventh International Conference on Networked Sensing Systems, INSS 2010, Kassel, Germany, June 15-18, 2010. IEEE, 2010. P. 233–240. DOI: 10.1109/INSS.2010.5573462.

113. Rosenthal D. CVC Technology on Hot and Cold Strip Rolling Mills // Rev. Met. Paris. 1988. Vol. 85, no. 7. P. 597–606. DOI: 10.1051/metal/198885070597.

114. Ryzhikov A., Borisyak M., Ustyuzhanin A., Derkach D. Normalizing flows for deep anomaly detection // CoRR. 2019. Vol. abs/1912.09323. arXiv: 1912.09323. URL: http://arxiv.org/abs/1912.09323.

115. Sakurada M., Yairi T. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction // Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, Gold Coast, Australia, QLD, Australia, December 2, 2014 / ed. by A. Rahman, J.D. Deng, J. Li. ACM, 2014. P. 4. DOI: 10.1145/2689746.2689747.

116. Schmidl S., Wenig P., Papenbrock T. Anomaly Detection in Time Series: A Comprehensive Evaluation // Proc. VLDB Endow. 2022. Vol. 15, no. 9. P. 1779–1797. DOI: 10.14778/3538598.3538602.

117. Schölkopf B., Williamson R.C., Smola A.J., *et al.* Support Vector Method for Novelty Detection // Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999] / ed. by S.A. Solla, T.K. Leen, K. Müller. The MIT Press, 1999. P. 582–588. URL: http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.

118. Senin P., Lin J., Wang X., *et al.* Time series anomaly discovery with grammar-based compression // Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015 / ed. by G. Alonso, F. Geerts, L. Popa, *et al.* OpenProceedings.org, 2015. P. 481–492. DOI: 10.5441/002/edbt.2015.42.

119. Shewhart W.A. The Economic Control of Quality of Manufactured Product // Journal of the Royal Statistical Society. 1932. Vol. 95. P. 546. DOI: 10.2307/2342413.

120. Shieh J., Keogh E.J. *i*SAX: Indexing and mining terabyte sized time series // Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008. 2008. P. 623–631. DOI: 10.1145/1401890.1401966.

121. Snir M. Technical perspective: The future of MPI // Commun. ACM. 2018. Vol. 61, no. 10. P. 105. DOI: 10.1145/3264415.

122. Son N.T. An improvement of disk aware discord discovery algorithm for discovering time series discord // 2020 5th International Conference on Green Technology and Sustainable Development (GTSD), November 27-28, 2020, Ho Chi Minh City, Vietnam. 2020. P. 19–23. DOI: 10.1109/GTSD50082.2020.9303111.

123. Su Y., Zhao Y., Niu C., *et al.* Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network // Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019. ACM, 2019. P. 2828–2837. DOI: 10.1145/3292500.3330672.

124. Supinski B.R. de, Scogland T.R.W., Duran A., *et al.* The Ongoing Evolution of OpenMP // Proc. IEEE. 2018. Vol. 106, no. 11. P. 2004–2019. DOI: 10.1109/JPROC.2018.2853600.

125. Thill M., Konen W., Bäck T. Markus-Thill/MGAB: The Mackey–Glass anomaly benchmark. Version v1.0.1. 2020. Accessed: 2023-04-18 DOI: 10.5281/ZENODO.3762385.

126. Thuy H.T.T., Anh D.T., Chau T.N.V. An effective and efficient hash-based algorithm for time series discord discovery // 2016 3rd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS), 14-16 September 2016, Danang, Vietnam. 2016. P. 85–90. DOI: 10.1109/NICS.2016.7725673.

127. Thuy T.T.H., Anh T.D., Chau T.N.V. A new discord definition and an efficient time series discord detection method using GPUs // ICSED

2021: 2021 3rd International Conference on Software Engineering and Development, November 19-21, 2021, Xiamen, China. 2021. P. 63–70. DOI: 10.1145/3507473.3507483.

