

федеральное государственное бюджетное образовательное
учреждение высшего образования
«Казанский национальный исследовательский
технический университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)

Компьютерных технологий и защиты информации

(наименование института)

Систем информационной безопасности

(наименование кафедры)

09.04.01 Информатика и вычислительная техника

(шифр и наименование направления подготовки)

К защите допустить

Зав. каф. СИБ

/ И. В. Аникин /

«_____» _____ 2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
на тему «Идентификация аномальных траекторий в неопределенных
пространственно-временных данных»

ОБУЧАЮЩИЙСЯ

А. Р. Марданова

(и^{нициалы, фамилия})

(личная подпись)

РУКОВОДИТЕЛЬ

д.т.н., проф. И. В. Аникин

(учёная степень, звание,

(личная подпись)

и^{нициалы, фамилия})

Казань, 2020

federal state budget educational institution of higher education
«Kazan National Research Technical University
named after A. N. Tupolev-KAI»
(KNRTU-KAI)

Computer Technology and Information Security
(name of the institute)
Information Security Systems
(name of the department)

09.04.01 Computer Science and Engineering
(code and name of the training area)

Allow
Head of the Department ISS

/ I. V. Anikin /

«_____» _____ 2020

FINAL QUALIFYING WORK
on the topic «Identification of Trajectory Anomalies in Uncertain
Spatiotemporal Data»

STUDENT	<u>A.R. Mardanova</u> (initials, surname)	_____	(personal signature)
SUPERVISOR	<u>Dr. Tech. Sc., Prof. I.V. Anikin</u> (academic degree, title, initials, surname)	_____	(personal signature)

Kazan, 2020

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. А.Н. ТУПОЛЕВА-КАИ

Институт Компьютерных технологий и защиты информации
Кафедра Систем информационной безопасности
Направление 09.04.01 «Информатика и вычислительная техника»

УТВЕРЖДАЮ:
Зав. кафедрой СИБ

_____ / И.В. Аникин /
«____» _____ 2020 г.

ЗАДАНИЕ
на выпускную квалификационную работу

Мардановой Айгуль Рустамовны
(фамилия, имя, отчество)

1. Тема выпускной квалификационной работы: «Идентификация аномальных траекторий в неопределенных пространственно-временных данных», утверждена приказом по университету от «1» апреля 2020 г. №1353-С.
2. Срок сдачи студентом законченной выпускной квалификационной работы: 12.08.2020 г.
3. Исходные данные к работе: литературные и Интернет-источники по методам и алгоритмам кластеризации траекторий, подходам к обнаружению аномалий, валидация .
4. Содержание работы (перечень подлежащих разработке вопросов и исходные данные к ним)
 - 4.1. Обоснование актуальности проблемы и определение основных задач.
 - 4.2. Введение основных понятий и концепций, используемых далее в работе.
 - 4.3. Анализ предметной области и существующих решений задачи обнаружения аномальных траекторий в неопределенных пространственно-временных данных.
 - 4.4. Разработка методики обработки исходных данных.
 - 4.5. Разработка фреймворка для обработки входных данных, кластеризации траекторий и определения аномалий.
 - 4.6. Тестирование и обсуждение полученных результатов. Подведение итогов.
5. Перечень графического материала (с указанием обязательных чертежей): не предусмотрено.
6. Консультанты по выпускной квалификационной работе (с указанием относящихся к ним разделов)

Раздел	Консультант (фамилия, инициалы)	Подпись, дата	
		Задание выдал	Задание принял
Основной раздел	Аникин И.В.		

7. Дата выдачи задания «_____» 2020 г.

Научный руководитель _____ / Аникин И.В. /
(подпись) (фамилия, инициалы)

Задание принял к исполнению _____ / Марданова А.Р. /
(подпись студента) (фамилия, инициалы)

КАЛЕНДАРНЫЙ ПЛАН

№ этапа	Наименование этапов выпускной квалификационной работы	Сроки выполнения этапов работы	Примечания
	Постановка задач и определение мотивации	15.04.2020	
1.	Анализ предметной области и существующих решений задачи обнаружения аномальных траекторий в неопределенных пространственно-временных данных.	30.04.2020	
2.	Реализация обработки входных данных	05.06.2020	
3.	Разработка фреймворка для обнаружения аномальных траекторий в неопределенных пространственно-временных данных	10.07.2020	
4.	Тестирование и анализ полученных результатов	30.07.2020	
5.	Написание и оформление выпускной квалификационной работы	08.08.2020	

Магистрант _____ / Марданова А.Р. /

Научный руководитель _____ / Аникин И.В. /

Оглавление

Список сокращений	3
Перечень обозначений	4
1 ВВЕДЕНИЕ	6
1.1 Постановка задачи	7
1.2 Структура работы	8
2 ОСНОВНЫЕ ПОНЯТИЯ	10
2.1 Источники входных данных	10
2.2 Определение траектории	10
2.2.1 Определение аномальной траектории	11
2.2.2 Классификация аномальных траекторий	11
2.3 Основные сложности	13
3 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	15
3.1 Методы обнаружения аномалий	15
3.2 Классификация методов кластеризации	20
3.3 Метрики измерения близости и дальности	25
3.4 Выводы	29
4 ПРЕДЛОЖЕННЫЙ ПОДХОД	31
4.1 Концептуальная модель фреймворка	31
4.2 Архитектура фреймворка	32
4.3 Кластеризация	34
4.3.1 Агломеративная иерархическая кластеризация	34
4.3.2 Моделирование кластеров	35
4.4 Сравнение траекторий	36
4.4.1 LCSS метрика для измерения схожести траекторий	36
4.4.2 Адаптивность параметров LCSS алгоритма	36
4.5 Проверка достоверности кластеров	38
4.6 Аппроксимация траекторий	40
4.6.1 Регрессионный анализ	41

4.6.2 Полиномиальная регрессия	41
5 РЕАЛИЗАЦИЯ ФРЕЙМВОРКА	43
5.1 Используемые технологии	43
5.2 Работа со входными данными	44
5.2.1 Описание входных данных	44
5.2.2 Структура файла входных данных	45
5.2.3 Обработка входных данных	46
5.2.4 Аппроксимация траекторий с использованием Полиномиальной Регрессии	48
5.2.5 Выбор ключевых точек в аппроксимированных траекториях	49
5.3 Анализ траекторий	54
5.3.1 Вычисление метрики схожести траекторий	54
5.3.2 Кластеризация	55
5.3.3 Моделирование кластеров	57
6 Результаты	58
6.1 Оценка точности	58
6.1.1 Результаты аппроксимации траекторий	58
6.1.2 Результаты кластеризации траекторий	62
7 ЗАКЛЮЧЕНИЕ	65
Литература	67
Список иллюстраций	72
Список таблиц	73
Список алгоритмов	74
Список листингов	74
ПРИЛОЖЕНИЕ	I
A. Алгоритм парсинга входных траекторий	I
B. Инициализация Полиномиальной Регрессии	V
C. Агломеративная Иерархическая Кластеризация	VI

Список сокращений

ГИС	Географические Информационные Системы
ИТС	Интеллектуальные Транспортные Системы
TC	Транспортное Средство
ПДД	Правила Дорожного Движения
ДТП	Дорожно-Транспортное Происшествие
GPS	Global Positioning System
ID	Identifier
SVM	Support Vector Machine
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DTW	Dynamic Time Warping
LCSS	Longest Common SubSequence
HMM	Hidden Markov Model
DI	Dunn's Index
AWT	Abstract Window Toolkit
ИПП	Интерфейс Прикладного Программирования
API	Application Programming Interface
ГПИ	Графический Пользовательский Интерфейс
GUI	Graphical User Interface
FARS	Fatality Analysis Reporting System
SSE	Sum of Squares due to Error
TSS	Total Sum of Squares

Перечень обозначений

τ, T	Траектория
TP	Точка Траектории
t	Временной параметр
x	Координата X
y	Координата Y
v_{avg}	Средняя скорость траектории
d_{ij}	Евклидово расстояние между траекториями T_i, T_j
$dist(a_i^k, b_j^k)$	Евклидово расстояние между точками a_i^k, b_j^k
R^2	Коэффициент детерминации R-квадрат

АННОТАЦИЯ

Работа состоит из ?? страниц, содержит 18 рисунков, 9 таблиц, источника, приложений.

1 ВВЕДЕНИЕ

В наши дни обработка и анализ пространственно-временных данных приобретает все большую популярность и находит все большее применение в приложениях, основанных на использовании Географических Информационных Систем (ГИС). Последние исследования в области ГИС и, в частности, технологий и инфраструктуры для ГИС способствовали развитию интеллектуальных городов. И Интеллектуальные Транспортные Системы (ИТС), подразумевающие анализ городского транспортного движения, являются одной из самых перспективных областей [1].

Интеллектуальное слежение в умных городах получило большое развитие за последнее десятилетие [4]. В последнее время растущее число дорог и общественных мест оснащаются видеокамерами для мониторинга, увеличивается количество общедоступных видеоданных для анализа [2]. Задача автоматического анализа данных, собранных в ходе видеонаблюдения за дорожным движением, привлекает все большее внимание научного сообщества [3].

В настоящее время существует множество задач и применений анализа городского транспортного движения, однако, согласно [4], отслеживание поведения транспортных средств (ТС) с помощью обработки видео-изображений является одним из самых многообещающих подходов. Одним из основных исследовательских подходов в области анализа городского трафика, предполагающих работу с данными с камер видеонаблюдения, является извлечение из пространственно-временных данных паттернов частых траекторий движения. Извлеченные траектории могут впоследствии быть применены для автоматического визуального наблюдения, регулирования транспортного движения, обнаружения подозрительных активностей и др. [5][6].

Еще одной важной подкатегорией в области анализа транспортного трафика является обнаружение аномалий в потоке данных [4]. Данная задача является очень актуальной и находит применение во многих приложениях для умных городов. Аномалия традиционно характеризуется как событие, экземпляр данных, который значительно отличается от большинства экземпляров в наборе данных и отклоняется от нормы [7]. В сфере видеонаблюдения за ТС аномальной деятельностью обычно называют события, которые нарушают общие закономерности, как правило, правила дорожного движения (ПДД) [3]. Такие необычные паттерны движения ТС, которые не соответствуют ожидаемому поведению, несут важную информацию, поскольку могут свидетельствовать об аномальных транспортных потоках в сети автомобильных дорог [4]. Например, случаи дорожно-

транспортного происшествия (ДТП) или затора в транспортном движении ведут к резкому изменению транспортных потоков. Это в свою очередь провоцирует появление траекторий движения, отклоняющихся от нормальных паттернов движения. Следовательно, распознавание аномалий может быть полезно для своевременного обнаружения случаев ДТП и предпринятия должных мер. Однако, в наш век информационного перенасыщения, когда огромные массивы данных доступны для обработки и анализа, ручная и обработка становится невыполнимой задачей, неавтоматизированные решения становятся невозможными и неподходящими из-за высокой степени сложности и времязатратности. Поэтому исследования научных сообществ направлены на разработку автоматических и полуавтоматических интеллектуальных методов для решения этих задач с максимально возможной минимизацией необходимости вовлечения человека-оператора [8].

Как отмечается в последних исследованиях в области анализа транспортного трафика, во многих приложениях, включая ИТС, чрезвычайно важно учитывать неопределенность данных. Причины неопределенности данных разнообразны. Например, неопределенность данных может быть в результате неточности измерений или неточности наблюдений. В случае получения данных о траектории с камер видеонаблюдения неопределенность данных может быть вызвана ограничениями используемых устройств или потерянным местоположением [9].

1.1 Постановка задачи

Как было отмечено выше, в наши дни анализ пространственно-временных данных играет важную роль в ежедневных процессах, в повседневной жизни, и процесс извлечения полезной информации из пространственно-временных данных является одной из ключевых задач и проблем при анализе данных трафика. Поскольку пространственно-временные данные о траекториях ТС являются многомерными и пространственно-временны характеристики траектории зависят между собой, традиционные подходы к анализу данных, предлагаемые для статических, единичных и независимых данных, становятся неэффективны и неуместны [10].

Основной целью работы в этом тезисе является разработка подхода к обработке неопределенных пространственно-временных данных о траекториях для решения задач определения частых траекторий и обнаружения аномалий, а также проведение сравнительного анализа предложенного решения. В качестве основы для проведения оценочных и контрольных тестов, направленных на проверку точности и эффективности предложенного подхода, будет разработан фреймворк (платформа) для извлечения часто встречающихся траекторий и обнаружения наомальных траекторий в трехмерных пространственно-временных данных траекторий, полученных с камер видеонаблюдения. Видео с камер наблюдения будет обрабатываться во внешней системе слежения, которая

извлекает траектории ТС и преобразует их в векторы, состоящие из точек слежения (точек траектории). Внедренный метод должен быть оценен с точки зрения точности и производительности, а также способности улучшить, повысить точность результатов в контексте таких особенностей входных данных, как неопределенность.

Для решения вышеупомянутых проблем следующие задачи должны быть выполнены:

- Провести анализ предметной области и существующих подходов и выбрать методы для решения задач определения частых траекторий движения и обнаружения аномалий;
- Исследовать возможность улучшения существующего метода и предложить метод для повышения точности результатов для выбранного метода в контексте использования данных с камер видеонаблюдения;
- Реализовать фреймворк с использованием выбранных алгоритмов для тестирования предложенного похода и проведения сравнительного анализа полученных результатов;
- Провести тестирование эффективности и точности реализованного подхода.

Эта работа будет сфокусирована на следующих типах аномальных траекторий:

- Аномальные траектории с аномальной пространственной информацией. Эта категория покрывает траектории с аномальным пространственным поведением, такие как запрещенные на перекрестках развороты на 180°, пересечение двойной сплошной линии, движение в обратном направлении.
- Аномальные траектории с аномальной пространственно-временной информацией. Этот тип аномалий относится к случаям, когда пространственная информация сама по себе может быть расценена как нормальная, но вместе с информацией о времени представляет собой аномальное поведение, например: движение с чрезвычайно высокой или низкой скоростью, неожиданные остановки.

1.2 Структура работы

Работа структурирована следующим образом. Весь отчет состоит из 7 частей. Первая часть представляет собой введение, где описана актуальность работы, обозначены цели и задачи, структура отчета. Во 2 главе вводятся основные понятия и необходимая терминология, используемая далее в работе. Глава 3 посвящена результатам анализа литературы в предметной области и обсуждению современного состояния поставленной проблемы. В главе 4 приведено подробное описание предлагаемого подхода к решению поставленной

1.2. СТРУКТУРА РАБОТЫ

задачи на концептуальном уровне, с алгоритмическим описанием используемых методов и архитектурных особенностей. Глава 5 описывает детали реализации фреймворка, формат структуры входных данных и процесс обработки входных данных. В главе 6 представлена подробная информация о подготовке экспериментов и приведены результаты проведения экспериментов, направленных на тестирование точности и эффективности использованных методов. Глава 7 представляет собой заключение и содержит краткое изложение полученных результатов и обсуждение дальнейших перспектив развития. В Приложении приведен исходный код для ключевых алгоритмов из реализованного фреймворка, подкрепленных описанием и комментариями, представленными в главе 5.

2 ОСНОВНЫЕ ПОНЯТИЯ

Эта глава предназначена для предоставления справочной информации, введения полезных определений и основных концепций подходов, используемых в следующих главах. В этой главе будут рассмотрены источники входных данных и связанные с ними проблемы.

2.1 Источники входных данных

Задачи определения частых траекторий и обнаружения аномалий могут быть реализованы применимо к различным источникам данных, например: устройства GPS (Global Positioning System, глобальная система позиционирования) и сенсорным сетям, когда данные о траектории собираются датчиками на движущихся объектах, которые периодически передают информацию о местоположении движущегося объекта во времени, или камеры видеонаблюдения. Данная работа будет сфокусирована на работе с последним типом источников входных данных.

Видеоданные с камер слежения являются необработанными данными и не используются непосредственно в качестве входных данных для разрабатываемой системы. Обработка необработанного видео выполняется в автономной системе слежения. Система слежения берет исходное видео с камер видеонаблюдения и обрабатывает его, выполняя обнаружение объектов и преобразовывая траекторию в ряд точек слежения на изображениях. Точки слежения, содержащие такую информацию, как идентификатор (ID) ТС, метку времени, пространственные координаты, используются в качестве входных данных.

2.2 Определение траектории

Траектории могут быть представлены как многомерные последовательности, содержащие упорядоченный во временном отношении список местоположений вместе с любой дополнительной информацией [7]. Таким образом, поскольку траектория, обозначенная как τ или T , представляет собой последовательные местоположения движущегося целевого объекта во времени, в случае получения данных наблюдения с одной камеры ее можно определить как:

$$\tau = T = (x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n) \quad (2.2.1)$$

где пары (x_i, y_i) обозначают положение целевого объекта на изображении в момент времени t_i [5]. В соответствии с этим траектории могут быть представлены в виде последовательности трехмерных точек, где 2D-объект предназначен для геометрических координат, а в третьем измерении хранится время [11].

Как правило, данные о траекториях являются сырьими, необработанными и содержат только минимальную информацию, такую как положение в пространстве и время, а также ID объекта отслеживания. Указанная информация может быть легко дополнена такой подробной информацией, как скорость, ускорение и направление, поскольку они могут быть извлечены из исходных данных о траектории [12].

2.2.1 Определение аномальной траектории

Двадцатичетырехчасовые записывающие камеры видеонаблюдения производят огромные объемы данных о движущихся объектах, и это увеличивает вероятность того, что наряду с объектами, имеющими нормальное поведение, некоторые из движущихся объектов будут демонстрировать ненормальное поведение. Такое исключительное поведение можно также назвать исключением, аномалией, отклонением (от нормы) [13][14]. Несмотря на то что не существует стандартизированного определения понятия аномалии, в статистике можно найти следующую расшифровку данного понятия [15]:

“An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs”.

Траекторные аномалии (или аномальные траектории) могут быть описаны как паттерны потока транспортного трафика, значительно отклоняющиеся от некоторого нормального шаблона поведения или, другими словами, несовместимые с остальными моделями поведения трафика. Предполагается, что аномальные траектории имеют большую локальную или глобальную разницу с большинством траекторий согласно выбранной метрике подобия [6].

Процесс обнаружения аномалий направлен на выявление необычных паттернов, которые кардинально отличаются от большинства экземпляров в исходных данных, для дальнейшей их обработки соответствующим образом [13]. Также необходимо отметить, что отношение аномальных паттернов траекторий к нормальным моделям активности должно быть относительно небольшим, чтобы можно было отличить аномалии от доминирующих нормальных паттернов.