128. Top50: Rank list of most powerful Russian computers. 2023. Accessed: 2023-03-28. https://top50.supercomputers.ru/list.

129. Voevodin V.V., Antonov A.S., Nikitenko D.A., *et al.* Supercomputer Lomonosov-2: Large scale, deep monitoring and fine analytics for the user community // Supercomput. Front. Innov. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.

130. Volkov I., Radchenko G.I., Tchernykh A. Digital Twins, Internet of Things and Mobile Medicine: A Review of Current Platforms to Support Smart Healthcare // Program. Comput. Softw. 2021. Vol. 47, no. 8. P. 578–590. DOI: 10.1134/S0361768821080284.

131. Wang J.T., Wang X., Lin K., *et al.* Evaluating a Class of Distance-Mapping Algorithms for Data Mining and Clustering // Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999 / ed. by U.M. Fayyad, S. Chaudhuri, D. Madigan. ACM, 1999. P. 307–311. DOI: 10.1145/312129.312264.

132. Wang Y., Han L., Liu W., *et al.* Study on wavelet neural network based anomaly detection in ocean observing data series // Ocean Engineering. 2019. Vol. 186. P. 106129. DOI: 10.1016/j.oceaneng.2019.106129.

133. Wei L., Keogh E., Xi X. SAXually Explicit Images: Finding Unusual Shapes // 6th International Conference on Data Mining (ICDM'06). 2006. P. 711–720. DOI: 10.1109/ICDM.2006.138.

134. Wenig P., Schmidl S., Papenbrock T. TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms // Proc. VLDB Endow. 2022. Vol. 15, no. 12. P. 3678–3681. DOI: 10.14778/3554821.3554873.

135. Wijk J.J. van, Selow E.R. van. Cluster and calendar based visualization of time series data // IEEE Symposium on Information Visualization 1999 (INFOVIS'99), San Francisco, California, USA, October 24-29, 1999. IEEE Computer Society, 1999. P. 4–9. DOI: 10.1109/INFVIS.1999.801851.

136. Wu Y., Zhu Y., Huang T., *et al.* Distributed discord discovery: Spark based anomaly detection in time series // 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015. 2015. P. 154–159. DOI: 10.1109/HPCC-CSS-ICESS.2015.228.

137. Xing Y., Liu F., Xiao N., *et al.* Capability for Multi-Core and Many-Core Memory Systems: A Case-Study With Xeon Processors // IEEE Access. 2019. Vol. 7. P. 47655–47662. DOI: 10.1109/ACCESS.2018.2881460.

138. Yankov D., Keogh E.J., Rebbapragada U. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets // Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA. 2007. P. 381–390. DOI: 10.1109/ICDM.2007.61.

139. Yankov D., Keogh E.J., Rebbapragada U. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets // Knowl. Inf. Syst. 2008. Vol. 17, no. 2. P. 241–262. DOI: 10.1007/s10115-008-0131-9.

140. Ye N., Chen Q. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems // Quality and Reliability Engineering International. 2001. Vol. 17, no. 2. P. 105–112. DOI: 10.1002/qre.392.

141. Yeh C.M., Zhu Y., Ulanova L., *et al.* Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets // IEEE 16th International Conference on Data Mining,

ICDM 2016, December 12-15, 2016, Barcelona, Spain / ed. by F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, *et al.* IEEE Computer Society, 2016. P. 1317–1322. DOI: 10.1109/ICDM.2016.0179.

142. Yeh C.M., Zhu Y., Ulanova L., *et al.* Time series joins, motifs, discords and shapelets: A unifying view that exploits the matrix profile // Data Min. Knowl. Discov. 2018. Vol. 32, no. 1. P. 83–123. DOI: 10.1007/s10618-017-0519-9.

143. Zhu B., Jiang Y., Gu M., Deng Y. A GPU acceleration framework for motif and discord based pattern mining // IEEE Trans. Parallel Distributed Syst. 2021. Vol. 32, no. 8. P. 1987–2004. DOI: 10.1109/TPDS.2021.3055765.

144. Zhu C., Liu Y., Wang L., Lu J. Economic Data Prediction Based on Time Series Analysis // 16th International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2021, Chengdu, China, November 26-28, 2021 / ed. by S. Chen, J. Hu, T. Li, *et al.* IEEE, 2021. P. 591–597. DOI: 10.1109/ISKE54062.2021.9755403.

145. Zhu Y., Imamura M., Nikovski D., Keogh E.J. Introducing time series chains: A new primitive for time series data mining // Knowl. Inf. Syst. 2019. Vol. 60, no. 2. P. 1135–1161. DOI: 10.1007/s10115-018-1224-8.

146. Zhu Y., Yeh C.M., Zimmerman Z., *et al.* Matrix profile XI: SCRIMP++: Time series motif discovery at interactive speeds // IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018. 2018. P. 837–846. DOI: 10.1109/ICDM.2018.00099.

147. Zimmerman Z., Kamgar K., Senobari N.S., *et al.* Matrix profile XIV: Scaling time series motif discovery with GPUs to break a quintillion pairwise comparisons a day and beyond // Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019. 2019. P. 74–86. DOI: 10.1145/3357223.3362721.

148. Zymbler M., Grents A., Kraeva Y., Kumar S. A parallel approach to discords discovery in massive time series data // Computers, Materials & Continua. 2021. Vol. 66, no. 2. P. 1867–1878. DOI: 10.32604/cmc.2020.014232.

149. Zymbler M., Kraeva Y. High-Performance Time Series Anomaly Discovery on Graphics Processors // Mathematics. 2023. Vol. 11, no. 14. DOI: 10.3390/math11143193.

150. Zymbler M., Kraeva Y., Latypova E., *et al.* Cleaning Sensor Data in Smart Heating Control System // Proceedings of 2020 Global Smart Industry Conference, GloSIC 2020, Chelyabinsk, Russia, November 17-19, 2020. 2020. P. 375–381. DOI: 10.1109/GloSIC50886.2020.9267813.

151. Zymbler M., Polyakov A., Kipnis M. Time series discord discovery on Intel many-core systems // 13th International Conference, PCT 2019, Kaliningrad, Russia, April 2-4, 2019, Revised Selected Papers. Communications in Computer and Information Science. Vol. 1063 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2019. P. 168–182. DOI: 10.1007/978-3-030-28163-2_12.

Список рисунков

Список таблиц

Список алгоритмов

Приложение А. Сравнение точности DiSSiD с аналогами

Ниже в табл. А.1, А.2, А.3, А.4 приведены результаты сравнения точности DiSSiD обнаружения аномалий с аналогами по метрикам R-AUC-ROC, R-AUC-PR, VUS-ROC и VUS-PR соответственно. В ячейке таблицы дано значение по соответствующей метрике и в скобках — ранг метода, указанного в соответствующей строке, среди всех аналогов на временном ряде, указанном в соответствующем столбце. Полужирным шрифтом даны результат и место лучшего метода на заданном временном ряде. Два последних столбца каждой таблицы являются резюмирующими, в них указаны соответственно средние значения метрики и ранга по всем временным рядам, задействованным в экспериментах, а также в скобках — среднее значение метрики и ранга метода.

Можно видеть, что при применении евклидова расстояния в функции контрастных потерь метод DiSSiD в среднем входит в тройку лучших методов по точности обнаружения аномалий. Однако использование модифицированной функции контрастных потерь с расстоянием MPdist в формуле (5.2) позволяет добиться лучшей в среднем точности обнаружения аномалий.