2.2.2 Классификация аномальных траекторий

Согласно литературе, аномальные траектории могут быть классифицированы следующим образом [14][16][17]:

- *Точечная аномалия, Point anomaly* – представляет собой наипростейший тип аномалий. Соответствует отдельному экземпляру данных, который расценивается как аномальный по отношению к остальному массиву данных, поскольку он значительно отличается от всех других экземпляров в наборе данных. Например, неподвижная машина на оживленной дороге.
- *Контекстная аномалия, Contextual anomaly* – экземпляр данных, который является аномальным в определенном контексте, но может быть нормальным в другом случае. Контекстная аномалия может также быть представлена как точечная в своем локальном окружении аномалия. Контекстуальные аномалии также называются условными аномалиями и представляют собой наиболее распространенную группу категорий, применимых к пространственно-временным данным. Например, траектории могут быть классифицированы на основе пространственных данных (координат) в пределах времени. Примерами контекстных аномалий могут быть траектории движения ТС с гораздо более высокой скоростью по сравнению с другими в том же транспортном потоке или движения ТС в противоположном направлении.
- *Собирательные аномалии, Collective anomalies* - множество экземпляров данных, которые в совокупности как группа представляют собой аномалию по отношению к остальной части данных, в то время как каждый экземпляр данных в отдельности не обязательно является аномальным. Данное определение может быть упрощено до следующей формы: набор соседних точечных аномалий или контекстных аномалий. Коллективные аномалии могут быть применены только к наборам данных, в которых существует зависимость между экземплярами данных

Другим способом систематизации аномальных траекторий может быть разделение их на следующие категории в соответствии со свойствами, которые использовались для выполнения классификации:

- *Пространственные аномальные траектории, Spatial trajectory anomaly* – классификация учитывает только пространственную информацию о траекториях движущихся объектов, например координаты местоположения. Примерами пространственных аномалий могут быть незаконные развороты, пересечение двойной сплошной линии или движение в противоположном направлении.
- *Временные аномальные траектории, Temporal trajectory anomaly* – аномалии, обнаруженные путем анализа только временных характеристик траекторий, таких как продолжительность, время перемещения. Например, траектория со значитель-

но большей продолжительностью или траектория, появляющаяся в аномальное время.

- *Пространственно-временные аномальные траектории, Spatiotemporal trajectory anomaly* - могут быть обнаружены путем анализа пространственной и временной информации в совокупности. Примерами пространственно-временных аномалий могут быть траектории ТС, движущихся со значительно высокой скоростью по сравнению с большинством траекторий. Также такие аномалии могут быть обнаружены в случае транспортных систем с реверсивным движением. Так как для таких полос движения разрешено изменение направления согласно некоторому известному или изученному графику, классификатор может анализировать направление траектории вместе с информацией о времени.

В соответствии со второй классификацией эта работа будет сфокусирована на определении аномальных траектории первого и третьего типов (пространственных и пространственно-временных аномальных траекторий).

2.3 Основные сложности

Поскольку пространственно-временные данные отличаются от других типов данных во многих аспектах, сложности связаны с используемым этого типа данных. Уникальным качеством пространственно-временных данных является то, что экземпляры данных не являются независимыми и одинаково распределенными, как это обычно предполагается во многих существующих подходах для интеллектуального анализа данных. Напротив, экземпляры пространственно-временных данных, связанные с результатами наблюдения, проведенными в близлежащих точках и близкое время, структурно коррелируют друг с другом в контексте пространства и времени, и важно учитывать наличие зависимостей между значениями в этих измерениях. Следовательно, многие из существующих подходов для интеллектуального анализа данных не применимы к пространственно-временным данным, поскольку игнорирование вышеупомянутых характеристик может привести к низкой точности результатов. Это ведет к необходимости изучения и использования различных методов обработки таких данных для сохранения всех связей между информационными доменами [7].

Неопределенность данных

Также следует отметить, что выбранный тип источника данных приводит к трудностям при обработке. Поскольку данные о траектории собираются с видеокамер, первая проблема заключается в неопределенности местоположения в результате ограничений точности измерений используемых камер, разрешения и качества полученных изображений, дрожания кадра [2]. Кроме того, камеры слежения размещаются в определенных

фиксированных местах на перекрестках, из-за чего одной из особенностей используемых данных являются положение движущегося объекта и перспектива, которые могут вызвать проблемы при работе с входными видеоданными [16]. Угол обзора камеры относительно земной горизонтальной поверхности и расстояние между отслеживаемым объектом и камерой могут влиять на качество обработки, понижая точность обнаружения и отслеживания объектов: чем меньше угол, тем существеннее проблема определения центра объекта [2][4]. Отслеживаемые объекты могут въезжать и выезжать в поле / из поля зрения камеры, но при этом оставаться отслеживаемыми, поскольку они частично видны. Это может привести к изменению траектории на границах поля зрения камеры: смещение траекторий ТС в зависимости от расположения объекта относительно камеры [2]. Качество извлечения и последующего анализа траекторий также зависит от входных данных о траекториях, включая такие критерии, как качество используемых камер, качество системы слежения, которая преобразует видеоданные в список траекторий, состоящий из точек слежения.

Более того, в текущей дипломной работе входные данные содержат траектории, извлеченные из видеокамер без фильтрации и предварительного анализа, поэтому:

1. входные данные содержат экземпляры и нормальных, и аномальных траекторий;
2. входные данные содержат траектории без указания меток классов.

Вышеупомянутые ограничения ведут к необходимости использовать неконтролируемые безнадзорные методы для автоматического извлечения паттернов нормальных и аномальных траекторий движения из непомеченных, неклассифицированных данных [18].

3 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В этой главе представлен анализ современного состояния предметной области и обзор существующих подходов, обсуждаются преимущества и недостатки существующих подходов. Каждый раздел завершается краткой резюмирующей секцией, в которой указывается выбранный подход и приводится короткая аргументация.

3.1 Методы обнаружения аномалий

Согласно [14], задача обнаружения аномальных событий и поведений, которая является основный фокусом этой работы, была предметом интереса и исследований научного сообщества начиная с XIX вв. В настоящее время существует огромное разнообразие различных методов для решения задачи обнаружения аномалий и отклонений применимых к данным видеографика, и эти подходы могут быть классифицированы различными способами.

Например, методы интеллектуального анализа данных в целом и методы обнаружения аномалий, а также алгоритмы кластеризации, которые в частности являются одним из способов решения задачи идентификации аномалий, могут быть классифицированы как контролируемые, полууправляемые и неконтролируемые на основании способа маркировки входных данных] [14] [17]:

1. *Контролируемые, Supervised.* Входные данные, используемые для обучения модели, содержат метки как для нормальных, так и для аномальных экземпляров данных. В результате алгоритм может строить модели как для нормальных, так и для аномальных классов;
2. *Полууправляемые, Semi-Supervised [14] or Weakly-Supervised [5].* Входной набор обучающих данных содержит метки классов только для нормальных экземпляров данных. Такие методы находят более широкое применение, нежели контролируемые подходы, поскольку аномальные экземпляры данных, как правило, непредсказуемы и случайны, и трудно заранее привести примеры, охватывающие всевозможные аномальные события;
3. *Неконтролируемые, Unsupervised.* Не требуют, чтобы входные данные были помечены ни для нормальных, ни для аномальных данных. Такие алгоритмы основаны на предположении, что нормальные экземпляры данных встречаются в наборе те-

стовых данных значительно чаще, нежели аномальные. И, следовательно, данные методы неприменимы в случаях, когда это предположение нарушается.

С другой стороны, согласно опросам, проведенным Чандолой в [14], Кумараном в [16] и Маликом в [17], методы обнаружения аномалий могут классифицироваться на основании базовой используемой методики, например: методы, основанные на классификации, на методе ближайших соседей, на кластеризации, статистике и т.д. Ниже приводится краткий обзор каждой из упомянутых групп.

Основанные на кластеризации

Основная концепция этих методов заключается в использовании классификатора, который сначала учится различать неискаженные и искаженные значения, а затем классифицирует каждый входной экземпляр [19]. Такие методы состоят из этапов обучения и тестирования. Фаза обучения предполагает изучение модели классификатора из набора обучающих данных, содержащего помеченные экземпляры данных. Изученный классификатор затем используется для классификации входной траектории как нормальной или аномальной путем присвоения метки класса на этапе тестирования.

В зависимости от того, как маркируются экземпляры данных тестирования, все методы обнаружения аномалий на основе классификации могут быть одноклассовыми или многоклассовыми. Первый тип предполагает, что все экземпляры обучающих данных являются нормальными и помечены как один класс. Во время фазы обучения модель изучает дискриминационную границу вокруг нормальных экземпляров, и траектория, которая не соответствует описанию изученного нормального класса, считается аномальной. Одноклассовые Support Vector Machines (SVMs) - это наиболее часто используемый подход, основанный на классификации, который применим к задаче обнаружения аномальной траектории, как это было предложено Piciarelli *et al.* [20] [21]. Однако этот подход требует, чтобы векторы траектории были одинаковой длины. Поскольку необработанные данные о траекториях обычно содержат различное количество точек траектории из-за разной скорости движущихся объектов, необходимо предварительно обработать необработанные траектории, чтобы нормализовать их и привести к векторам одинаковой длины [21]. Более того, SVM занимают много времени и памяти при работе с огромными объемами многомерных данных [22].

Вторая категория предполагает изучение нескольких классов на этапе обучения, а затем использование классификатора для проверки входной траектории на соответствие каждому изученному классу. В литературе приводятся различные описания фазы обучения и меток данных обучения. Согласно [14], данные обучения содержат только нормальные экземпляры данных с соответствующими метками нормальных классов, а во время фазы обучения модель изучает множество дискриминационных границ во-

3.1. МЕТОДЫ ОБНАРУЖЕНИЯ АНОМАЛИЙ

круг каждого класса нормальных экземпляров. Траектория, которая не соответствует ни одному из изученных нормальных описаний классов, считается аномальной. Другими словами, аномальная траектория не будет принята ни одним из классификаторов. В [16] предполагается, что модель изучается с использованием обучающих данных, содержащих метки для нормальных и аномальных классов. Следовательно, классификатор может классифицировать входную траекторию как принадлежащую нормальному или аномальному классу.

Преимущество двухэтапных алгоритмов, основанных на классификации, заключается в быстрой фазе тестирования благодаря предварительно рассчитанной модели классификатора, используемой для классификации каждого входного экземпляра. Также такие алгоритмы могут хорошо работать в случаях, когда аномальные экземпляры данных образуют класс или кластер [19]. Однако для этапа обучения требуются четко помеченные данные обучения, которые часто недоступны.

Основанные на методе ближайших соседей [14] или Основанные на близости / плотности распределения [16][19]

Подходы на основе близости определяют, является ли экземпляр данных нормальным или аномальным, основываясь на том, насколько близко или далеко он расположен по отношению к соседям [16]. Подходы, основанные на ближайших соседях и плотности, базируются на предположении, что «нормальные экземпляры данных имеют плотную окрестность, в то время как аномальные экземпляры данных встречаются далеко от их ближайших соседей» [14].

Чтобы иметь возможность сравнивать окружающую плотность для рассматриваемого экземпляра с плотностью вокруг его локальных соседей, необходимо указать меру расстояния (различия) или сходства между двумя экземплярами данных [19]. С помощью метода вычисления оценки аномалии методы могут быть сгруппированы в две категории: 1) оценка аномалии рассчитывается как расстояние экземпляра данных до его k -го ближайшего соседа и 2) для вычисления относительной оценки аномалии вычисляется плотность каждого экземпляра данных [14].

У данных подходов есть несколько недостатков. Прежде всего, по сравнению с методами обнаружения аномалий, основанных на классификации, вычислительная сложность этапа тестирования значительно выше, поскольку ближайшие соседи вычисляются путем вычисления расстояния для каждого экземпляра тестовых данных со всеми экземплярами из данных тестирования или обучения. В случае многомерных данных траекторий задача вычисления расстояния становится еще более сложной. Кроме того, точность маркировки уменьшается, когда основное предположение нарушается: когда нормальные экземпляры имеют разреженную окрестность или аномальные экземпляры имеют плотную [14].

Основанные на кластеризации

Кластеризация - это эффективный подход, направленный на группирование экземпляров данных в разные классы, называемые кластерами, на основе их сходства таким образом, что объекты в одном кластере похожи друг на друга и не похожи на объекты в других кластерах [10][22]. Кластеризация пространственно-временных данных предполагает группирование объектов на основании их пространственного и временного сходства. Чтобы сравнить экземпляры данных перед их объединением в кластеры, необходимо измерить сходство или расстояние между ними.

Существует три типа методов обнаружения аномалий на основе кластеризации со следующими допущениями: 1) нормальные экземпляры данных относятся к кластеру, в то время как аномальные экземпляры данных не связаны ни с одним кластером, 2) нормальные экземпляры данных находятся близко к центру кластера, а аномальные экземпляры лежат далеко от ближайшего центра кластера, и 3) нормальные экземпляры данных лежат в больших и плотных кластерах, тогда как аномалии связаны с разреженными кластерами или кластерами с небольшим числом элементов [14][16]. Методы первого типа могут быть реализованы с использованием одного из методов кластеризации, которые не требуют, чтобы каждый экземпляр данных принадлежал к какому-либо кластеру, например, DBSCAN [23]. Алгоритмы из второй группы состоят из двух этапов: 1) кластеризация данных и 2) вычисление оценки аномалий для каждого экземпляра данных. Методы последнего типа требуют порогового значения для размера кардинальности и/или плотности кластера, который должен быть определен, чтобы решить, относится ли кластер к нормальным или аномальным данным.

Необходимость вычисления расстояния между траекториями в некоторых подходах на основе кластеризации делает их похожими на подходы, основанные на методе ближайших соседей. Как указано в [14], основное различие этих методов заключается в том, как они обрабатывают экземпляры данных: в методах на основе кластеризации каждый экземпляр оценивается относительно соответствующего кластера, в то время как в методах на основе ближайших соседей каждый экземпляр проверяется в отношении к ближайшим соседям и рассматривается вместе со своим ближайшим соседством. Следовательно, выбор метода вычисления расстояния играет важную роль и существенно влияет на результаты и производительность.

С другой стороны, разделение всех обучающих данных на группы делает алгоритмы на основе кластеризации похожими на алгоритмы на основе классификации. Хотя в подходах на основе классификации класс назначается на основе заданных меток, в то время как в подходах на основе кластеризации классификация заранее не указывается [19].

3.1. МЕТОДЫ ОБНАРУЖЕНИЯ АНОМАЛИЙ

Одним из основных преимуществ методов, основанных на кластеризации, является способность большинства из них работать без присмотра, в неконтролируемом режиме. В случае получения данных о траекториях с видеокамер наблюдения, наиболее подходящими являются методы обучения без контроля, поскольку классификация и маркировка часов видеоданных является весьма трудоемкой задачей. Кроме того, ручная маркировка входных данных может привести к ошибкам из-за вмешательства оператора, как результат введение человеческого фактора.

Более того, методы на основе кластеризации могут приспособливаться к работе со сложными типами данных благодаря адаптируемости алгоритмов кластеризации. Однако в то же время они являются дорогостоящими в вычислительном отношении и используются в основном для сравнительно низкоразмерных данных, сильно зависимы от выбранного алгоритма кластеризации и не могут эффективно справляться с ситуациями, когда аномалии формируют значительные отдельные кластерные группы [14].

Методы с применением моделей [16][19] или Статистические [14]

Основная концепция алгоритмов, основанных на использовании моделей, заключается в том, что они представляют данные в виде набора параметров для создания модели нормального поведения. Одним из преимуществ таких подходов, основанных на использовании модели, является то, что они не требуют от пользователя ввода каких-либо входных параметров, поскольку все значения параметров могут быть получены из данных. Подходы, основанные на статистике, можно рассматривать как подкатегорию подходов, основанных на использовании моделей, и они считаются одними из самых ранних алгоритмов и могут использоваться в качестве основы различными методами обнаружения аномалий [17]. Как указано в [14], основная идея статистических подходов состоит в том, что экземпляры данных, возникающие в областях высокой вероятности стохастической модели, считаются нормальными, тогда как экземпляры данных из областей низкой вероятности относятся к аномалиям. Таким образом, статистические подходы основаны на использовании статистической стохастической модели для подгонки к заданным данным и последующем применении статистического теста вывода, также называемого тестом на несоответствие, чтобы определить, является ли экземпляр данных нормальным или аномальным. Из основной концепции следует, что «на основе результатов прикладного статистического теста аномалии имеют низкую вероятность быть сгенерированными из изученной стохастической модели» [14].

Статистические методы в свою очередь могут быть параметрическими или непараметрическими [17]. В параметрических подходах нормальные данные должны соответствовать параметрическому распределению и функции плотности вероятности с параметрами, вычисленными на основании заданных данных [16]. Одним из преимуществ параметрических методов является то, что размер данных не влияет на модель: модели растут только

в зависимости от сложности модели. Однако необходимость приспособить данные под какую-то заранее выбранную модель распределения усложняет и ограничивает применение таких подходов: сложности возникают при сопоставлении данных одному распределению. В этом случае можно использовать модель множественного распределения, чтобы сопоставить некоторые кластеры данных с конкретными распределениями [19]. Одним из наиболее известных примеров параметрических методов является метод регрессии [17].

В противоположность этому, непараметрические подходы основаны на использовании непараметрических статистических моделей со структурами, которые не определены заранее: исходные данные используются для динамического определения структуры. Такие подходы не основываются на заранее сделанных предположениях о статистическом распределении данных [17].

Поскольку статистические подходы основаны на подборе статистической модели, ее выбор существенно влияет на результаты, сложность вычислений и производительность. Тем не менее, основное допущение статистических подходов, которое заключается в том, что данные поступают из определенного распределения, не всегда может быть выполнено, особенно для случая многомерных данных [14].

Выводы

На основании приведенного описания различных подходов, их преимуществ и недостатков, было решено сосредоточиться на подходах обнаружения аномалий на основе кластеризации по нескольким причинам:

- 1) они могут работать в режиме без контроля без вмешательства оператора и человеческого фактора и не требуют наличия меток во входных данных,
- 2) входные данные могут содержать экземпляры аномальный таректорий,
- 3) метод кластеризации может быть легко применен к таким многомерным данным, как траектории, путем определения подходящей меры подобия.

Из этого следует, что необходимо определиться с методом кластеризации и метрикой измерения сходства траекторий.

3.2 Классификация методов кластеризации

Кластеризация является одной из наиболее глубоко изученных форм интеллектуального анализа данных, и огромное разнообразие методов кластеризации уже было предложено и реализовано [10]. Анализ современного состояния предметной области и связанных исследовательских работ показал, что все традиционные подходы кластеризации обычно делятся на пять типов: методы разделения (декомпозиции), иерархические,

основанные на плотности, основанные на моделях и методы на основе сетки [5][?]. В следующих параграфах будет кратко представлена концепция каждой из категорий с выделением основных предположений и обсуждением преимуществ и недостатков.