Табл. A.1. Сравнение точности DiSSiD с аналогами (метрика R-AUC-ROC)

	Метод	Временные ряды										
		SMD	**OPP**	**Daphnet**	**ECG-2 (803, 805)**	**ECG-2 (803, 806)**	**ECG-3 (803, 805, 806)**	**MITDB**	**IOPS**	**YAHOO**	**Средний VUS-ROC**	**Средний ранг**
Без учителя	**IForest**	0.8789 (11)	0.9998 (2)	0.7724 (5)	0.8987 (4)	0.9109 (4)	0.8473 (4)	0.7693 (4)	0.9765 (4)	**0.9928 (1)**	0.8941 (2)	4.33 (2)
	LOF	0.9332 (6)	0.3395 (16)	**0.8710 (1)**	0.7837 (11)	0.9053 (5)	0.8348 (5)	0.7202 (6)	0.7442 (12)	0.9870 (5)	0.7910 (9)	6.33 (4)
	MP	0.9667 (3)	0.3525 (14)	0.8552 (2)	0.7397 (12)	0.9364 (3)	0.7680 (11)	0.8981 (2)	0.9385 (8)	0.9916 (3)	0.8274 (7)	6.45 (5)
	DAMP	0.8902 (9)	0.5519 (13)	0.6998 (7)	0.6644 (15)	0.7361 (16)	0.6988 (14)	0.5392 (15)	0.6834 (14)	0.6178 (15)	0.6757 (15)	13.11 (16)
	NormA	**0.9844 (1)**	0.3428 (15)	0.6185 (10)	0.5289 (17)	0.7960 (11)	0.3904 (17)	0.7639 (5)	0.9078 (9)	0.5952 (16)	0.6587 (16)	11.22 (11)
	PCA	0.9497 (5)	**0.9999 (1)**	0.4479 (16)	0.8201 (6)	0.8353 (8)	0.7968 (10)	0.6908 (8)	0.9918 (2)	0.9872 (4)	0.8355 (6)	6.67 (6)
	POLY	0.6534 (16)	0.9926 (3)	0.5509 (14)	0.7996 (8)	0.8003 (10)	0.7755 (12)	0.6244 (10)	0.7650 (11)	0.9607 (8)	0.7692 (10)	10.22 (10)
С частичным привлечением учителя	**AE**	0.8933 (8)	0.7224 (9)	0.7377 (6)	0.9793 (3)	0.8768 (7)	0.9783 (2)	0.5791 (14)	0.8299 (10)	0.9920 (2)	0.8432 (5)	6.78 (7)
	Bagel	0.8694 (12)	nan (17)	0.6120 (11)	0.8032 (7)	0.7718 (15)	0.8269 (7)	0.6472 (9)	0.6116 (15)	0.8911 (11)	0.7542 (11)	11.56 (12)
	DeepAnT	0.7765 (14)	0.5646 (11)	0.6874 (9)	0.7857 (10)	0.7897 (13)	0.7974 (9)	0.6198 (11)	0.5736 (16)	0.8909 (12)	0.7206 (12)	11.67 (13)
	IE-CAE	0.9062 (7)	0.9890 (4)	0.6930 (8)	0.7919 (9)	0.8788 (6)	0.7979 (8)	0.6184 (12)	**0.9978 (1)**	0.9516 (9)	0.8472 (4)	7.11 (8)
	LSTM-AD	0.8355 (13)	0.5589 (12)	0.6679 (12)	0.7391 (13)	0.7232 (17)	0.7245 (13)	0.6079 (13)	0.5175 (17)	0.8138 (13)	0.6876 (13)	13.67 (17)
	OceanWNN	0.8887 (10)	0.8816 (7)	0.5606 (13)	0.8573 (5)	0.7949 (12)	0.8295 (6)	0.7174 (7)	0.9699 (6)	0.8926 (10)	0.8214 (8)	8.45 (9)
	OCSVM	0.2102 (17)	0.7477 (8)	0.4376 (17)	0.7221 (14)	0.7740 (14)	0.6810 (15)	0.3254 (17)	0.9719 (5)	0.9869 (6)	0.6508 (17)	12.56 (14)
	TAnoGAN	0.6756 (15)	0.9402 (5)	0.4905 (15)	0.6270 (16)	0.8012 (9)	0.3947 (16)	0.4156 (16)	0.9648 (7)	0.8026 (14)	0.6791 (14)	12.56 (14)
	DiSSiD (L1)	0.9549 (4)	0.6922 (10)	0.8376 (3)	0.9813 (2)	0.9915 (2)	0.9729 (3)	**0.9222 (1)**	0.7265 (13)	0.5498 (17)	0.8477 (3)	6.11 (3)
	DiSSiD (MPdist)	0.9695 (2)	0.9188 (6)	0.8274 (4)	**0.9850 (1)**	**0.9937 (1)**	**0.9888 (1)**	0.8491 (3)	0.9846 (3)	0.9842 (7)	**0.9446 (1)**	**3.11 (1)**

Табл. A.2. Сравнение точности DiSSiD с аналогами (метрика R-AUC-PR)

	Метод	Временные ряды										
		SMD	**OPP**	**Daphnet**	**ECG-2 (803, 805)**	**ECG-2 (803, 806)**	**ECG-3 (803, 805, 806)**	**MITDB**	**IOPS**	**YAHOO**	**Средний VUS-ROC**	**Средний ранг**
Без учителя	**IForest**	0.0797 (13)	0.9950 (2)	0.3577 (4)	0.7279 (4)	0.7202 (3)	0.5957 (4)	0.2838 (6)	0.9031 (5)	0.8974 (2)	0.6178 (2)	4.78 (2)
	LOF	0.1511 (6)	0.036 (14)	**0.5484 (1)**	0.3374 (10)	0.5228 (10)	0.4273 (8)	0.2143 (7)	0.6050 (11)	0.8624 (6)	0.4116 (9)	8.11 (9)
	MP	0.3252 (3)	0.0358 (15)	0.3849 (3)	0.3227 (13)	0.5459 (8)	0.3529 (11)	**0.3984 (1)**	0.8594 (8)	0.8949 (3)	0.4578 (8)	7.22 (7)
	DAMP	0.0851 (12)	0.0566 (13)	0.2391 (8)	0.2479 (16)	0.2791 (13)	0.2363 (14)	0.1055 (10)	0.2114 (15)	0.1637 (17)	0.1805 (17)	13.11 (15)
	NormA	0.3639 (2)	0.0344 (16)	0.2089 (12)	0.2326 (17)	0.3317 (12)	0.1508 (16)	0.1630 (9)	0.8568 (9)	0.5172 (13)	0.3177 (13)	11.78 (13)
	PCA	0.1662 (5)	**0.9989 (1)**	0.1242 (17)	0.6773 (5)	0.6967 (4)	0.5721 (5)	0.3224 (4)	0.9627 (2)	0.8672 (5)	0.5986 (3)	5.33 (3)
	POLY	0.1507 (7)	0.9041 (4)	0.2209 (10)	0.6050 (6)	0.5871 (7)	0.5168 (6)	0.3199 (5)	0.6264 (10)	0.7677 (8)	0.5221 (5)	7.00 (6)
С частичным привлечением учителя	**AE**	0.0931 (11)	0.2065 (8)	0.2115 (11)	**0.8734 (1)**	0.6762 (5)	0.8894 (2)	0.0752 (15)	0.3944 (12)	0.8857 (4)	0.4784 (7)	7.67 (8)
	Bagel	0.0599 (15)	nan (17)	0.2319 (9)	0.3295 (12)	0.1933 (17)	0.3042 (12)	0.0861 (12)	0.2988 (13)	0.4982 (14)	0.2503 (14)	13.45 (16)
	DeepAnT	0.0509 (16)	0.0601 (12)	0.2669 (7)	0.3306 (11)	0.2373 (14)	0.2922 (13)	0.0770 (14)	0.1853 (16)	0.5644 (11)	0.2294 (15)	12.67 (14)
	IE-CAE	0.1434 (8)	0.9083 (3)	0.3287 (6)	0.5641 (7)	0.5934 (6)	0.5068 (7)	0.1785 (8)	**0.9840 (1)**	0.7431 (9)	0.5500 (4)	6.11 (5)
	LSTM-AD	0.0618 (14)	0.0648 (11)	0.1697 (14)	0.2907 (15)	0.1939 (16)	0.2347 (15)	0.0776 (13)	0.1626 (17)	0.4358 (16)	0.1880 (16)	14.56 (17)
	OceanWNN	0.1077 (10)	0.4620 (7)	0.1786 (13)	0.5558 (8)	0.2062 (15)	0.3686 (10)	0.1048 (11)	0.9096 (3)	0.5597 (12)	0.3837 (11)	9.89 (10)
	OCSVM	0.0098 (17)	0.1927 (9)	0.1324 (16)	0.3832 (9)	0.3669 (11)	0.3809 (9)	0.0450 (17)	0.8741 (7)	0.8613 (7)	0.3607 (12)	11.33 (11)
	TAnoGAN	0.1250 (9)	0.8024 (5)	0.1594 (15)	0.2955 (14)	0.5291 (9)	0.1424 (17)	0.0715 (16)	0.9085 (4)	0.4854 (15)	0.3910 (10)	11.56 (12)
	DiSSiD (L1)	0.1765 (4)	0.1149 (10)	0.4485 (2)	0.8522 (3)	0.8704 (2)	0.8600 (3)	0.3967 (2)	0.2625 (14)	0.5970 (10)	0.5087 (6)	5.56 (4)
	DiSSiD (MPdist)	**0.5342 (1)**	0.5270 (6)	0.3444 (5)	0.8729 (2)	**0.9192 (1)**	**0.9114 (1)**	0.3842 (3)	0.8880 (6)	**0.9113 (1)**	**0.6992 (1)**	**2.89 (1)**