Методы, основанные на разделении или декомпозиции

Эта категория методов объединяет методы, основанные на случайном разделении данных траекторий, а затем перегруппировке кластеров путем переназначения объектов из одного кластера в другой для минимизации целевой функции. Методы требуют предварительного определения значения параметра, обычно обозначаемого как k , который определяет количество конечных кластеров, которые должны быть сформированы на базе исходных данных. Основное требование заключается в том, что количество итоговых разделов должно быть меньше числа исходных точек данных, поскольку каждый раздел образует кластер. Это означает, что каждый раздел должен быть непустым и содержать как минимум один экземпляр данных, а каждый экземпляр данных должен быть включен в ровно один кластер.

Одним из наиболее известных алгоритмов кластеризации, основанных на разделении, является алгоритм K -Means, в котором сначала k кластерных центров инициализируются случайным образом, а затем точки данных итеративно переназначаются в ближайший центр кластера на основе отличия, несоответствия, чтобы минимизировать итоговую ошибку кластеризации [24]. Ошибка кластеризации определяется как сумма квадратов Евклидовых расстояний между каждой точкой набора данных и соответствующим центром кластера [25]. Процесс прекращается, когда в кластерных центрах больше нет изменений.

Недостатками традиционного метода кластеризации K-Means являются невозможность формирования кластеров произвольной формы, сильная зависимость от начальной случайной инициализации центров кластеров и высокое потребление памяти [10]. Также сложной задачей является поиск подходящей техники разбиения.

Иерархические методы

В иерархических методах исходный набор данных разбивается на несколько уровней для организации в виде иерархического дерева кластеров. Полученная иерархическая структура может быть изображена в виде дерева [24].

Существует два различных способа иерархической декомпозиции: 1) восходящая (комбинирующая) и 2) разделяющая (декомпозиционная). Они называются агломеративной дивизивной (дивизионной) кластеризациями соответственно [26].

Агломеративные алгоритмы иерархической кластеризации начинаются с присвоения каждого экземпляра данных отдельному одноэлементному кластеру, при этом число начальных кластеров равняется точному количеству экземпляров данных во входных дан-

3.2. КЛАССИФИКАЦИЯ МЕТОДОВ КЛАСТЕРИЗАЦИИ

ных, а затем продолжается процесс объединения кластеров на основе их сходства, пока все начальные кластеры не будут объединены в один кластер или в заранее определенное количество кластеров [27]. Это делается путем многократного выполнения следующих двух шагов: 1) определение двух ближайших кластеров и затем 2) объединение этих двух кластеров [26]. Матрица близости кластеров используется для хранения значений схожести между кластерами и обновляется на каждом этапе путем вычисления расстояний между новым кластером и другими кластерами.

Дивизивные алгоритмы иерархической кластеризации работают в обратном порядке: сначала все экземпляры данных принадлежат одному кластеру, а затем шаг за шагом кластеры разбиваются на более мелкие кластеры, пока все кластеры не станут одноэлементными кластерами или пока не будут удовлетворены некоторые предварительно определенные конечные условия.

Предполагается, что иерархическая кластеризация является простым алгоритмом, однако возникает необходимость выбора между агломеративными и дивизивными методами. Дивизивная кластеризация более затратна в вычислениях, поэтому она менее распространена, нежели агломеративные подходы. Необратимость процессов расщепления или объединения кластеров в традиционных алгоритмах иерархической кластеризации также является особенностью и сложностью таких алгоритмов [10].

Поскольку агломеративный подход включает объединение кластеров, важной задачей алгоритмов агломеративной кластеризации является определение и вычисление сходства или расстояния между кластерами. Это сходство может также называться межкластерным расстоянием. В случае двух кластеров с одной траекторией в каждой из них сходство между ними упрощено и равно значению схожести между соответствующими траекториями. Для кластеров с несколькими траекториями сходство вычисляется в соответствии с выбранным методом связи. В литературе в качестве наиболее распространенных приводятся следующие методы связи: средняя ссылка (метод связи по одной минимальной ссылке, single linkage), максимальная ссылка (метод связи по максимальной ссылке, maximum linkage), средняя ссылка (метод связи по усредненной ссылке, average linkage) [24][28]. Выбор метода связывания зависит от домена приложения [26]. В single linkage расстояние между двумя кластерами определяется как минимальное расстояние между двумя траекториями из этих кластеров, что означает, что сходство между двумя кластерами определяется двумя ближайшими траекториями. Метод average linkage связывания подразумевает выбор максимального расстояния между двумя траекториями в двух кластерах в качестве межкластерного расстояния, то есть оно определяется с использованием самого дальнего расстояния между парами траекторий. Метод average linkage предполагает вычисление усредненного расстояния между всеми парами траекторий в рассматриваемых двух кластерах.

Удобство подходов агломеративной иерархической кластеризации заключается в том, что они не требуют предварительно заданного числа итоговых кластеров, поэтому они подходят для кластеризации траекторий ТС, поскольку количество кластеров с нормальными или аномальными траекториями заранее неизвестно. Однако наиболее известным недостатком алгоритмов иерархической кластеризации является то, что они не устойчивы к шуму и могут страдать от аномалий.

Плотностные алгоритмы

По сравнению с подходами основанной на разделении или иерархической кластеризации методы, основанные на плотности, определяют сходство на основе их плотности [22]. Область с данными добавляется к ближайшему кластеру, если плотность точек в области остается больше, чем предопределенное пороговое значение [10]. Кластеры образуют плотные области объектов и разделены разреженными областями с низкой плотностью.

Основное преимущество подходов кластеризации на основе плотности заключается в том, что они способны формировать кластеры произвольной формы, а не только сферической [10]. Также они подходят для бесконтрольной кластеризации огромных наборов данных траекторий и не требуют заранее заданного количества кластеров [5][22]. Однако качество результатов сильно зависит от количества траекторий в наборе обучающих данных, доступных для анализа: при малом размере обучающего набора данных результаты будут недостоверными и неточными.

Наиболее известным и широко используемым алгоритмом на основе плотности является DBSCAN, предложенный автором М. Эстер в [23]. В соответствии с этим алгоритмом точки входных данных могут быть классифицированы следующим образом: ядерные центральные точки (корневые или ядерные объекты, ключевые точки), плотно-достижимые объекты и аномалии на основе параметров ε , $minPts$ и пороговое значение плотности. Параметр ε и $minPts$ задают максимальную удаленность (максимальную окрестность точек) и минимальное количество удовлетворяющих точек при выборе ключевых точек: как минимум $minPts$ точек должны быть расположены внутри ε -окрестности от базовой точки; эти точки называются плотно-достижимыми из выбранной базовой точки. Вышеупомянутые параметры должны быть предварительно определены пользователем, но их трудно определить корректно. Каждый кластер должен содержать хотя бы одну базовую точку. Точки обозначаются как аномальные, если они не являются плотно-достижимыми ни от одной из других точек.

Алгоритмы на основе использования сеточной структуры данных

Основная идея алгоритмов кластеризации на основе использования сеточной структуры данных заключается в применении сеточной структуры данных с несколькими разрешениями: пространство данных квантуется на конечное число ячеек (единиц), которые

3.2. КЛАССИФИКАЦИЯ МЕТОДОВ КЛАСТЕРИЗАЦИИ

образуют многоразмерную структуру сетки. Каждая ячейка хранит общую информацию об объектах данных в своем подпространстве [22]. Поскольку операции кластеризации выполняются на созданной сетке, а также в каждой ячейке пространственной сетки могут быть вычислены важные характеристики траекторий, качество сжатия данных существенно влияет на качество результатов [7]. Плотность близко расположенных плотных клеток может помочь определить кластеры. Траектория может рассматриваться как аномальная, если она отличается от ожидаемой траектории количеством покрытых ячеек сетки [22].

Главным преимуществом таких алгоритмов кластеризации на основе сеточной структуры данных является улучшенная производительность: высокая скорость обработки и время обработки становятся независимыми от размера исходных данных, на время обработки влияет только количество ячеек в каждом измерении [10].

Подходы на основе использования моделей

По сравнению с вышеупомянутыми методами, которые анализируют расстояние между объектами данных, в модельных подходах предполагается, что данные генерируются несколькими распределениями вероятностей, где каждый компонент представляет собой кластер данных. Таким образом, математическая модель присваивается каждому кластеру, а затем метод пытается подобрать наиболее подходящие данные для выбранной модели. Методы этого типа стремятся повысить адаптивность между данными и некоторыми статистическими моделями [10][22]. Идея алгоритмов, основанных на использовании моделей, состоит в том, что для определения местоположения кластеров они описывают пространственное распределение точек входных данных с помощью определения функций плотности. Такие подходы обычно используются для кластеризации специфических данных и сильно зависят от выбранных функций и модели [5].

Подчеркивается, что подходы, основанные на использовании моделей, показывают хорошую производительность при работе со сложными типами данных. В эту категорию обычно входят статистические и нейронные методы [10].

Графовые алгоритмы [7]

Графовые алгоритмы представляют собой еще одну категорию методов кластеризации в применении к данным траекторий ТС. Liu в [29] представил основанный на графах подход для решения проблемы обнаружения аномалий в потоках данных транспортного трафика. Структура графика использовалась для хранения трафика: узлы представляют собой регионы, а граничные веса отображают поток трафика. Краевые аномалии на графике обозначают аномалии движения, и дерево причинных аномалий затем можно использовать для дальнейшего анализа этих аномалий для поиска причинных взаимодействий.

Другая высокоуровневая классификация методов кластеризации может состоять только из двух подклассов на основе свойств генерированных кластеров: иерархические подходы и подходы, основанные на разделении [24]. Иерархические алгоритмы группируют объекты в кластеры начиная с одноэлементных кластеров и доходя до одного итогового кластера, содержащего все экземпляры данных, или в обратном направлении. Второй подкласс алгоритмов кластеризации разделяют заданный набор данных на заранее определенное количество кластеров в одноуровневой структуре.

Для выполнения кластеризации необходимо определить метрику сходства между двумя траекториями. Различные существующие меры расстояния будут рассмотрены в следующих секциях.

Выводы

На основе приведенного описания различных подходов кластеризации, их ограничений, преимуществ и недостатков, было решено сосредоточиться на алгоритме иерархической кластеризации, а именно на агломеративной иерархической кластеризации, поскольку она может справиться с ограничениями используемых в этой работе входных данных, которые заключаются в следующем: отсутствие входных меток, неизвестное количество результирующих кластеров, присутствие как нормальных, так и аномальных траекторий во входных данных.

3.3 Метрики измерения близости и дальности

Как упоминалось ранее, подходы, основанные на кластеризации, требуют определения меры сходства между двумя траекториями. Кроме того, меры расстояния и подобия также используются для сравнения траектории с кластером или сравнения пары кластеров между собой. Выбор метрики сходства сильно зависит от формата траектории. Траектории, представленные в виде многомерных данных, могут содержать количественные или качественные характеристики, непрерывные или двоичные. В такой классификации функции измерения расстояния больше подходят для работы с непрерывными объектами, а меры сходства - для работы с качественно определенными траекториями [24]. Входные траектории-векторы в этой работе содержат пространственную информацию наряду с временной, которую можно назвать качественными непрерывными данными. Это означает, что в этом случае функции измерения расстояния между траекториями являются более подходящими.

Помимо этого, функции расстояния и подобия могут быть классифицированы как 1) способные принимать необработанные представления траекторий без каких-либо шагов предварительной обработки и 2) ожидающие на вход предварительно обработанные представления траекторий. Предварительная обработка может включать нификацию длины траекторий или уменьшение размерности векторов траекторий [28].

Одними из наиболее известных и широко используемых традиционных метрик подобия являются: Евклидово расстояние, расстояние Фреше, DTW, LCSS.

Евклидово расстояние (Euclidean Distance)

Евклидово расстояние между двумя векторами траекторий вычисляется как сумма квадратов разностей соответствующих пространственных координат [18]:

$$d_{ij} = \|T_i - T_j\|_E = \sqrt{\sum_{k=1}^m ((t_{i_x}^k - t_{j_x}^k)^2 + (t_{i_y}^k - t_{j_y}^k)^2)} \quad (3.3.1)$$

где обе траектории состоят из m точек слежения и представлены двумерными векторами $T_i = \{t_i^1, t_i^2, \dots, t_i^m\}$ и $T_j = \{t_j^1, t_j^2, \dots, t_j^m\}$. Кортежи $(t_{i_x}^k, t_{i_y}^k)$ представляют собой пространственные координаты для k -ой точки слежения i -ой траектории из набора данных.

Однако Евклидово расстояние работает только с траекториями с равным количеством точек отслеживания. Поскольку обычно ТС движутся с разной скоростью и по разному маршруту, длина траектории всегда различна. Это означает, что исходные траектории необходимо предварительно обработать и привести к единому размеру [28]. Кроме того, традиционное Евклидово расстояние работает с двумерными данными и не может обрабатывать временную информацию. Более того, Евклидово расстояние зависит от направления траектории: изменение направления на обратное может привести к неправильному измерению расстояния, что, в свою очередь, может привести к ошибкам в кластеризации. Также возможны ошибки при работе с траекториями, движущимися аналогичным образом, но с различными скоростями и частотой дискретизации (частотой измерения) [30].

Расстояние Фреше (Fréchet Distance)

В основе расстояния Фреше лежит Евклидово расстояние. Оно учитывает позиционные и последовательные отношения точек траектории при расчете схожести траекторий. Основная идея этого подхода состоит в том, чтобы вычислить Евклидово расстояние для каждой пары точек из двух траекторий, а затем обозначить максимальное Евклидово расстояние как расстояние Фреше между ними [10][31]. Однако, поскольку учитывается только максимальное расстояние, подход чувствителен к наличию аномалий.

Алгоритм динамической трансформации шкалы времени (DTW)

Алгоритм динамической трансформации шкалы времени (Dynamic Time Warping, DTW) является одним из алгоритмов измерения сходства между двумя последовательностями временных рядов, которые могут различаться по скорости. Целью методов сравнения временных рядов является создание метрики расстояния между ними. Метод DTW

направлен на поиск соответствия между зависимыми от времени последовательностями, такими как траектории, и способен обрабатывать траектории различной длины [10].

Согласно [10], DTW расстояние может быть вычислено следующим образом (3.3.2):

$$D_D(T_i, T_j) = \begin{cases} 0 & m = n = 0 \\ \infty & m = 0 \text{ or } n = 0 \\ dist(a_i^k, b_j^k) + \min \begin{cases} D_D(Rest(T_i), Rest(T_j)) \\ D_D(Rest(T_i), T_j) \\ D_D(T_i, Rest(T_j)) \end{cases} & \text{others} \end{cases} \quad (3.3.2)$$

где $D_D(T_i, T_j)$ - расстояние DTW между двумя сегментами траектории с длинами m и n , $dist(a_i, b_j)$ - Евклидово расстояние между двумя точками траектории. Функция $Rest(T_i)$ берет оставшуюся часть траектории после исключения последней точки траектории a_i . Из формулы видно, что в случае траекторий нулевой длины расстояние DTW равно 0, для случая, когда только одна из двух траекторий не пуста, расстояние между ними считается бесконечным. Для двух непустых траекторий минимальное расстояние между ними вычисляется рекурсивным способом.

Хотя важным преимуществом метода DTW является его способность обрабатывать векторы траектории различной длины, расстояние DTW не устойчиво к шуму и требует, чтобы точки траектории были непрерывными. Кроме того, вычисление расстояния DTW является очень трудоемким и сложным из-за необходимости сравнивать расстояния между каждой парой траекторий.

Метод наибольшей общей подпоследовательности (LCSS)

Расстояние Longest Common SubSequence (LCSS) пытается сравнивать две последовательности траекторий на основе самой длинной общей подпоследовательности между ними. Алгоритм LCSS работает с дискретными значениями и рассчитывает наибольшее количество эквивалентных точек между двумя траекториями. Задача поиска самой длинной общей подпоследовательности обычно решается рекурсивно [10]: возможные смещения или сдвиги рассчитываются в каждом измерении и используются для обеспечения максимальной LCSS [32]. Основная идея расстояния LCSS заключается в том, что оно позволяет растягивать две траектории. По сравнению с DTW и Евклидовыми расстояниями, LCSS позволяет некоторым элементам (точкам траектории) оставаться несогласованными [33], и, по сравнению с DTW, LCSS более устойчив к наличию аномалий [34].

LCSS метрика может быть рассчитана по следующей формуле (3.3.3) [28]:

$$D_{LCSS}(T_1, T_2) = 1 - \frac{LCSS_{\delta, \epsilon}(T_1, T_2)}{\min(m, n)} \quad (3.3.3)$$

где m и n - длины траекторий T_1 и T_2 соответственно. $\frac{LCSS_{\delta, \epsilon}(T_1, T_2)}{\min(m, n)}$ также может называться метрикой схожести LCSS и принимает значение от 0 до 1.

$LCSS_{\delta, \epsilon}(T_1, T_2)$, самая длинная общая подпоследовательность между траекториями, представляет собой количество совпадающих точек траектории между траекториями T_1 и T_2 и определяется следующим образом (3.3.4):

$$LCSS_{\delta, \epsilon}(T_1, T_2) = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ 1 + LCSS_{\delta, \epsilon}(\text{Head}(T_1), \text{Head}(T_2)) & \begin{array}{l} (\text{if } |t_{1_{x,m}} - t_{2_{x,n}}| < \epsilon \\ \text{and } |t_{1_{y,m}} - t_{2_{y,n}}| < \epsilon \\ \text{and } |m - n| \leq \delta) \end{array} \\ \max \begin{cases} LCSS_{\delta, \epsilon}(\text{Head}(T_1), T_2) \\ LCSS_{\delta, \epsilon}(T_1, \text{Head}(T_2)) \end{cases} & \text{otherwise} \end{cases} \quad (3.3.4)$$

Как видно из приведенной формулы, вычисление LCSS зависит от двух постоянных параметров: δ (расположение точек [32]) и ϵ (расстояние между точками [32] или соответствующее пороговое значение для сравнения пространственной близости точек [33]):

- параметр δ определяет максимальную удаленность в отношении времени между двумя точками траектории, в пределах которой мы можем искать соответствие данной точки траектории другой. Также может быть определено как значение, представляющее максимальную разность индексов между двумя входными траекториями, разрешенными при расчете [32].
- константа ϵ определяет минимальную границу пространственной близости при поиске совпадений. Согласно [32], это число с плавающей запятой, которое представляет собой максимально допустимое расстояние между точками траектории в каждом измерении, чтобы считать их эквивалентными: разница между X - и Y -координатами меньше, чем значение ϵ означает, что точки находятся относительно близко друг к другу и могут рассматриваться как эквивалентные. В этом случае расстояние LCSS увеличивается на 1.