Табл. А.3. Сравнение точности DiSSiD с аналогами (метрика VUS-ROC)

	Метод	Временные ряды										
		SMD	**OPP**	**Daphnet**	**ECG-2 (803, 805)**	**ECG-2 (803, 806)**	**ECG-3 (803, 805, 806)**	**MITDB**	**IOPS**	**YAHOO**	**Средний VUS-ROC**	**Средний ранг**
Без учителя	**IForest**	0.8662 (10)	0.9974 (2)	0.7548 (5)	0.8666 (4)	0.8784 (4)	0.8317 (4)	0.7470 (5)	0.9712 (5)	**0.9804 (1)**	0.8771 (2)	4.45 (2)
	LOF	0.9291 (5)	0.3700 (15)	**0.8410 (1)**	0.7218 (12)	0.8261 (7)	0.7611 (10)	0.6806 (7)	0.6840 (13)	0.9670 (4)	0.7534 (9)	8.22 (9)
	MP	0.9567 (3)	0.3878 (14)	0.8364 (2)	0.6556 (15)	0.9023 (3)	0.6748 (14)	0.8715 (2)	0.8702 (8)	0.9791 (3)	0.7927 (8)	7.11 (6)
	DAMP	0.8819 (9)	0.5476 (13)	0.6946 (7)	0.6679 (14)	0.7172 (16)	0.6948 (13)	0.5316 (15)	0.6704 (14)	0.6074 (15)	0.6682 (13)	12.89 (15)
	NormA	**0.9828 (1)**	0.3649 (16)	0.6006 (12)	0.5202 (17)	0.7872 (9)	0.3883 (17)	0.7565 (4)	0.7604 (10)	0.6004 (16)	0.6402 (15)	11.33 (12)
	PCA	0.9234 (6)	**0.9993 (1)**	0.4460 (17)	0.7977 (6)	0.8033 (8)	0.7813 (9)	0.6775 (8)	0.9878 (2)	0.9665 (5)	0.8203 (6)	6.89 (14)
	POLY	0.6318 (16)	0.9930 (3)	0.5389 (14)	0.7824 (8)	0.7743 (12)	0.7599 (11)	0.6131 (11)	0.7264 (11)	0.9516 (7)	0.7524 (10)	10.33 (10)
С частичным привлечением учителя	**AE**	0.8529 (11)	0.6905 (10)	0.7302 (6)	0.9644 (3)	0.8508 (6)	0.9596 (2)	0.5748 (14)	0.7963 (9)	0.9802 (2)	0.8222 (5)	7.00 (5)
	Bagel	0.8514 (12)	nan (17)	0.6129 (11)	0.7949 (7)	0.7544 (14)	0.8108 (6)	0.6261 (9)	0.5711 (16)	0.8673 (12)	0.7361 (10)	11.56 (13)
	DeepAnT	0.7729 (14)	0.5576 (11)	0.6841 (8)	0.7809 (9)	0.7803 (11)	0.7918 (7)	0.6174 (10)	0.5733 (15)	0.8767 (11)	0.7150 (11)	10.67 (11)
	IE-CAE	0.9028 (7)	0.9882 (4)	0.6781 (9)	0.7760 (10)	0.8627 (5)	0.7861 (8)	0.5990 (13)	**0.9889 (1)**	0.9477 (9)	0.8366 (4)	7.33 (7)
	LSTM-AD	0.8359 (13)	0.5478 (12)	0.6575 (10)	0.7247 (11)	0.7161 (17)	0.7132 (12)	0.6021 (12)	0.5121 (17)	0.8171 (13)	0.6807 (12)	13.0 (17)
	OceanWNN	0.8916 (8)	0.8845 (7)	0.5634 (13)	0.8504 (5)	0.7857 (10)	0.8188 (5)	0.7068 (6)	0.9714 (4)	0.8949 (10)	0.8186 (7)	7.56 (8)
	OCSVM	0.2109 (17)	0.6984 (9)	0.4486 (16)	0.6993 (13)	0.7255 (15)	0.6501 (15)	0.3233 (17)	0.9137 (7)	0.9622 (6)	0.6258 (16)	12.78 (14)
	TAnoGAN	0.6437 (15)	0.9417 (5)	0.4964 (15)	0.5972 (16)	0.7738 (13)	0.3889 (16)	0.4070 (16)	0.9700 (6)	0.7438 (14)	0.6625 (14)	12.89 (15)
	DiSSiD (L1)	0.9474 (4)	0.7016 (8)	0.8331 (3)	0.9655 (2)	**0.9861 (1)**	0.9564 (3)	**0.9153 (1)**	0.7058 (12)	0.5511 (17)	0.8402 (3)	5.67 (3)
	DiSSiD (MPdist)	0.9654 (2)	0.9199 (6)	0.8267 (4)	**0.9731 (1)**	0.9811 (2)	**0.9767 (1)**	0.8350 (3)	0.9725 (3)	0.9501 (8)	**0.9334 (1)**	**3.33 (1)**

Табл. А.4. Сравнение точности DiSSiD с аналогами (метрика VUS-PR)