Параметры δ и ϵ существенно влияют на результаты, поэтому задача выбора оптимальных значений для них является очень важной и сложной [28][30]. Функция $\text{Head}(T)$ определена так, чтобы возвращать первые $M - 1$ точки из траектории T , представляющие

траекторию с удаленной последней точкой траектории. Согласно реализации, приведенной в [32], вычисление LCSS, основанное на подходе динамического программирования, имеет сложность $O((m + k)\delta)$. Однако алгоритм требует предопределенных постоянных значений параметров δ и ε в качестве входных данных для метода. Кроме того, из-за рекурсивного способа вычислений LCSS имеет высокую вычислительную стоимость при работе с траекториями большой длины [35].

Выводы

Метрика расстояния LCSS является наиболее подходящей в этой работе, поскольку она позволяет исходным траекториям содержать шум и аномалии, иметь различную длину, скорость объектов и частоту дискретизации (локальные временные сдвиги в траекториях) [28]. Кроме того, среди вышеупомянутых методов расстояние LCSS является наиболее надежным подходом против помех.

3.4 Выводы

Родственные методы

Вышеупомянутая цель была исследована и решена в многочисленных работах с использованием различных методов. Поскольку на самом деле нормальные события являются более частыми и доминируют в данных, а ненормальные события редки и их трудно описать явно, многие подходы основаны на неконтролируемой кластеризации траекторий. Для этой дипломной работы в качестве основы был выбран подход, предложенный авторами Ghrab, Fendri, Hammami в [28]. Он направлен на выявление отклонений с помощью кластеризации траекторий.

Предложенный подход можно описать как двухэтапный подход с автономной оффлайн кластеризацией для выявления частых траекторий и онлайн классификацией входной траектории для обозначения ее как нормальной или аномальной.

Кластеризация выполняется неконтролируемым образом с использованием алгоритма агломеративной иерархической кластеризации, работающего на матрице расстояний между траекториями. Для выполнения кластеризации расстояние LCSS используется в качестве меры сходства траекторий. Формулы и описание расстояния LCSS приведены в предыдущем разделе (3.3.3-3.3.4).

Преимущества

Одним из преимуществ предлагаемого способа является то, что выбранная мера подобия не требует, чтобы траектории были одинаковой длины, так что можно избежать предварительной обработки траекторий, что само по себе является процессом высокой сложности. Более того, тренировочные данные могут содержать как нормальные, так и аномальные траектории: алгоритм будет выделять как нормальные, так и аномальные кла-

стеры. Кластеры с высокой плотностью объектов будут представлять классы нормальных траекторий, разреженные кластеры - классы аномальных траекторий.

Недостатки

Однако недостатком предлагаемого способа является то, что расстояние LCSS не учитывает такие проблемы данных видеонаблюдения, как перспектива и положение движущегося объекта относительно видеокамеры.

Поставленные задачи

Таким образом, эта дипломная работа будет направлена на изучение возможности повышения точности результатов путем использования адаптивных параметров δ и ε , которые используются для расчета LCSS расстояния, зависящих от перспективы и расстояния от камеры. Это включает выполнение следующих задач:

- изучение наличия и природы функциональной зависимости, возможно существующей между параметрами δ и ε и расстоянием от камеры,
- протестировать результаты работы предложенного подхода на различных входных значениях параметров.

4 ПРЕДЛОЖЕННЫЙ ПОДХОД

В этой главе будет дано описание концептуальной модели предложенного подхода. В первых секциях описаны базовая схема работы и архитектура системы. В последующих секциях представлены детали используемых методов для обработки входных данных и аппроксимации траекторий, схематично изложены основные алгоритмы.

4.1 Концептуальная модель фреймворка

Основной целью данной работы является анализ пространственно-временных данных о траекториях движущихся ТС, полученных с камер видеонаблюдения, и решение задачи определения аномалий. Для решения поставленной задачи подход, основанный на кластеризации, был выбран как основа: сначала входные траектории используются как обучающие данные для определения кластеров и их моделирования; кластеры классифицируются как нормальные или аномальные в зависимости от количества траекторий в кластере; затем классификатор помечает входную траекторию как нормальную или аномальную на основании обученной модели кластеров.

Основной вклад этой работы заключается в попытке минимизировать проблему, связанную с неопределенностью данных, и повысить точность результатов путем использования адаптивных параметров при подсчете схожести траекторий: в новом подходе будет учитываться позиция движущегося объекта по отношению к камере. Для этих целей был разработан фреймворк, решающий задачи определения частых траекторий движения и обнаружения аномальных.

Основной рабочий процесс фреймворка состоит из следующих этапов (Figure 4.1.1):

- обработка входных данных и парсинг траекторий,
- фильтрация траекторий на основании количества точек траектории и итогового пройденного расстояния,
- аппроксимация траекторий с использованием полиномиальной регрессии,
- подсчет схожести траекторий и заполнение “матрицы подобия”,
- кластеризация траекторий,
- определение нормальных и аномальных кластеров, визуализация кластеров,
- моделирование нормальных кластеров

- классификация входной траектории как нормальной или аномальной.

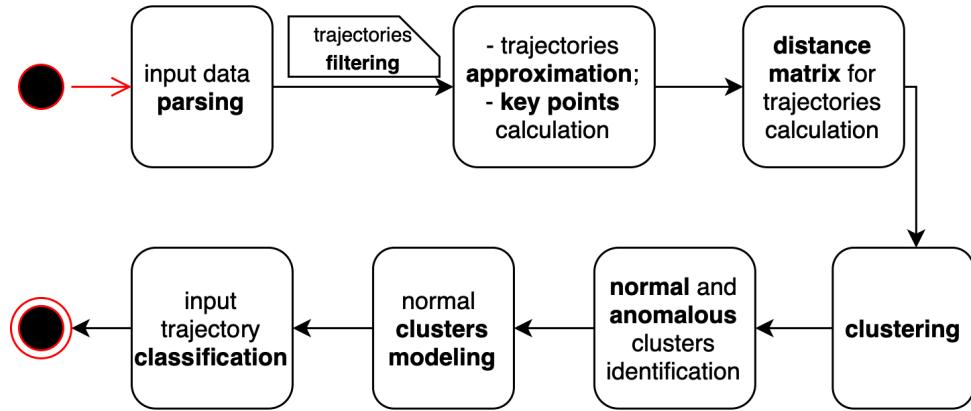


Рис. 4.1.1: Основные этапы работы фреймворка

4.2 Архитектура фреймворка

Архитектура предложенного подхода основана на работе [28], подробно описанной ранее, и состоит из двух частей (4.2.1):

- *offline* – кластеризация и определение паттернов наиболее часто встречающихся траекторий, и
- *online* – классификация новой траектории как нормальной или аномальной.

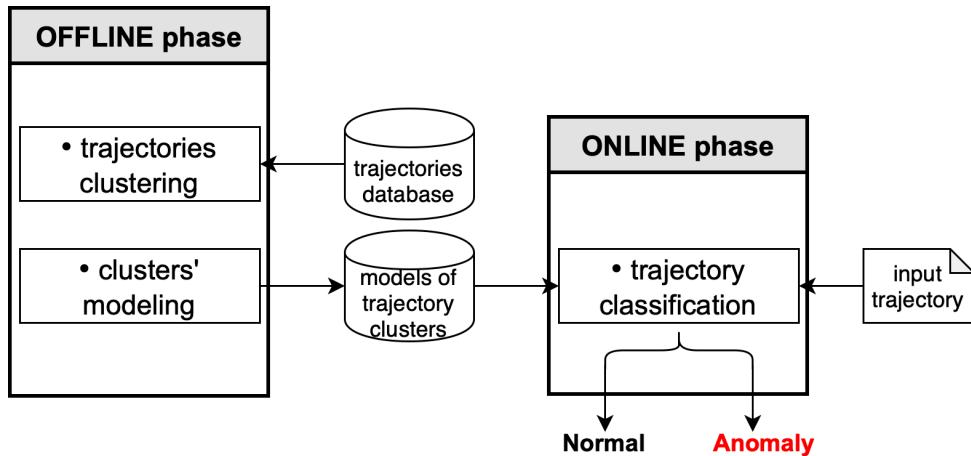


Рис. 4.2.1: Схема двухэтапного предложенного подхода

Фреймворк состоит из нескольких модулей, каждый из которых отвечает за выполнение одной из задач из вышеперечисленных этапов рабочего процесса (Figure 4.2.2):

- *entity* – содержит классы-сущности для хранения таких объектов, как *Trajectory*, *TrajectoryPoint*, *Cluster* и др.;
- *parsing* – отвечает за парсинг входных данных: считывает данные из ’txt’-файла и создает объекты *TrajectoryPoint* и *Trajectory*;
- *csv* – изолирует логику работы с ’csv’-файлами, содержит методы чтения и записи, которые используются сначала для сохранения подсчитанных LCSS метрик, а затем для их подгрузки для дальнейшей кластеризации;
- *approximation* – содержит реализацию аппроксимации траекторий методом полиномиальной регрессии и необходимые сопутствующие классы, такие как класс *Polynomial*, расширяющий функционал стандартного *PolynomialFunction* из библиотеки Apache Math, и др.;
- *visualization* – отвечает за визуализацию и сохранение полученных результатов, содержит методы для чтения, изменения, сохранения объектов класса *BufferedImage*;
- *clustering* – состоит из класса *Clustering*, который содержит методы для подсчета значений LCSS метрики, кластеризации объектов класса *Trajectory* и создания объектов класса *Cluster*;
- *exception* – содержит исключения, которые теоретически могут быть выброшены в ходе работы фреймворка (например, *TrajectoryParserException*);
- *misc* – содержит вспомогательные утилитные классы для хранения константных значений и базовых методов, общих для всех модулей разрабатываемой системы.

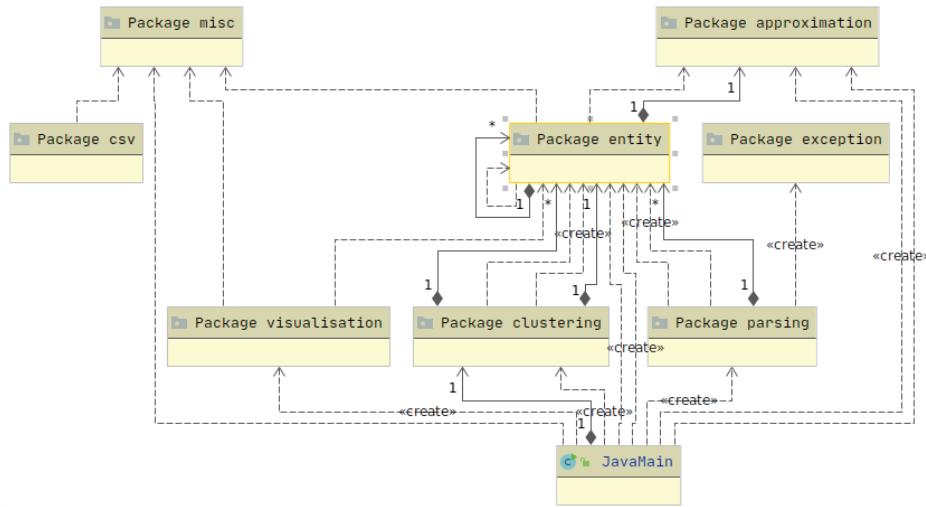


Рис. 4.2.2: Архитектура фреймворка

4.3 Кластеризация

4.3.1 Агломеративная иерархическая кластеризация

Для кластеризации был выбран агломеративный иерархический алгоритм кластеризации, работающий в режиме бесконтрольного обучения (unsupervised learning). Описание этого подхода приведено в алг. 1 [28].

Algorithm 1: Описание Агломеративной Иерархической кластеризации

Input: A Database of Trajectories: trajectories

Output: Clusters of Trajectories: clusters

Initialization:

- initialize the clusters with one trajectory in each cluster

Clusters merging:

while number of clusters is greater than 1 **do**

- calculate similarity matrix D between pairs of clusters based on single linkage approach using LCSS similarity measure;
- find the smallest distance between clusters in D;
- merge two clusters with the corresponding smallest distance into a single cluster;
- remove two merged clusters;

end

Как уже упоминалось ранее, методы агломеративной иерархической кластеризации предполагают объединение кластеров, что требует определения меры расстояния между кластерами. В [28] авторы провели сравнительный анализ различных связующих методов для определения схожести кластеров (linkage methods), включая single (метод связи по одной минимальной ссылке), maximum (метод связи по максимальной ссылке) и average

(метод связи по средней ссылке) методы. Согласно проведенным тестам, single метод показал наилучшие результаты и поэтому будет использоваться в качестве метода связывания в текущей работе.

Метод связи по минимальной ссылке предполагает выбор минимального расстояния между двумя траекториями в качестве межкластерного расстояния и может быть представлен как [28]:

$$D_{min}(C_i, C_j) = \min_{T_1 \in C_i, T_2 \in C_j} D_{LCSS}(T_1, T_2), \quad (4.3.1)$$

где (C_i, C_j) обозначают два кластера, а (T_1, T_2) соответствуют двум траекториям из двух кластеров соответственно.

4.3.2 Моделирование кластеров

Чтобы более эффективно выполнить дальнейшую классификацию входной траектории, в выбранной в качестве основы работе было предложено создать модели кластеров (модели траекторий), классифицированных как нормальные. Моделью кластера называется ее компактное представление. Модели нормальных кластеров можно рассматривать как шаблоны частых траекторий, поскольку они представляют собой весь кластер траекторий с максимальной точностью.

Существуют две основные концепции построения модели кластера [36]:

- выбор репрезентативной траектории из кластера. Такая траектория считается центром кластера. Самым простым способом обозначить репрезентативную траекторию является определение только ее центроида (центроид кластера, путь центроида). В качестве дополнения центроид может быть расширен определением диапазона, области пути центроида;
- определение модели на основании траекторий, относящихся к кластеру, с использованием вероятностных моделей, таких как Гауссовские наблюдательные эмиссионные HMM (Hidden Markov Model). Такой метод требует предварительной обработки траекторий и показывает лучшие результаты на вероятностно моделируемых траекториях.

По сравнению со второй концепцией, выбор репрезентативной траектории для каждого кластера в качестве модели является более простым методом. Более того, он не требует, чтобы траектории имели одинаковое количество точек траектории [28].

Авторы [28] предлагают легкий подход к моделированию кластера без какой-либо предварительной обработки траекторий: модель кластера - это траектория, наименее всего удаленная от других, в результате чего ее можно рассматривать как репрезентативный

центр кластера. Это означает, что необходимо определить траекторию с минимальным средним расстоянием LCSS до других траекторий в этом кластере (4.3.2):

$$CM(C) = \min_{T \in C} \frac{1}{|C|} \sum_{T' \in C} D_{LCSS}(T, T') \quad (4.3.2)$$

4.4 Сравнение траекторий

Для измерения схожести и расстояния между траекториями для выполнения кластеризации в этой работе будет использоваться LCSS расстояние. LCSS расстояние подразумевает вычисление самой длинной общей подпоследовательности между двумя входными траекториями с использованием двух значений параметров: δ и ε .

4.4.1 LCSS метрика для измерения схожести траекторий

Согласно методу LCSS, параметры δ и ε являются постоянными и задаются заранее. Однако в текущем предлагаемом подходе для работы с неопределенностью данных о траектории, содержащих точки траектории, расположенные произвольным образом по отношению к камере, было решено реализовать адаптивность значений параметров. Будет исследована функциональная зависимость параметров от положения движущегося объекта на перекрестке относительно камеры.

При визуальном анализе траекторий, отображенных на исходных изображениях с камер, можно сделать несколько выводов: поскольку нижняя часть изображения представляет область, расположенную ближе к камере наблюдения, движущиеся объекты в верхней части изображения более удалены от камеры, и в результате более плотно расположены относительно друг друга на изображении. ε представляет собой пороговую величину, регулирующую пространственную удаленность точек траектории при подсчете расстояния между ними. Следовательно, он должен быть адаптирован к удаленности и уменьшаться по мере удаления точки траектории от камеры.

Описание алгоритма подсчета LCSS приведено в алг. 2.

4.4.2 Адаптивность параметров LCSS алгоритма

Классический метод подсчета LCSS предполагает использование постоянных значений для параметров δ и ε и не подразумевает их адаптивности. Тем не менее, одной из целей в этой работе является исследование возможности повышения точности результатов путем изучения функциональной зависимости между этими параметрами и удаленностью движущейся точки от камеры.

По причине того что камера расположена в фиксированном месте на перекрестке, возможны проблемы, связанные с перспективой. Из рис. 5.2.2, на котором изображено распределение исходных траекторий, видно, что при удалении от камеры, расположенной

Algorithm 2: Описание алгоритма LCSS метрики

Input: First trajectory: T_1 ,
 Second trajectory: T_2 ,
 Temporal remoteness threshold: δ ,
 Spatial remoteness threshold: ε

Output: LCSS distance for two trajectories

begin

- // Initialization
 - calculate length of T_1 ;
 - calculate length of T_2 ;
- // LCSS similarity calculation
 - if** T_1 or T_2 is empty **then**
 - | return 0;
 - else**
 - if** difference between X-coordinates $< \varepsilon$
 AND difference between Y-coordinates $< \varepsilon$
 AND difference between trajectory lengths $< \delta$ **then**
 - | increase LCSS by 1;
 - | call recursive for trajectories excluding last points;
 - else**
 - | calculate LCSS for first trajectory and second trajectory excluding last point;
 - | calculate LCSS for first trajectory excluding last point and second trajectory;
 - | take maximum between these LCSS values;
 - end**
 - end**
 - // LCSS distance calculation
 - LCSS distance = $1 - \text{LCSS similarity} / \min(\text{input lengths})$

end

в нижней передней части изображения, траектории становятся более плотно расположеными относительно друг друга (входные данные будут описаны и изображены в следующей главе). В случае константного значения ε , контролирующего допустимое пространственное отклонение между точками траектории при тестировании пространственного равенства точек траектории, точки траектории, расположенные далеко от камеры (то есть появляющиеся в верхней части изображений с камер и расположенные более плотно друг к другу), будут неправильно признаны похожими. Это может негативно повлиять на последующий анализ и исказить дальнейшие результаты кластеризации.

Следовательно, значение коэффициента ε должно меняться в соответствии с расстоянием от местоположения камеры: ε должен уменьшаться по мере удаления точки траектории от камеры и увеличиваться по мере приближения точки траектории к камере. Таким образом, можно предположить, что ε и расстояние от камеры находятся в обрат-

ной зависимости, что означает, что в итоговой формуле расстояние от камеры должно присутствовать в знаменателе формулы с некоторым коэффициентом.

Подход к определению адаптивных параметров, рассмотренный в данной работе, описан в алг. 3 и изображен на рис. 4.4.1.

Algorithm 3: Определение адаптивных параметров LCSS

Input: First trajectory point: TP_1 ,
Second trajectory point: TP_2
Output: Adaptive ε value for TP_1 and TP_2

```

begin
    // Initialization
    - compute location of a camera point ( $CP$ );
    - compute Euclidean distance for two pairs  $d_1(TP_1, CP)$  and  $d_2(TP_2, CP)$ ;
    - compute corresponding  $\varepsilon_1$  and  $\varepsilon_2$  values;
    - take the  $\max(\varepsilon_1, \varepsilon_2)$  as a final  $\varepsilon$  to compare  $TP_1$  and  $TP_2$ .
end

```

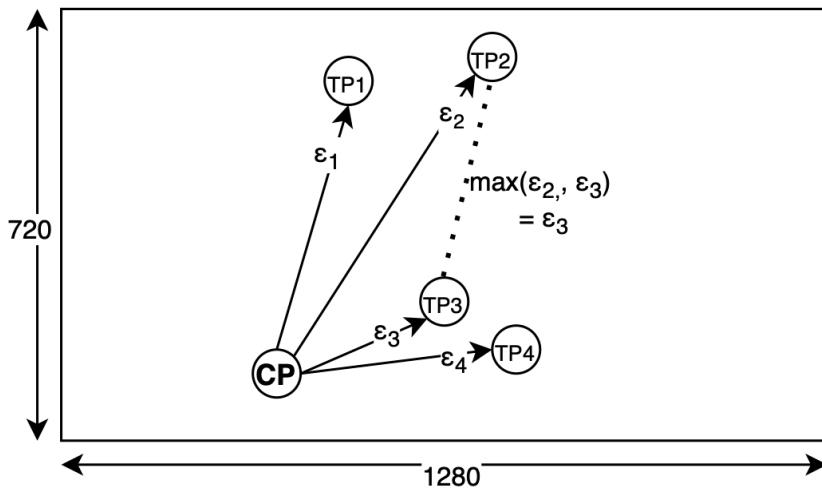


Рис. 4.4.1: Принцип выбора адаптивного значения ε

Чтобы оптимизировать подсчет расстояния от точки траектории до точки камеры и избежать его многократного пересчета во время работы алгоритма вычисления LCSS, это расстояние рассчитывается заранее и сохраняется для каждой точки траектории.

4.5 Проверка достоверности кластеров

Поскольку целью настоящей работы является поиск оптимальных значений адаптивных параметров для вычисления метрики сходства траекторий, необходимо проанализировать и сравнить полученные результаты после выполнения кластеризации.

Согласно [37] метрики проверки достоверности результатов кластерного решения могут быть классифицированы следующим образом:

- **Internal cluster validation** – внутренняя метрика оценки кластеров. Результат кластеризации оценивается на основе кластеризованных входных данных. Он основан на внутренней информации и не содержит ссылок на внешнюю информацию.
- **External cluster validation** – внешняя метрика оценки кластеров. Оценка результатов кластеризации выполняется в соответствии с известными извне результатами, например, данными метками класса. Такая проверка не подходит для неконтролируемой (unsupervised) кластеризации, поскольку в таком случае входные метки неизвестны.
- **Relative cluster validation** – относительная метрика оценки кластеров. Оценка результатов кластеризации выполняется с помощью одного и того же алгоритма с использованием разных входных параметров, таких как количество кластеров и т.д..

В то же время кластеризация - это, прежде всего, метод бесконтрольного неконтролируемого извлечения данных, использующий входные данные, не содержащие меток данных. Это приводит к необходимости проверять итоговые кластеры неконтролируемым образом.

Одной из наиболее широко используемых и известных метрик для оценки алгоритмов кластеризации является индекс достоверности Данна (Dunn's Validity Index, DI), который был предложен Дж. К. Данном в 1974 году в [38]. Это внутренняя метрика оценки, предназначенная для идентификации компактных кластеров с небольшой дисперсией между элементами кластера, которые хорошо отделены друг от друга, то есть кластеры достаточно удалены от окружающих кластеров по сравнению с межклusterной дисперсией [39]. DI рассчитывается как отношение минимального межклusterного расстояния d_{min} к максимальному внутриклusterному диаметру d_{max} , и для k кластеров может быть определен следующим образом (4.5.1) [40]:

$$DI = \frac{d_{min}}{d_{max}} = \frac{\min_{\substack{1 \leq i \leq k \\ i+1 \leq j \leq k}} dist(c_i, c_j)}{\max_{\substack{1 \leq l \leq k}} diam(c_l)} \quad (4.5.1)$$

где минимальное межклusterное расстояние d_{min} в соответствии с методом связи по минимальной ссылке сводится к подсчету минимального расстояния между двумя траекториями из разных кластеров. Максимальный внутриклusterный диаметр d_{max} , или самое большое внутриклusterное расстояние, предполагает вычисление диаметра кластера как расстояния между его двумя самыми дальними траекториями [41].

Пример определения DI для 3 кластеров приведен на рис. 4.5.1. Согласно этому примеру (4.5.2) может быть переписан следующим образом:

$$DI = \frac{d_{min}}{d_{max}} = \frac{\min(dist_{min}^1, dist_{min}^2, dist_{min}^3)}{\max(diam_{max}^1, diam_{max}^2, diam_{max}^3)} \quad (4.5.2)$$

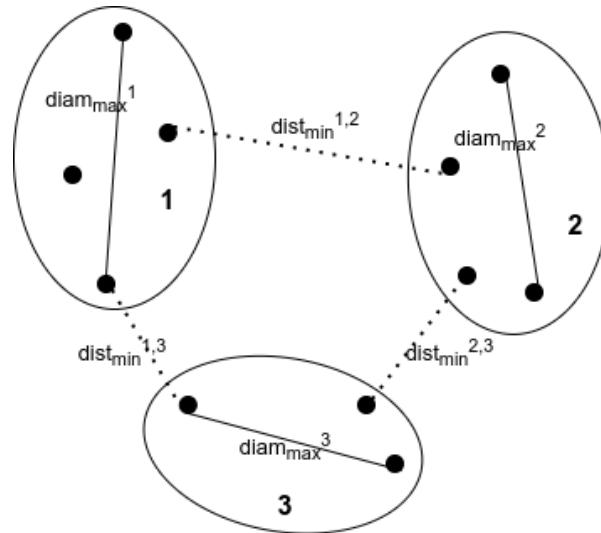


Рис. 4.5.1: Объяснение индекса DI

Более высокие значения DI указывают на лучшие результаты кластеризации. Траектории, расположенные далеко друг от друга внутри одного кластера, должны отличаться от траекторий, относящихся к другим кластерам. Близкие к 1 значения DI означают, что минимальные расстояния до траекторий из разных кластеров остаются больше, чем расстояние до самых дальних траекторий внутри одного кластера. Однако вычислительная стоимость DI сильно зависит от данных: вычислительная сложность увеличивается с ростом количества кластеров и размерности данных [37].

4.6 Аппроксимация траекторий

Несмотря на то что метрика подобия LCSS работает с траекториями произвольных длин и по умолчанию не требует предварительной обработки траекторий, подсчет LCSS метрики становится чрезвычайно вычислительно сложным и времязатратным с ростом длины траектории в результате рекурсивности. По этой причине в текущей работе было решено уменьшить размер траекторий путем аппроксимации исходных траекторий. Это приводит к потере точности, но позволяет получить приемлемые результаты за адекватное количество времени.

Концепция подбора аппроксимирующей кривой функции - это один из стандартных подходов для выполнения аппроксимации [42]. Основная задача - найти подходящее соотношение или закон, возможно существующий между входными (независимыми) и выходными (зависимыми) переменными для заданного набора входных данных наблю-

даемых значений. И подбор кривой - это процесс выражения связи между переменными в терминах алгебраических уравнений. Основная цель подбора кривой - найти параметры для модели (уравнения или функции), подходящей для экспериментальных данных.

4.6.1 Регрессионный анализ

Одним из наиболее широко используемых подходов, основанных на концепции подбора кривой, является регрессионный анализ, который также рассматривается как форма подхода прогнозирующего моделирования и, согласно традиционному определению, изучает взаимосвязь между зависимой переменной (результатом) Y и одним или более независимыми переменными X и, как правило, находит тренды в данных. Другими словами, он предполагает «использование отношения между переменными для подбора линии наилучшего соответствия или уравнения регрессии, которое можно использовать для прогнозирования» [43].

С целью упростить процедуру подбора аппроксимирующей функции обычно предполагается, что независимые переменные X измеряются без ошибок, в то время как значения зависимых переменных Y измеряются с некоторой случайной ошибкой. Для данных с небольшим отношением ошибки измерения в независимой переменной к диапазону значений этой переменной можно правомерно использовать регрессионный анализ по методу наименьших квадратов [42].

Регрессия может быть линейной или полиномиальной (нелинейной, криволинейной) в зависимости от функции, которая аппроксимирует данные: линейная регрессия применима к отношениям, аппроксимируемым прямой линией, тогда как криволинейная регрессия относится к отношениям, аппроксимируемым кривой. Благодаря более широкому диапазону функций, с которыми может работать полиномиальная регрессия, она обеспечивает лучшее приближение входных отношений по сравнению с линейной регрессией [43]. Даже если невозможно заранее определить тип функции для аппроксимации, чтобы получить наивысшую точность результатов, может оказаться полезным визуализация и анализ отображения исходных данных для нахождения какого-либо поведенческого паттерна, такого как линейная, квадратичная или зависимость более высокого порядка [42].

4.6.2 Полиномиальная регрессия

Визуализация исходных данных траекторий представлена на рис. 5.2.2. Как видно из рисунка, ни линейные функции, ни функции 2-го порядка не могут соответствовать данным должным образом из-за сложности форм траектории. По этой причине было решено сосредоточиться на приближении с использованием полиномиальной регрессии высшего порядка. Оценка полиномиальной регрессии с различными степенями будет приведена далее, и последующее обсуждение и реализация будут направлены на то,

чтобы найти подходящее полиномиальное уравнение n -го порядка со значениями параметров для представления каждой входной траектории в качестве «функции траектории». Поскольку данные траектории представлены двумерными пространственными данными вместе с временными данными, и необходимо аппроксимировать пространственную информацию, x - и y -координаты будут рассматриваться как зависимые переменные, а $time$ будет использоваться как независимая переменная. Следовательно, полиномиальная регрессия будет выполняться дважды с двумя выходными полиномиальными функциями, представляющими $x(t)$ и $y(t)$ для каждой из входных траекторий T :

$$\forall T = [\dots (x_i, y_i, t_i) \dots] \Rightarrow T(t) = \begin{cases} x = x(t) \\ y = y(t) \end{cases} \quad (4.6.1)$$

Таким образом, траектории будут преобразованы из формата списка точек траекторий в уравнения (функции времени), определенные в геометрическом пространстве, которые могут с высокой точностью представлять все исходные траектории. Выбор ключевых точек репрезентативного полинома может уменьшить размер траектории, тем самым сокращая итоговые эксплуатационные затраты и вычислительную сложность LCSS. Кроме того, математические уравнения способны хранить информацию в плотной форме, и, помимо других преимуществ, такое сокращение данных приводит к уменьшению занимаемого пространства и повышению эффективности хранения [42]. Также так называемые встроенные «функции траектории» могут обеспечивать интерполяцию и позволяют определять пропущенные, потерянные точки траекторий.

5 РЕАЛИЗАЦИЯ ФРЕЙМВОРКА

В этой главе дается описание разработки фреймворка с предоставлением деталей реализации описанной ранее концепции на базе выбранного стека технологий. На основе изложенного ранее рабочего процесса фреймворка были реализованы отдельные модули, подробное описание каждого из них представлено в следующих разделах. Первые разделы описывают обработку и аппроксимацию входных данных траектории. В последующих разделах представлены детали реализации решения задач кластеризации, моделирования кластеров и классификации входных траекторий. По причине того что не было найдено подходящих готовых реализаций для выбранных алгоритмов, все реализации алгоритмов, кроме полиномиальной регрессии и решения полиномиальных уравнений, были написаны с нуля и представлены в Приложениях.

5.1 Используемые технологии

Для реализации фреймворка был выбран язык программирования Java с сопутствующими библиотеками и Apache Maven в качестве инструмента для автоматизации сборки для проектов Java следующих версий:

- Java - 11 OpenJDK
- Apache Maven - 3.6.3
- Commons Math: The Apache Commons Mathematics Library - 3.4.1
- Java AWT, Javax Swing

Java используется как основной язык программирования. Библиотека Commons Math library была выбрана для реализации аппроксимации траекторий, поскольку она предлагает реализацию для таких классов, как *PolynomialFunction* и *PolynomialSolver*.

Java AWT (Abstract Window Toolkit) представляет собой ИПП (Интерфейс прикладного программирования, API) для реализации ГПИ (Графический Пользовательский Интерфейс, GUI) в Java приложениях и поставляется как часть Java JDK начиная с версии OpenJDK 7. В данной работе Java AWT будет использоваться преимущественно для создания и манипуляции объектами *BufferedImage* для визуализации исходных изображений с камер и отображения траекторий на них.

Java Swing, так же как и Java AWT, является инструментальной библиотекой, созданной для тех же целей предоставления ГПИ приложениям, написанным на Java,

и реализации возможности написания приложений, основанных на применении окон (оконных приложений). Однако Swing является более новой и продвинутой библиотекой, поддерживающей более сложные, усовершенствованные ГПИ-компоненты, нежели Java AWT. В текущей работе Swing используется для создания окон для визуализации изображений с исходными траекториями, результатами аппроксимации и кластеризации.

5.2 Работа со входными данными

5.2.1 Описание входных данных

Согласно исследованию, проведенному Министерством транспорта США на основе данных Системы отчетов об анализе смертности (Fatality Analysis Reporting System, FARS) и Национальной системы отбора проб автомобилей (National Automotive Sampling System), почти 40 процентов всех зарегистрированных в 2008 году аварий произошли на перекрестках [44]. Следовательно, в настоящее время анализ транспортной активности на перекрестках на перекрестках дорог имеет большое значение в контексте безопасности, и выявление небезопасных транспортных траекторий, нарушающих ПДД, может быть одним из шагов на пути к улучшению статистики.

В представленной работе видео с камер слежения используется для обучения и тестирования системы. Тестовые видео получены с помощью ИТС, реализованных и установленных на четырех перекрестках в г. Казань:

1. Перекресток улиц Право-Булачная - Пушкина, 1.txt (рис. 5.2.1a).
2. Перекресток улиц Несмелова - Кировская Дамба, 2.txt (рис. 5.2.1b).
3. Перекресток улиц Московская - Галиаскара Камала, 3.txt (рис. 5.2.1c).
4. Перекресток улиц Московская - Парижской Коммуны, 4.txt (рис. 5.2.1d).

Каждый перекресток представляет собой четырехстороннее пересечение дорог, оборудованное одной регистрирующей камерой. Примеры изображений с установленных видеокамер представлены на рис. 5.2.1.

Файлы с входными данными содержат 624, 211, 231, 237 траекторий ТС для каждого из указанных перекрестков соответственно.

Под аномальной траекторией подразумевается траектория движения ТС через перекресток, которая значительно отличается от большинства известных, нормальных траекторий. Например, в случае, когда на перекрестке запрещен поворот направо с крайней левой полосы движения, такое поведение будет неизвестно системе. В случае появления такой траектории она будет признана системой аномальной.



Рис. 5.2.1: Примеры изображений с видеокамер на перекрестках 1-4

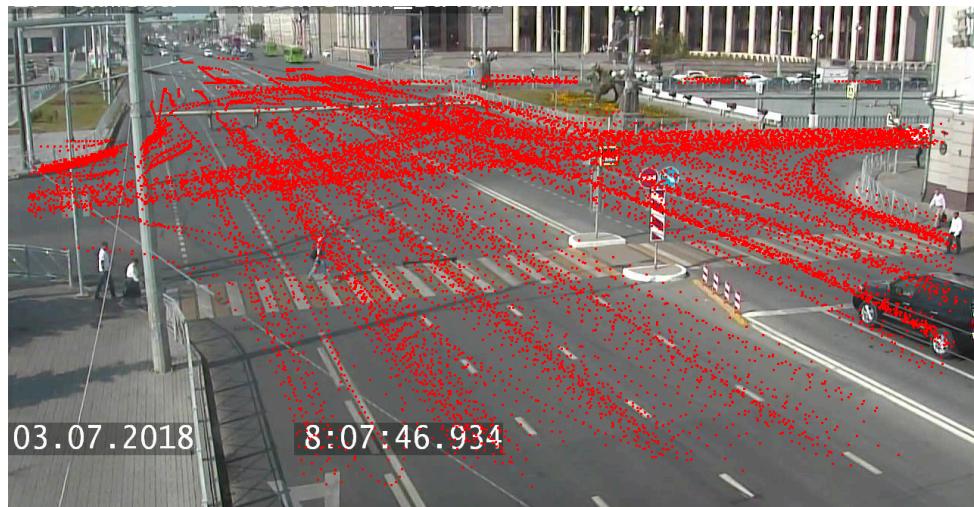


Рис. 5.2.2: Пример работы системы слежения на данных с первого перекрестка

5.2.2 Структура файла входных данных

Система отслеживания, как было описано ранее, обрабатывает видео с камер слежения и подготавливает его для дальнейшего анализа: преобразует видеопоток в набор векторов с точками отслеживания (далее точки траектории) на изображениях (рис. 5.2.2).

Файлы входных данных имеют следующую структуру:

$$[[[(x_1^1, y_1^1), \dots, (x_1^n, y_1^n)], [t_1, \dots, t_n]], [[(x_2^1, y_2^1), \dots, (x_2^m, y_2^m)], [t_1, \dots, t_m]], \dots] \quad (5.2.1)$$

Как видно из структуры файла входных данных, каждая траектория представлена двухэлементным массивом, где первый массив хранит координаты в виде массива пар (x_i^j, y_i^j) , а второй массив содержит временные метки для каждой пространственной точки в соответствующем порядке (t_i) . Извлеченные координаты x и y соответствуют пикселям на входных изображениях. В формуле 5.2.1 нижний индекс пространственных координат указывает на порядковый номер траектории, а верхний индекс представляет собой порядковый номер точки отслеживания. Внешний массив является массивом траекторий.

5.2.3 Обработка входных данных

Поскольку выбранный алгоритм ожидает получить траектории в виде многомерных векторов, исходные входные данные необходимо преобразовать к подходящему формату. Для этих целей был реализован пользовательский парсер, принимающий текстовый файл “txt” с траекториями в указанном ранее формате в качестве входных данных и в результате возвращающий список объектов *Trajectory*. Объект *Trajectory* состоит из нескольких объектов *TrajectoryPoint* со следующими атрибутами: x -координата, y -координата, время t . Исходный код метода синтаксического анализа приведен в Приложении А.