	Метод	Временные ряды										
		SMD	**OPP**	**Daphnet**	**ECG-2 (803, 805)**	**ECG-2 (803, 806)**	**ECG-3 (803, 805, 806)**	**MITDB**	**IOPS**	**YAHOO**	**Средний VUS-PR**	**Средний ранг**
Без учителя	**IForest**	0.0673 (13)	0.9731 (2)	0.3255 (5)	0.6559 (4)	0.5893 (3)	0.5519 (4)	0.2618 (6)	0.8595 (5)	**0.7360 (1)**	0.5578 (2)	4.78 (2)
	LOF	0.1492 (5)	0.0395 (14)	**0.5055 (1)**	0.3195 (12)	0.4048 (10)	0.3583 (9)	0.1864 (7)	0.4887 (11)	0.6789 (8)	0.3479 (11)	8.56 (9)
	MP	0.2762 (3)	0.0392 (15)	0.3552 (3)	0.2910 (14)	0.4582 (9)	0.2981 (12)	0.3608 (2)	0.7471 (8)	0.7353 (2)	0.3957 (8)	7.56 (7)
	DAMP	0.0879 (11)	0.0581 (13)	0.2366 (8)	0.2464 (16)	0.2699 (13)	0.2347 (14)	0.1034 (11)	0.2234 (15)	0.1583 (17)	0.1799 (17)	13.11 (15)
	NormA	0.3412 (2)	0.0361 (16)	0.2058 (12)	0.2263 (17)	0.3086 (11)	0.1470 (16)	0.1565 (9)	0.7227 (9)	0.5276 (13)	0.2969 (13)	11.67 (12)
	PCA	0.1443 (6)	**0.9946 (1)**	0.1219 (17)	0.6115 (5)	0.5685 (4)	0.5214 (5)	0.3013 (5)	**0.9253 (1)**	0.6806 (7)	0.5410 (3)	5.67 (4)
	POLY	0.1243 (8)	0.9063 (3)	0.2213 (10)	0.5582 (6)	0.5113 (7)	0.4797 (6)	0.3070 (4)	0.5567 (10)	0.6975 (6)	0.4847 (5)	6.67 (6)
С частичным привлечением учителя	**AE**	0.0767 (12)	0.1979 (8)	0.2160 (11)	0.7758 (2)	0.5589 (5)	0.7651 (2)	0.0759 (15)	0.3720 (12)	0.7238 (4)	0.4181 (7)	7.89 (8)
	Bagel	0.0559 (15)	nan (17)	0.2269 (9)	0.3302 (11)	0.1878 (17)	0.2988 (11)	0.0833 (12)	0.2678 (13)	0.4871 (14)	0.2422 (14)	13.22 (16)
	DeepAnT	0.0522 (16)	0.0605 (12)	0.2573 (7)	0.3350 (10)	0.2346 (14)	0.2906 (13)	0.0795 (14)	0.1834 (16)	0.5659 (12)	0.2288 (15)	12.67 (14)
	IE-CAE	0.1297 (7)	0.9002 (4)	0.3079 (6)	0.5234 (8)	0.5397 (6)	0.4739 (7)	0.1713 (8)	0.9163 (2)	0.7050 (5)	0.5186 (4)	5.89 (5)
	LSTM-AD	0.0653 (14)	0.0650 (11)	0.1711 (14)	0.2897 (15)	0.1934 (16)	0.2330 (15)	0.0799 (13)	0.1595 (17)	0.4478 (16)	0.1894 (16)	14.56 (17)
	OceanWNN	0.1075 (9)	0.4678 (7)	0.1812 (13)	0.5544 (7)	0.2003 (15)	0.3596 (8)	0.1058 (10)	0.9085 (4)	0.6126 (10)	0.3886 (9)	9.22 (10)
	OCSVM	0.0119 (17)	0.1795 (9)	0.1388 (16)	0.3548 (9)	0.3069 (12)	0.3315 (10)	0.0474 (17)	0.7533 (7)	0.6639 (9)	0.3098 (12)	11.78 (13)
	TAnoGAN	0.0965 (10)	0.8090 (5)	0.1609 (15)	0.3002 (13)	0.4634 (8)	0.1430 (17)	0.0714 (16)	0.9130 (3)	0.4591 (15)	0.3796 (10)	11.33 (11)
	DiSSiD (L1)	0.1543 (4)	0.1222 (10)	0.4124 (2)	0.7477 (3)	**0.8008 (1)**	0.7505 (3)	**0.3718 (1)**	0.2464 (14)	0.5961 (11)	0.4669 (6)	5.45 (3)
	DiSSiD (MPdist)	**0.4889 (1)**	0.5340 (6)	0.3332 (4)	**0.7801 (1)**	0.7927 (2)	**0.8124 (1)**	0.3544 (3)	0.7922 (6)	0.7306 (3)	**0.6243 (1)**	**3.00 (1)**