Входные данные содержат траектории различной длины и с различным пройденным расстоянием. Однако из-за неточности и ошибок в системе слежения некоторые траектории выглядят неправдоподобно и не имеют смысла. Одна из возможных причин этого - пропавший отслеживаемый объект или потеря системой слежения местоположения объекта. В отличие от случая потерянного местоположения, когда пропущенное местоположение может быть найдено с использованием моделей аппроксимации и регрессии, потерянный объект отслеживания не может быть впоследствии исправлен. По этой причине, чтобы улучшить качество результатов, было решено отфильтровать входные траектории и игнорировать короткие траектории с малым пройденным расстоянием, где пройденное расстояние рассчитывается как евклидово расстояние между первой и последней точками траектории. Для фильтрации параметров использовались следующие значения: $minLength = 10$ (*trajectory points*), $minTotalDist = 80$ (*pixels*). Результаты фильтрации с отображением удаленных и сохраненных траекторий показаны на рис. 5.2.3.

Как уже упоминалось ранее, текущая работа сфокусирована на задаче выявления двух типов нарушений: пространственных и пространственно-временных. Для выявления аномалий первой группы достаточно проанализировать пространственную информацию о траекториях. Обнаружение аномалий второй категории, которая формируется из траекторий ТС, движущихся с аномально низкой или высокой скоростью, требует учета



Рис. 5.2.3: Результаты фильтрации траекторий на примере данных с первого перекрестка

временной информации вместе с пространственной. По этой причине средняя постоянная скорость v рассчитывается для каждой из входных траекторий T в конце этапа парсинга данных как (5.2.2):

$$v_{avg}(T) = \frac{distance_{total}}{time_{total}}, \quad (5.2.2)$$

где $distance_{total}$ - итоговое пройденное расстояние между первой и последней точками траектории, а $time_{total}$ - прошедшее время. Общее расстояние может быть вычислено как сумма евклидовых расстояний между точками траектории на соседних кадрах. Поскольку известно, что кадры сняты с межкадровым интервалом 0,01 секунды, вычисление скорости может быть реализовано следующим образом (лист. 5.1):

Листинг 5.1: Подсчет скорости ТС

```

1 /**
2 * Calculates average speed for the trajectory in 'pixels per sec'
3 */
4 public double calcSpeed(Trajectory t) {
5     double dist = 0.0;
6     for (int i = 0; i < t.length() - 1; i++) {
7         dist += t.get(i).distanceTo(t.get(i + 1));
8     }
9
10    double time = (t.get(length() - 1).getTime() - t.get(0).getTime()) *
11        interFrameTime;
12
13    double avgSpeed = dist / time;
14    return avgSpeed;
15
16 /**
17 * Calculates the Euclidean distance between two trajectory points
18 */

```

```

19 * @param this      first (current) trajectory point
20 * @param other     second trajectory point
21 * @return          Euclidean distance
22 */
23 public double distanceTo(TrajectoryPoint other) {
24     if (this == other) {
25         return 0;
26     }
27     double d = Math.pow(this.x - other.x, 2) + Math.pow(this.y - other.y,
28                 2);
29     return Math.sqrt(d);

```

5.2.4 Аппроксимация траекторий с использованием Полиномиальной Регрессии

Как обсуждалось ранее, полиномиальная регрессия будет использоваться для аппроксимации исходных траекторий. Были использованы готовые реализации классов-сущностей полинома¹ (необходим для дальнейшего анализа уравнений аппроксимации и поиска ключевых точек) и метода решения полиномиальных уравнений² из библиотеки Apache Commons Math 3.4.1. Для выполнения полиномиальной регрессии³ была выбрана реализация, предоставленная авторами Р. Седжвиком и К. Уэйном для языка программирования Java [45]. Все готовые к использованию реализации были пользовательскими вспомогательными методами.

Класс *PolynomialRegression* принимает в качестве входных данных ожидаемую степень аппроксимирующего полинома (d) и два массива данных из N точек, состоящих из действительных чисел: массив независимых переменных ($double[] t$), данные о времени в данном случае, и массив зависимых переменных ($double[] x, double[] y$), пространственные x - или y -координаты. Затем он выполняет полиномиальную регрессию для входного набора из N точек данных (t_i, x_i) или (t_i, y_i) и пытается подобрать многочлен $x = \beta_0 + \beta_1 t + \beta_2 t^2 + \dots \beta_d t^d$, где β_i - коэффициенты регрессии, минимизирующие сумму квадратов ошибок модели полиномиальной регрессии. Поиск наилучшего решения для полиномиальных параметров основан на методе наименьших квадратов [42].

Для достижения лучшей аппроксимации оценка результатов полиномиальной регрессии выполнялась с использованием коэффициента детерминации, обозначаемого как R^2

¹Реализация полинома <https://javadoc.io/doc/org.apache.commons/commons-math3/3.4.1/org/apache/commons/math3/analysis/polynomials/PolynomialFunction.html>

²Реализация метода решения полиномиальных функций <https://www.javadoc.io/doc/org.apache.commons/commons-math3/3.4.1/org/apache/commons/math3/analysis/solvers/LaguerreSolver.html>

³Реализация полиномиальной регрессии <https://algs4.cs.princeton.edu/14analysis/PolynomialRegression.java>

(также известного как *R-squared*, *R*-оценка, *коэффициент регрессии Пирсона*) [46]. R^2 измеряет долю дисперсии зависимой переменной, которая может быть объяснена регрессионной моделью с заданными параметрами и является предсказуемой из независимой (объясняющей) переменной, и вычисляется следующим образом:

$$R^2 = 1 - \frac{SSE}{TSS} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2} \quad (5.2.3)$$

где *SSE* (Sum of Squares due to Error, сумма квадратов остатков, или ошибок, регрессии) рассчитывается как сумма квадратов разностей между фактическими значениями y_i и прогнозируемыми (расчетными) значениями зависимых переменных \hat{y}_i , и *TSS* (Total Sum of Squares, общая сумма квадратов) рассчитывается как сумма квадратов отклонений фактического значения y_i от среднего значения \bar{y} .

Коэффициент R^2 может принимать значения между $[0, 1]$, где близкое к 1 значение указывает на то, что существует сильная зависимость между выбранными параметрами и модель (в данном случае полиномиальное уравнение) отлично предсказывает данные [47]. Модели со значением коэффициента выше 0,8 принято считать адекватными, значение 1 свидетельствует о наличии функциональной зависимости между переменными.

В этой работе полиномиальная регрессия была выполнена для всех исходных траекторий (Приложение В). Полученные регрессионные модели сравнивались по показателям R^2 , результаты анализировались относительно формы и скорости траекторий. Демонстрация, сравнение и обсуждение полученных результатов представлены в следующей главе.

5.2.5 Выбор ключевых точек в аппроксимированных траекториях

Главной целью и причиной использования аппроксимированных траекторий вместо исходных для дальнейшего анализа было уменьшение сложности подсчета LCSS метрики. По этой причине длина траекторий должна быть уменьшена путем выбора нескольких ключевых репрезентативных точек из аппроксимированных траекторий путем анализа полиномов аппроксимации.

Из математики известно, что критическими точкам полинома $f(t)$ являются точки, где полиномиальная функция не дифференцируема или производная в этой точке равна нулю (стационарные точки). Стационарные точки, включая локальные минимумы и максимумы, восходящие и нисходящие точки перегиба можно найти путем анализа производных функций. Стационарные точки являются решениями уравнения, в котором первая производная функции приравнена нулю: $f'(t) = 0$. Точки перегиба могут быть найдены путем дальнейшего анализа второй производной: они соответствуют решениям уравнения $f''(t) = 0$. Для решения полиномиальных уравнений использовались готовые реализации методов решения полиномиальных уравнений из библиотеки Apache Commons Math: *LaguerreSolver*, *BisectionSolver* со следующими входными параметрами:

- $maxItem = 30000$ – задает максимально допустимое количество итераций в ходе поиска решения полиномиального уравнения,
- $min = firstTimePoint$ – определяет нижнюю границу допустимых значений решения;
- $max = lastTimePoint$ – определяет верхнюю границу допустимых значений решения;
- $startValue = min + 1$ – задает начальную точку для метода решения уравнения, чтобы начать поиск решения.

Программа решения уравнений вызывалась для полиномиальных функций, полученных как первая и вторая производные, взятые из полиномов для X - и Y -координат. Вызов метода решения приведен в лист. 5.2:

Листинг 5.2: Вызов метода решения полиномиальных уравнений

```
1 BaseAbstractUnivariateSolver bisectionSolver = new BisectionSolver();
2 BaseAbstractUnivariateSolver laguerreSolver = new LaguerreSolver();
3
4 for (Polynomial derivativeFunc :
5     List.of(derivativeFuncX1, derivativeFuncX2,
6             derivativeFuncY1, derivativeFuncY2)) {
7
8     for (BaseAbstractUnivariateSolver solver :
9         List.of(bisectionSolver, laguerreSolver)) {
10        solver.solve(30000, derivativeFunc,
11                     currentTr.getTrajectoryPoints().stream()
12                         .mapToInt(TrajectoryPoint::getTime)
13                         .min().getAsInt(),
14                     currentTr.getTrajectoryPoints().stream()
15                         .mapToInt(TrajectoryPoint::getTime)
16                         .max().getAsInt(),
17                     currentTr.getTrajectoryPoints().stream()
18                         .mapToInt(TrajectoryPoint::getTime)
19                         .min().getAsInt() + 1);
20    }
21 }
```

Решения, найденные двумя методами, объединяются: остаются только точки, относящиеся к разным точкам времени.

В случае анализа траекторий точки перегиба очень значимы и репрезентативны, поскольку они несут важную информацию о форме траектории: ключевые точки обозначают места основных поворотов или изменения траектории.

Однако критические точки не всегда могут быть найдены из-за вычислительных ограничений полиномиальных функций высокого порядка. Следовательно, критических точек, идентифицированных таким образом, недостаточно, чтобы полностью описать поведение исходной траектории и использовать для дальнейшего анализа, поскольку они не предоставляют всей информации о границах формы траектории и не всегда могут отобразить все повороты. По этой причине было решено добавить граничные для траектории ключевые точки, взяв отдельно минимальные и максимальные координаты X и Y и вычислив соответствующие точки траектории с использованием соответствующей регрессионной модели (лист. 5.3).

Листинг 5.3: Вычисление граничных ключевых точек траектории

```
1 // an input image with a resolution 1280*720 -> pixels for X in [0,
2   1280], pixels for Y in [0, 720]
3 double minX = 1280, maxX = 0, minY = 720, maxY = 0;
4 int tForMinX = 0, tForMaxX = 0, tForMinY = 0, tForMaxY = 0;
5 for (int time : currentTr.getTrajectoryPoints().stream()
6   .mapToInt(TrajectoryPoint::getTime)
7   .boxed().collect(toList())) {
8   double predictedX = currentTr.getRegressionX()
9     .predict(time);
10  double predictedY = currentTr.getRegressionY()
11    .predict(time);
12  if (predictedX < minX) {
13    minX = predictedX;
14    tForMinX = time;
15  }
16  if (predictedX > maxX) {
17    maxX = predictedX;
18    tForMaxX = time;
19  }
20  if (predictedY < minY) {
21    minY = predictedY;
22    tForMinY = time;
23  }
24  if (predictedY > maxY) {
25    maxY = predictedY;
26    tForMaxY = time;
27  }
28}
29 for (int tt : List.of(tForMinX,tForMaxX,tForMinY,tForMaxY)) {
30   currentTr.addKeyPoint(new TrajectoryPoint(
31     (int) Math.round(currentTr.getRegressionX().predict(tt)),
```

```
32     (int) Math.round(currentTr.getRegressionY() .predict (tt)) ,  
33     tt) ) ;  
34 }
```

Результаты аппроксимации полиномиальной регрессией и вычисления ключевых точек представлены на рисунке 5.2.4. Точки траектории, относящиеся к исходным данным траектории, изображены с использованием красного цвета, тогда как синие точки траектории соответствуют точкам, полученным с использованием функций полиномиального приближения для тех же временных точек. Ключевые точки каждой траектории выделены жирными синими квадратными точками. Видно, что функция аппроксимации близка к исходной функции траектории, для некоторых траекторий приближение дает те же координаты, что и в исходных данных (для траекторий с полиномами аппроксимации, имеющими $R^2 \approx 1.0$). Также можно заметить, что ключевые точки точно придерживаются линии аппроксимации и правильно описывают траекторию, следовательно, они могут использоваться вместо исходных точек траектории для упрощения дальнейшего анализа траекторий.



Рис. 5.2.4: Результаты полиномиальной регрессии с обозначением ключевых точек

5.3 Анализ траекторий

5.3.1 Вычисление метрики схожести траекторий

Как упоминалось ранее, LCSS метрика будет использоваться в качестве метрики для измерения сходства между траекториями. Следовательно, расстояние LCSS (метрика различия) будет рассчитываться на основе метрики сходства LCSS в соответствии с вышеупомянутыми формулами. Стоит отметить, что LCSS метрика является симметричной и для пары траекторий может быть вычислена только один раз [30].

Несмотря на то что реализация LCSS метрики присутствует в пакете R [32], представленная реализация не позволяет использование динамических, адаптивных параметров δ и ε . По этой причине в рамках реализации фреймворка была написан пользовательская реализация метода подсчета LCSS метрики, представленный в лист. 5.4.

Листинг 5.4: Вычисление LCSS

```

1 /**
2 * Calculates LCSS for two input trajectories
3 *
4 * @param t1      first trajectory
5 * @param t2      second trajectory
6 * @param δ       δ parameter: how far we can look in time to match a given
7 *               point from one T to a point in another T
8 * @param ε       ε parameter: the size of proximity in which to look for
9 *               matches
10 * @return        LCSS for t1 and t2
11 */
12
13
14 private Double calcLCSS(Trajectory t1, Trajectory t2, Double δ, Double ε)
15 {
16     int m = t1.length();
17     int n = t2.length();
18
19     if (m == 0 || n == 0) {
20         return 0.0;
21     } else {
22
23         if (abs(t1.get(m - 1).getX() - t2.get(n - 1).getX()) < ε
24             && abs(t1.get(m - 1).getY() - t2.get(n - 1).getY()) < ε
25             && abs(m - n) <= δ) {
26             return 1 + calcLCSS(head(t1), head(t2), δ, ε);
27         } else {
28             return max(
29                 calcLCSS(head(t1), t2, δ, ε),
30                 calcLCSS(t1, head(t2), δ, ε));
31         }
32     }
33 }
```

```

26     }
27 }
28 /**
29 * Calculates shortened trajectory by excluding last trajectory point
30 *
31 * @param t trajectory
32 * @return trajectory without last trajectory point
33 */
34 */
35 private Trajectory head(Trajectory t) {
36     Trajectory tClone = t.clone();
37     tClone.getTrajectoryPoints().remove(tClone.length() - 1);
38     return tClone;
39 }
```

5.3.2 Кластеризация

Поскольку не было найдено подходящей реализации иерархической кластеризации для траекторий с использованием расстояния LCSS и возможностью обрабатывать адаптивные значения параметров, кластеризация, а также вычисление метрики LCSS, были написаны с нуля. Реализация была написана на основе алгоритма 1, представленного выше, в общих чертах описывающего алгоритм. Исходный код кластеризации приведен в Приложении С.

Кластеризация выполняется итеративно методом последовательного объединения двух ближайших кластеров в один с последующим пересчетом матрицы подобия (близости) кластеров. Метод кластеризации принимает в качестве входных данных параметр *OUTPUT_CLUSTERS_COUNT*, который определяет ожидаемое итоговое количество кластеров. Если полученное значение пустое (*null*), оно будет рассматриваться как 1, и кластеризация будет выполняться до тех пор, пока все кластеры не будут объединены в один в соответствии с базовым алгоритмом агломеративной иерархической кластеризации, или пока дальнейшее объединение кластеров будет невозможно.

Валидация кластеров

Как было описано ранее, валидация кластеров будет проводиться с использованием индекса Данна (DI). В лист. 5.5 представлена реализация подсчета DI значения для итоговых кластеров.

Листинг 5.5: Вычисление DI

```

1 /**
2 * Dunn's Validity Index (DI) = dist_min / diam_max
3 * dist_min = min inter-cluster distance (minimum distance between two
4 * clusters;
```

```

4  * single-linkage -> min distance between two trajectories from two
5   clusters)
6  */
7 private void validateClusters() {
8     // sort trajectories inside each cluster according to trajectory ID
9     clusters.forEach(cluster ->
10         cluster.getTrajectories().sort(Comparator.comparing(Trajectory::
11             getId)));
12
13     double minDist = Double.MAX_VALUE;
14     for (int i = 0; i < clusters.size(); i++) {
15         for (int j = i + 1; j < clusters.size(); j++) {
16             if (clustLCSSDistances[clusters.get(i).getId()][clusters.get(
17                 j).getId()] < minDist)
18                 minDist = clustLCSSDistances[clusters.get(i).getId()][
19                     clusters.get(j).getId()];
20         }
21     }
22
23     double maxDiam = clusters.stream().mapToDouble(cluster -> {
24         double maxDist = 0;
25         for (int i = 0; i < cluster.getTrajectories().size(); i++) {
26             for (int j = i + 1; j < cluster.getTrajectories().size(); j
27                 ++ ) {
28                 if (trajLCSSDistances[cluster.getTrajectories().get(i).
29                     getId()][cluster.getTrajectories().get(j).getId()] > maxDist)
30                     maxDist = trajLCSSDistances[cluster.getTrajectories()
31                         ().get(i).getId()][cluster.getTrajectories().get(j).getId()];
32             }
33         }
34         return maxDist;
35     }).max().getAsDouble();
36
37     double DI = minDist / maxDiam;
38     LOGGER.info(String.format("DI = %.2f", DI));
39 }

```

Как видно из представленного листинга кода, сначала считается минимальная дистанция между всеми парами кластеров (*minDist*), затем выясняется максимальный диаметр среди диаметров всех кластеров (*maxDiam*). Поскольку реализация основана на предположении, что траектории в кластерах появляются в порядке упорядочивания идентификаторов по возрастанию и внутренний цикл начинается с индекса, на 1 большего

внешнего индекса кластера или траектории, перед подсчетами траектории кластеров будут упорядочены для упрощения дальнейших расчетов.

5.3.3 Моделирование кластеров

Clusters modeling implementation

6 Результаты

В этой главе описываются оценочные тесты, выполненные для проверки предложенного подхода на основе разработанного фреймворка. Следующие разделы предназначены для предоставления информации о подготовке исходных данных, представления и обсуждения полученных результатов. Сначала будет рассмотрена аппроксимация траекторий с использованием полиномиальной регрессии. Далее будет проведена проверка точности результатирующих кластеров точность с использованием вышеупомянутого индекса DI. Глава завершается обсуждением выходных результатов этапа классификации и демонстрацией результатов проверки качества обнаружения аномалий.

6.1 Оценка точности

В этом разделе подробно описываются подготовка и результаты оценочных тестов, выполненных для проверки точности, достоверности полученных результатов.

6.1.1 Результаты аппроксимации траекторий

Для того чтобы принять решение о степени полиномов для аппроксимации, было проведено несколько экспериментов. Были предприняты попытки аппроксимировать исходные траектории полиномами 3-ей, 4-ой, 5-ой степеней соответственно. В следующей таблице приведены минимальные и средние значения коэффициента детерминации R^2 для каждого из экспериментов (табл. 6.1.1).

В этом параграфе будет приведено обсуждение результатов экспериментов на примере траекторий с первого перекрестка (*1.txt*) до и после фильтрации траекторий. Из табл. 6.1.1 видно, что средние значения коэффициента R^2 приемлемы для всех экспериментов. Однако минимальные значения R^2 , которые равны 0,66 и 0,466 для первого эксперимента с полиномами только 3-ей степени для неотфильтрованных траекторий являются неудовлетворительными, что означает, что модель может корректно предсказать только половину всех точек траекторий. Фильтрация и игнорирование коротких траекторий с малым итоговым пространственным смещением значительно улучшили результаты. Однако результаты для аппроксимации с использованием полиномов 3-ей степени все еще неудовлетворительны. Это может повлиять на последующий анализ и ухудшить дальнейший анализ. По этой причине было выполнено приближение с использованием полиномов различных степеней одновременно следующим образом:

Таблица 6.1.1: Значения коэффициента R^2 для полиномов различных степеней

Degrees of polynomials	R^2 score			
	X		Y	
	min	avg	min	avg
1.txt (before filtering)				
{3}	0.66	0.994	0.466	0.989
{3, 4}	0.897	0.998	0.823	0.994
{3, 4, 5}	0.949	0.998	0.864	0.995
1.txt				
{3}	0.689	0.997	0.777	0.995
{3, 4}	0.942	0.999	0.872	0.997
{3, 4, 5}	0.98	0.999	0.88	0.997
2.txt				
{3, 4}	0.992	0.9997	0.832	0.996
3.txt				
{3, 4}	0.815	0.995	0.867	0.996
4.txt				
{3, 4}	0.879	0.995	0.722	0.993

- сначала выполнить аппроксимацию, используя низшую степень полинома в качестве отправной точки,
- сравнить полученные значения коэффициента детерминации R^2 с заранее заданным граничным значением (0,98 в данном случае); если полученное значение меньше предопределенного порога, увеличить ожидаемую степень полинома и выполнить полиномиальную регрессию снова,
- продолжать увеличивать степень полинома до получения приемлемых значений R^2 или до достижения заданного ограничения максимальной допустимой степени полинома (5 в данном случае).

Аппроксимация полиномами 3-ей и 4-ой степеней в совокупности значительно улучшила как минимальные, так и средние значения R^2 . Однако добавление полинома 5-ой



Рис. 6.1.1: Траектории, аппроксимированные полиномами 4-ой степени

степени не оказалось существенного влияния на результаты и улучшило средний коэффициент только на 0,01 по сравнению с предыдущим экспериментом. В связи с этим было решено сосредоточиться на приближении с использованием полиномиальных функций 3-ей и 4-ой степеней и провести эксперименты на траекториях с других перекрестков (*2.txt*, *3.txt*, *4.txt*).

Для упрощения повествования траектории будут классифицированы на две группы в зависимости от степени полиномов, использованных для аппроксимации: первая группа включает в себя траектории, аппроксимированные полиномиальными функциями 3-ей степени, в то время как вторая группа состоит из траекторий, аппроксимируемых полиномами 4-ой степени. Траектории обеих групп были проанализированы с точки зрения формы и средней скорости. На рис. 6.1.1 показана вторая группа траекторий, а в табл. 6.1.2 указаны представлены минимальная, средняя и максимальная скорости траекторий для обеих групп.

Далее траектории, аппроксимированные полиномами 3-ей 4-ой степеней будут обозначаться первой и второй группами траекторий соответственно. Можно заметить, что траектории из разных групп имеют очень разные скорости. Первая группа включает в себя траектории с гораздо более высокими скоростями, особенно выделяется, что максимальная скорость для второй группы почти равна средней скорости для первой группы, а средняя скорость для первой группы почти в четыре раза больше, чем для второй группы для случая неотфильтрованных траекторий. После исключения коротких траекторий из рассмотрения результаты стали более плавными, однако те же наблюдения и выводы имеют место быть. Также на рис. 6.1.1 видно, что полиномиальные функции 4-го порядка использовались для аппроксимации траекторий более сложной формы или траекторий с плотно расположенными точками траектории.

Такая тенденция в основном различима при рассмотрении траекторий с первого перекрестка. Данные с первого перекрестка могут считаться репрезентативным набором траекторий, поскольку он имеет наибольшее количество экземпляров данных. Также стоит отметить, что второй набор данных траекторий (*2.txt*) имеет только 10 траекторий в первой группе, поэтому анализ скорости траекторий не может быть очень точным.

Таблица 6.1.2: Сравнение минимальной, средней, максимальной скоростей ТС

Degree of a polynomial	Speed (<i>pixels per sec</i>)		
	min	avg	max
1.txt (before filtering)			
{3}	18.555	335.365	1721.499
{4}	1.206	72.34	374.396
1.txt			
{3}	61.814	372.435	909.121
{4}	26.603	229.053	602.773
2.txt			
{3}	85.705	494.016	1107.96
{4}	183.087	613.865	900.737
3.txt			
{3}	13.65	301.481	1012.748
{4}	29.26	206.119	764.25
4.txt			
{3}	43.01	269.074	872.33
{4}	22.92	163.431	708.154

Выводы

Таким образом, из вышеизложенных результатов следует, что полиномиальные функции высшего порядка предпочтительнее для аппроксимации траекторий следующих групп:

- траектории медленно движущихся или неактивных ТС (включая траектории ТС, ожидающих на перекрестках на светофоре), которые продемонстрированы на 6.1.1a;
- траектории ТС, движущихся с непостоянной скоростью или ускорением на некоторых участках траектории (могут быть представлены неравномерным распределением точек траектории, где плотные области точек траектории сигнализируют об ускорении в течение этих временных интервалов; как результат, полиномы низкого

порядка не могут описать такую сложную зависимость между пространственной координатой и временем);

- траектории сложной формы (резкие повороты, улитки Паскаля), особенно различимо на рис. 6.1.1b-c.

6.1.2 Результаты кластеризации траекторий

Было проведено 2 эксперимента на одном и том же наборе данных (*2.txt*) с разным числом итоговых кластеров: 8 и 9. Согласно результатам экспериментов, объединение до 8 кластеров продемонстрировало более точное разбиение данных и более высокие значения DI. Результаты выполнения кластеризации на примере данных с первого перекрестка представлены на рис. 6.1.2, кластеры данных обозначены различными цветами. На рис. 6.1.2a видно, что возможно и желательно дальнейшее объединение кластеров, представленных желтым и зеленым цветами. Рис. 6.1.2b демонстрирует результат продолжения кластеризации (в данном случае объединения кластеров), в котором два указанных кластера объединены в один на основании малого значения расстояния между ними (высокой степени схожести).

Валидация полученных кластеров была проведена с использованием метрики DI, были получены следующие значения:

- 9 кластеров – $DI = 0.94$,
- 8 кластеров – $DI = 0.95$.

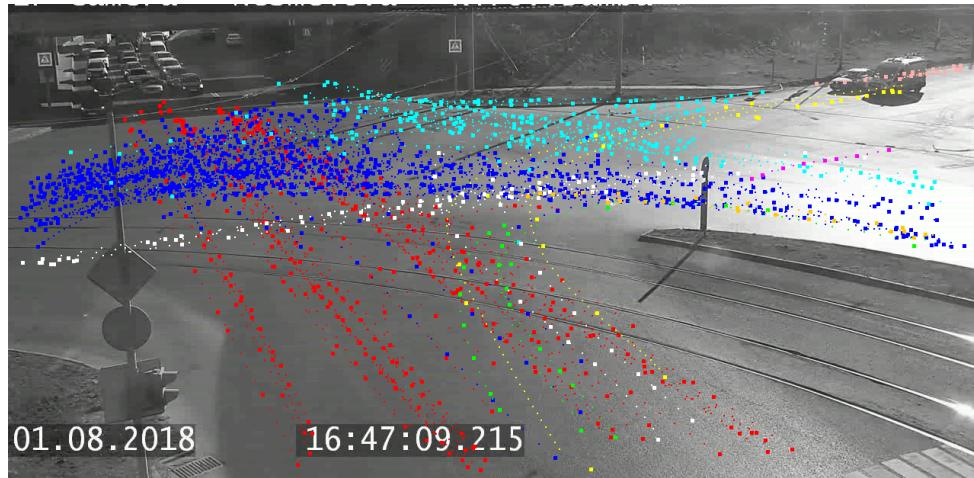
Однако, несмотря на высокое значение DI, можно заметить, что результат кластеризации содержит ошибки: синие ключевые точки, соответствующие синему кластеру траекторий, накладываются на траектории красного кластера. Возможной причиной такого поведения является высокая степень сходства между этими траекториями и другими траекториями синего кластера в левой верхней половине рисунка: эта область представлена густым скоплением точек синего цвета.

При подсчете метрики LCSS для кластеризации были реализованы адаптивные параметры δ и ε . В качестве функциональной зависимости между параметров ε и расстоянием между точкой траектории и камеры были выбраны следующие отношения:

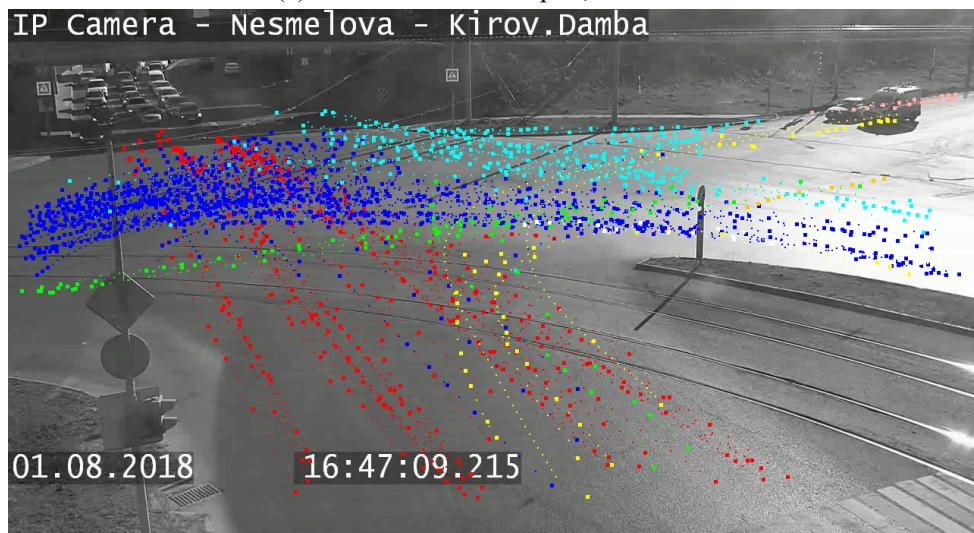
$$\delta = coeff * \min(t1.length(), t2.length()) \quad (6.1.1)$$

$$\varepsilon_x = coeff * (maxX - minX) / distanceToCamera \quad (6.1.2)$$

$$\varepsilon_y = coeff * (maxY - minY) / distanceToCamera \quad (6.1.3)$$



(a) 9 итоговых кластеров, DI = 0.94



(b) 8 итоговых кластеров, DI = 0.95

Рис. 6.1.2: Результаты кластеризации на примере траекторий со второго перекрестка

где $distanceToCamera$ вычисляется как Евклидово расстояние между данной точкой траектории и точкой местоположения камеры на текущем перекрестке, а $coeff$ - коэффициент, значение которого будет подобрано экспериментальным путем. В данном случае наилучшие результаты были получены для значения коэффициента $coeff = 20.0$ и статистика средних и минимальных значений адаптивных ε для каждой из осей представлены ниже:

ε

$X :$

max = 351.397 (pixels)

min = 23.248 (pixels)

$Y :$

max = 151.252 (pixels)

min = 10.004 (pixels)

Выводы

Из вышепредставленного следует, что, по результатам как количественной оценки метрикой DI, так и визуального теста, кластеризация данных со второго перекрестка работает точнее до распределения данных по 8 кластерам.

7 ЗАКЛЮЧЕНИЕ

В этой работе был предложен и проанализирован подход для идентификации траекторных аномалий в неопределеных пространственно-временных данных. Для оценки и реализации подхода была разработан фреймворк, решающий поставленную задачу. Исходный код реализации доступен в репозитории GitHub [48]. Для решения задачи обнаружения аномалий в качестве основы использовался кластерный подход, а именно алгоритм агломеративной иерархической кластеризации. Для вычисления меры сходства и различия между траекториями и кластерами была выбрана и реализована метрика расстояния LCSS. Однако, как было упомянуто в предыдущих главах, вычисление расстояния LCSS по классическому алгоритму становится невозможным для длинных траекторий. По этой причине была выполнена аппроксимация входных траекторий с использованием полиномиальной регрессии. Согласно результатам оценки, наилучшая точность аппроксимации достигается при совместном использовании полиномиальных функций 3-ей и 4-ой степеней. Таким образом, кластеризация была выполнена на отфильтрованном наборе приближенных входных траекторий с использованием отобранных ключевых точек для каждой из них. Точность выполненной кластеризации была оценена с использованием индекса DI и равна 0,95, что говорит о качественном разделении на четкие различимые кластеры.

В результате данной работы можно сделать следующие выводы:

- аппроксимация коротких траекторий с непостоянной скоростью требует использования полиномиальных функций более высоких степеней,
- несмотря на то что LCSS метрика расстояния позволяет траекториям быть разной длины, ее вычисление становится чрезвычайно трудоемким и времязатратным для траекторий с более чем 11-12 точками траекторий,
- аппроксимация траекторий с использованием полиномиальной регрессии дает точные результаты, поскольку заранее известно, что существует функциональная зависимость между пространственными координатами траектории и временным параметром (согласно принципам физики, скорости, ускорения и т.д.).

Дальнейшее развитие

Предложенный подход и разработанный фреймворк спроектированы в манере обучения оффлайн что означает, что модели поведения нормальных и аномальных траекторий

изучаются и запоминаются заранее и не обновляются впоследствии. Последующие исследования могут включать изучение возможности обновления базы данных нормальных траекторий по мере поступления новых данных, чтобы сделать подход и сам фреймворк адаптирующимся к реальным актуальным данным транспортного трафика.

Литература

- [1] Y. Djenouri, A. Belhadi, J. C. Lin, D. Djenouri, and A. Cano. A Survey on Urban Traffic Anomalies Detection Algorithms. *IEEE Access*, 7:12192–12205, 2019.
- [2] C. Koetsier, S. Busch, and M. Sester. Trajectory Extraction for Analysis of Unsafe Driving Behaviour. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42(2/W13):1573–1578, June 2019.
- [3] R. Ranjith, J. J. Athanasiou, and V. Vaidehi. Anomaly Detection using DBSCAN Clustering Technique for Traffic Video Surveillance. In *2015 7th International Conference on Advanced Computing (ICoAC)*, pages 1–6, December 2015.
- [4] F. Mehboob, M. Abbas, R. Jiang, A. Rauf, S. A. Khan, and S. Rehman. Trajectory Based Vehicle Counting and Anomalous Event Visualization in Smart Cities. *Cluster Computing*, 21:443–452, March 2018.
- [5] S. A. Ahmed, D. P. Dogra, S. Kar, and P. P. Roy. Trajectory-Based Surveillance Analysis: A Survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(7):1985–1997, 2019.
- [6] F. Meng, G. Yuan, S. Lv, Z. Wang, and S. Xia. An overview on trajectory outlier detection. *Artificial Intelligence Review*, February 2018.
- [7] G. Atluri, A. Karpatne, and V. Kumar. Spatio-Temporal Data Mining: A Survey of Problems and Methods. *ACM Computing Surveys*, 51(4), 2017.
- [8] F. Tung, J. S. Zelek, and D. A. Clausi. Goal-Based Trajectory Analysis for Unusual Behaviour Detection in Intelligent Surveillance. *Image Vision Comput.*, 29(4):230–240, March 2011.
- [9] Y. Li, J. Bailey, L. Kulik, and J. Pei. Mining Probabilistic Frequent Spatio-Temporal Sequential Patterns with Gap Constraints from Uncertain Databases. In *2013 IEEE 13th International Conference on Data Mining*, pages 448–457, December 2013.
- [10] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A Review of Moving Object Trajectory Clustering Algorithms. *Artificial Intelligence Review*, 47(1):123–144, January 2017.

- [11] A. d’Acierno, A. Saggese, and M. Vento. Designing Huge Repositories of Moving Vehicles Trajectories for Efficient Extraction of Semantic Data. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):2038–2049, August 2015.
- [12] V. Bogorny V. C. Fontes. Discovering Semantic Spatial and Spatio-Temporal Outliers from Moving Object Trajectories. *ArXiv*, abs/1303.5132, 2013.
- [13] H. Liu, X. Li, J. Li, and S. Zhang. Efficient Outlier Detection for High-Dimensional Data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(12):2451–2461, December 2018.
- [14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3), July 2009.
- [15] F. E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11(1):1–21, February 1969.
- [16] S. K. Kumaran, D. P. Dogra, and P. P. Roy. Anomaly Detection in Road Traffic Using Visual Surveillance: A Survey. *ArXiv: Computer Vision and Pattern Recognition*, January 2019.
- [17] K. Malik, H. Sadawarti, and G. Kalra. Comparative analysis of outlier detection techniques. *International Journal of Computer Applications*, 97:12–21, July 2014.
- [18] D. Kumar, J. Bezdek, S. Rajasegarar, C. Leckie, and M. Palaniswami. A Visual-Numeric Approach to Clustering and Anomaly Detection for Trajectory Data. *The Visual Computer*, 33(3):265–281, March 2017.
- [19] S. W. T. T. Liu, H. Y. T. Ngan, M. K. Ng, and S. J. Simske. Accumulated Relative Density Outlier Detection For Large Scale Traffic Data. In *Electronic Imaging*, volume 9, pages 1–10, 2018.
- [20] P. Batapati, D. Tran, W. Sheng, M. Liu, and R. Zeng. Video Analysis for Traffic Anomaly Detection using Support Vector Machines. In *Proceedings of the 11th World Congress on Intelligent Control and Automation (WCICA)*, pages 5500–5505, March 2014.
- [21] C. Piciarelli, C. Micheloni, and G. L. Foresti. Trajectory-Based Anomalous Event Detection. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1544–1554, December 2008.
- [22] H.-L. Nguyen, Y.-K. Woon, and W. K. Ng. A Survey on Data Stream Clustering and Classification. *Knowledge and Information Systems*, 45:535–569, December 2014.

- [23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.
- [24] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.
- [25] G. F. Tzortzis and A. C. Likas. The Global Kernel k -Means Algorithm for Clustering in Feature Space. *IEEE Transactions on Neural Networks*, 20(7):1181–1194, July 2009.
- [26] T. Bock. DisplayR Blog. What is Hierarchical Clustering? <https://www.displayr.com/what-is-hierarchical-clustering/>. Internet Resource, Accessed: 2020-07-05.
- [27] C. R. Patlolla. Understanding the Concept of Hierarchical Clustering Technique. <https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec>, December 2018. Internet Resource, Accessed: 2020-07-05.
- [28] N. B. Ghrab, E. Fendri, and M. Hammami. Abnormal Events Detection Based on Trajectory Clustering. In *2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGIV)*, pages 301–306, 2016.
- [29] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing. Discovering Spatio-Temporal Causal Interactions in Traffic Data Streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1010–1018, New York, NY, USA, 2011. Association for Computing Machinery.
- [30] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering Similar Multidimensional Trajectories. In *Proceedings 18th International Conference on Data Engineering*, pages 673–684, February 2002.
- [31] T. Eiter and H. Mannila. Computing Discrete Fréchet Distance *. In *Technical report CD-TR 94/64, Technische Universität Wien*, 1994.
- [32] K. Toohey. R Package Documentation. Similarity Measures. LCSS. <https://rdrr.io/cran/SimilarityMeasures/man/LCSS.html>, May 2019. Internet Resource, Accessed: 2020-06-30.
- [33] K. Toohey and M. Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, May 2015.

- [34] M. Vlachos, M. Hadjieleftheriou, D. Gunopoulos, and E. Keogh. Indexing multidimensional time-series. *The VLDB Journal*, 15:1–20, July 2006.
- [35] Zhang Zhang, Kaiqi Huang, and Tieniu Tan. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. volume 3, pages 1135–1138, January 2006.
- [36] B. T. Morris and M. M. Trivedi. A survey of vision-based trajectory learning and analysis for surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8):1114–1127, August 2008.
- [37] D. Dey. Dunn index and DB index – Cluster Validity Indices. <https://www.geeksforgeeks.org/dunn-index-and-db-index-cluster-validity-indices-set-1/>. Internet Resource, Accessed: 2020-07-07.
- [38] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.
- [39] M. Pathak. DataCamp. Hierarchical Clustering in R. <https://www.datacamp.com/community/tutorials/hierarchical-clustering-R>, July 2018. Internet Resource, Accessed: 2020-06-30.
- [40] Z. Ansari, M.F. Azeem, W. Ahmed, and A. Babu. Quantitative evaluation of performance and validity indices for clustering the web navigational sessions. *World of Computer Science and Information Technology (WCSIT) Journal*, 1(5):217–226, 2011.
- [41] B. Desgraupes. Clustering indices. 2016.
- [42] I. Hadi and M. Sabah. Behavior formula extraction for object trajectory using curve fitting method. *International Journal of Computer Applications*, 104:28–37, 10 2014.
- [43] A. Pant. Introduction to Linear Regression and Polynomial Regression. <https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb>, January 2019. Internet Resource, Accessed: 2020-07-15.
- [44] E.-H. Choi and National Highway Traffic Safety Administration. Crash Factors in Intersection-Related Crashes: An On-Scene Perspective. In *NHTSA Technical Report DOT HS 811 366*, September 2010.
- [45] R. Sedgewick and K. Wayne. Polynomial Implementation. <https://algs4.cs.princeton.edu/14analysis/PolynomialRegression.java.html>. Internet Resource, Accessed: 2020-07-11.

- [46] Y. A. W. Shardt. *Statistics for Chemical and Process Engineers: A Modern Approach*, chapter 3.2 Regression Models, pages 90–104. Springer International Publishing, Cham, Switzerland, 2015.
- [47] Minitab Blog Editor. Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit? <https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit>, May 2013. Internet Resource, Accessed: 2020-07-17.
- [48] Framework for Identification of Trajectory Anomalies in Uncertain Spatiotemporal Data. <https://github.com/aygulmardanova/mt-anomalies>. Internet Resource, Accessed: 2020-08-02.

Список иллюстраций

4.1.1 Основные этапы работы фреймворка	32
4.2.1 Схема двухэтапного предложенного подхода	32
4.2.2 Архитектура фреймворка	34
4.4.1 Принцип выбора адаптивного значения ε	38
4.5.1 Объяснение индекса DI	40
5.2.1 Примеры изображений с видеокамер на перекрестках 1-4	45
5.2.2 Пример работы системы слежения на данных с первого перекрестка	45
5.2.3 Результаты фильтрации траекторий на примере данных с первого перекрестка	47
5.2.4 Результаты полиномиальной регрессии с обозначением ключевых точек	53
6.1.1 Траектории, аппроксимированные полиномами 4-ой степени	60
6.1.2 Результаты кластеризации на примере траекторий со второго перекрестка	63

Список таблиц

6.1.1 Значения коэффициента R^2 для полиномов различных степеней	59
6.1.2 Сравнение минимальной, средней, максимальной скоростей ТС	61

Алгоритмы

1	Описание Агломеративной Иерархической кластеризации	34
2	Описание алгоритма LCSS метрики	37
3	Определение адаптивных параметров LCSS	38

Листинги

5.1	Подсчет скорости ТС	47
5.2	Вызов метода решения полиномиальных уравнений	50
5.3	Вычисление граничных ключевых точек траектории	51
5.4	Вычисление LCSS	54
5.5	Вычисление DI	55
7.1	Алгоритм парсинга входных траекторий	I
7.2	Инициирование полиномиальной регрессии	V
7.3	Реализация кластеризации	VI

ПРИЛОЖЕНИЕ

А. Алгоритм парсинга входных траекторий

Листинг 7.1: Алгоритм парсинга входных траекторий

```
1 package ru.griat.rcse.parsing;
2
3 import ru.griat.rcse.entity.Trajectory;
4 import ru.griat.rcse.entity.TrajectoryPoint;
5 import ru.griat.rcse.exception.TrajectoriesParserException;
6 import org.apache.commons.io.FilenameUtils;
7
8 import java.io.FileInputStream;
9 import java.io.IOException;
10 import java.io.InputStream;
11 import java.util.ArrayList;
12 import java.util.List;
13
14 /*
15 * Parser to parse input trajectories from text file
16 *
17 * stop symbols:
18 * if meet number - read until ']' or ',' or ')'
19 * [ - check for next, if [ - check for next, if [ - isX=true, if value -
20 * save x,
21 * */
22 public class TrajectoriesParser {
23
24     private int openingSqBracketNumber;
25
26     private boolean trajectoryStarted = false;
27     private boolean trajectoryCoordinatesStarted = false;
28     private int indexOfT;
29     private int indexOfTP;
30
31     private StringBuilder x;
32     private StringBuilder y;
33     private StringBuilder t;
34
35     private List<TrajectoryPoint> trajectoryPoints;
36     private List<Trajectory> trajectories;
37
38     public TrajectoriesParser() {
39         openingSqBracketNumber = 0;
```

```
39     indexOfT = 0;
40     indexOfTP = 0;
41
42     x = new StringBuilder();
43     y = new StringBuilder();
44     t = new StringBuilder();
45
46     trajectoryPoints = new ArrayList<>();
47     trajectories = new ArrayList<>();
48 }
49
50 /**
51 * Parses input 'txt'-file
52 *
53 * @param fileName full path to the input data file with trajectories
54 * @return list of extracted trajectories
55 */
56 public List<Trajectory> parseTxt(String fileName) throws IOException,
57     TrajectoriesParserException {
58
59     InputStream reader = new FileInputStream(FilenameUtils.normalize(
60         fileName));
61     int intch;
62     while ((intch = reader.read()) != -1) {
63         char nextChar = (char) intch;
64         while ((nextChar == ',' || nextChar == ' '))
65             nextChar = (char) reader.read();
66         while (nextChar == '[') {
67             increaseOpeningSqBracketsCount();
68             nextChar = (char) reader.read();
69         }
70         while (trajectoryCoordinatesStarted) {
71             if (nextChar == '(') {
72                 readCoordinates(reader);
73             }
74             nextChar = (char) reader.read();
75             if (nextChar == ']') {
76                 increaseClosingSqBracketsCount();
77             }
78         }
79         nextChar = (char) reader.read();
80         while ((nextChar == ',' || nextChar == ' '))
81             nextChar = (char) reader.read();
82         if (trajectoryStarted) {
83             if (nextChar == '[') {
```

```
82         increaseOpeningSqBracketsCount();
83         readTime(reader);
84     } else {
85         throw new TrajectoriesParserException("After coordinates array
86         with timestamps was expected");
87     }
88     finishProcessingTrajectory();
89 }
90
91 reader.close();
92 return trajectories;
93 }
94
95 private void processBracketsCount() {
96     if (openingSqBracketNumber == 1) {
97         trajectoryStarted = false;
98         trajectoryCoordinatesStarted = false;
99     }
100    if (openingSqBracketNumber == 2) {
101        trajectoryStarted = true;
102        trajectoryCoordinatesStarted = false;
103    }
104    if (openingSqBracketNumber == 3) {
105        trajectoryCoordinatesStarted = true;
106    }
107 }
108
109 /**
110 * Reads an x and y values from file after '(' and before next ')'
111 */
112 private void readCoordinates(InputStream reader) throws IOException {
113     char nextChar = (char) reader.read();
114     while (nextChar != ',') {
115         if (nextChar >= '0' && nextChar <= '9')
116             x.append(nextChar);
117         nextChar = (char) reader.read();
118     }
119     while (nextChar != ')') {
120         if (nextChar >= '0' && nextChar <= '9')
121             y.append(nextChar);
122         nextChar = (char) reader.read();
123     }
124     processTrajectoryPoint();
125 }
```

```
126 /**
127 * Reads time and saves it into already initialized trajectory by
128 updating trajectoryPoint at indexOfTP position in a current
129 trajectory
130 */
131 private void readTime(InputStream reader) throws IOException {
132     char nextChar = (char) reader.read();
133     while (nextChar != ']') {
134         while (nextChar != ',' && nextChar != ']') {
135             t.append(nextChar);
136             nextChar = (char) reader.read();
137         }
138         if (nextChar == ']') {
139             increaseClosingSqBracketsCount();
140             trajectoryPoints.get(indexOfTP).setTime(Integer.parseInt(t.
141                         toString().trim()));
142             indexOfTP++;
143             t = new StringBuilder();
144             nextChar = (char) reader.read();
145         }
146     }
147     private void increaseOpeningSqBracketsCount() {
148         openingSqBracketNumber++;
149         processBracketsCount();
150     }
151
152     private void increaseClosingSqBracketsCount() {
153         openingSqBracketNumber--;
154         processBracketsCount();
155     }
156
157 /**
158 * Adds parsed trajectory into an array of output trajectories and
159 prepares for the next input trajectory by resetting to 0 indexes and
160 buffers
161 */
162 private void finishProcessingTrajectory() {
163     trajectories.add(new Trajectory(indexOfT, trajectoryPoints));
164     trajectoryPoints = new ArrayList<>();
165     indexOfT++;
166     indexOfTP = 0;
167     trajectoryStarted = false;
```

```

166     increaseClosingSqBracketsCount();
167 }
168
169 /**
170 * Creates a new TrajectoryPoint with collected x and y
171 * Clear the buffer
172 */
173 private void processTrajectoryPoint() {
174     TrajectoryPoint point = new TrajectoryPoint(
175         Integer.parseInt(x.toString().trim()),
176         Integer.parseInt(y.toString().trim())
177     );
178     trajectoryPoints.add(point);
179
180     x = new StringBuilder();
181     y = new StringBuilder();
182 }
183
184 }
```

B. Инициирование Полиномиальной Регрессии

Листинг 7.2: Инициирование полиномиальной регрессии

```

1 // initialization
2 double[] t, x, y;
3 int degree = 3;
4 double thresholdR2 = 0.97;
5 double minR2forX = 1.0, minR2forY = 1.0;
6 int minR2forXid = -1, minR2forYid = -1;
7
8 Invocation of the polynomial regression for each trajectory
9 for (int tId = 0; tId < trajectories.size(); tId++) {
10     PolynomialRegression regressionX;
11     PolynomialRegression regressionY;
12
13     Trajectory currentTr = trajectories.get(tId);
14     t = currentTr.getTrajectoryPoints().stream()
15         .mapToDouble(TrajectoryPoint::getTime).toArray();
16     x = currentTr.getTrajectoryPoints().stream()
17         .mapToDouble(TrajectoryPoint::getX).toArray();
18     y = currentTr.getTrajectoryPoints().stream()
19         .mapToDouble(TrajectoryPoint::getY).toArray();
20     regressionX = new PolynomialRegression(t, x, degree);
21     regressionY = new PolynomialRegression(t, y, degree);
22 }
```

```

23 //      if regression results are not satisfactory (means that degree of
24 //      polynomial is not enough)
25 //      try to obtain an equation with a higher degree
26 if (regressionX.R2() < thresholdR2)
27     regressionX = new PolynomialRegression(t, x, degree + 1);
28 if (regressionY.R2() < thresholdR2)
29     regressionY = new PolynomialRegression(t, y, degree + 1);
30
31 currentTr.setRegressionX(regressionX);
32 currentTr.setRegressionY(regressionY);
33
34 //      calculation of minimum  $R^2$ 
35 if (regressionX.R2() < minR2forX) {
36     minR2forX = regressionX.R2();
37     minR2forXid = tId;
38 }
39 if (regressionY.R2() < minR2forY) {
40     minR2forY = regressionY.R2();
41     minR2forYid = tId;
42 }
43
44 // calculation of average  $R^2$ 
45 double avgR2forX = trajectories.stream()
46     .mapToDouble(tr -> tr.getRegressionX().R2())
47     .average().getAsDouble();
48 double avgR2forY = trajectories.stream()
49     .mapToDouble(tr -> tr.getRegressionY().R2())
50     .average().getAsDouble();
51
52 // print results
53 LOGGER.info("min R2 for X is for trajectory {}: {}", minR2forXid,
54             minR2forX);
55 LOGGER.info("avg R2 for X is: {}", avgR2forX);
56 LOGGER.info("min R2 for Y is for trajectory {}: {}", minR2forYid,
57             minR2forY);
58 LOGGER.info("avg R2 for Y is: {}", avgR2forY);

```

С. Агломеративная Иерархическая Кластеризация

Листинг 7.3: Реализация кластеризации

```

1 import org.slf4j.Logger;
2 import org.slf4j.LoggerFactory;
3 import ru.griat.rcse.entity.Cluster;
4 import ru.griat.rcse.entity.Trajectory;

```

```
5 import ru.griat.rcse.entity.TrajectoryPoint;
6
7 import java.util.ArrayList;
8 import java.util.List;
9
10 import static java.lang.Math.*;
11
12 public class Clustering {
13
14     private static final Logger LOGGER = LoggerFactory.getLogger(
15         Clustering.class.getName());
16     private static final int OUTPUT_CLUSTERS_COUNT = 17;
17
18     private List<Cluster> clusters;
19
20     private Double[][] trajLCSSDistanсes;
21     private Double[][] clustLCSSDistanсes;
22     private int minX, maxX, minY, maxY;
23     private TrajectoryPoint cameraPoint;
24
25     public Clustering(List<Trajectory> trajectories) {
26         clusters = new ArrayList<>();
27         trajLCSSDistanсes = new Double[trajectories.size()][trajectories
28             .size()];
29         clustLCSSDistanсes = new Double[trajectories.size()][
30             trajectories.size()];
31     }
32
33
34     public Double[][] getTrajLCSSDistanсes() {
35         return trajLCSSDistanсes;
36     }
37
38     public void setTrajLCSSDistanсes(Double[][] trajLCSSDistanсes) {
39         this.trajLCSSDistanсes = trajLCSSDistanсes;
40         for (int i = 0; i < trajLCSSDistanсes.length; i++) {
41             System.arraycopy(
42                 trajLCSSDistanсes[i], 0,
43                 clustLCSSDistanсes[i], 0,
44                 trajLCSSDistanсes.length);
45         }
46     }
47
48     /**
49      * set borders for an input image in terms of pixels
50      * calculate the position of a camera
51  }
```

```

47  /*
48   * public void setBorders(int minX, int maxX, int minY, int maxY) {
49   *     this.minX = minX; this.maxX = maxX;
50   *     this.minY = minY; this.maxY = maxY;
51   *     this.cameraPoint = new TrajectoryPoint(
52   *         (int) Math.round(0.25 * maxX),
53   *         (int) Math.round(0.95 * maxY));
54   */
55
56 /**
57 * Single linkage
58 * LCSS similarity measure
59 *
60 * @param trajectories database of trajectories
61 * @return clusters of trajectories
62 */
63 public List<Cluster> cluster(List<Trajectory> trajectories) {
64     initClusters(trajectories);
65     whileCluster(OUTPUT_CLUSTERS_COUNT);
66     return clusters;
67 }
68
69 /**
70 * initialize clusters with each trajectory singly
71 */
72 public void initClusters(List<Trajectory> trajectories) {
73     trajectories.forEach(trajectory ->
74         clusters.add(new Cluster(trajectory.getId(), trajectory)
75     );
76 }
77
78 /**
79 * stopPoint - desired number of clusters to stop:
80 * if null - stop when 1 cluster is left
81 * if no joins are possible, stop.
82 */
83 public void whileCluster(Integer stopPoint) {
84     if (stopPoint == null)
85         stopPoint = 1;
86     int numOfClusters = clusters.size();
87     int id1;
88     int id2;
89     double minClustDist;
90     while (numOfClusters > stopPoint) {
91         id1 = -1;

```

```

91         id2 = -1;
92         minClustDist = Double.MAX_VALUE;
93         for (int i1 = 0; i1 < clusters.size(); i1++) {
94             for (int i2 = i1 + 1; i2 < clusters.size(); i2++) {
95                 if (i1 != i2
96                     && clustLCSSDistances[clusters.get(i1).getId()
97 ()][clusters.get(i2).getId()] != null
98                     && clustLCSSDistances[clusters.get(i1).getId()
99 ()][clusters.get(i2).getId()] < minClustDist) {
100                     minClustDist = clustLCSSDistances[clusters.get(
101 i1).getId()][clusters.get(i2).getId()];
102                     id1 = i1;
103                     id2 = i2;
104                 }
105             }
106         }
107         // join i1 and i2 clusters, add i1 traj-es to cluster i2
108         clusters.get(id1).appendTrajectories(clusters.get(id2).
109             getTrajectories());
110         // recalculate D for i1 and i2 lines -> set i2 line all to
111         // NULLS
112         recalclClustersDistMatrix(id1, id2);
113         // remove i2 from 'clusters'
114         clusters.remove(id2);
115         // numOfClusters--;
116     }
117
118 /**
119 * Calculates LCSS distance for two input trajectories
120 * Smaller the LCSS distance - the better (0.0 - equal trajectories)
121 *
122 * @param t1 first trajectory
123 * @param t2 second trajectory
124 * @return LCSS distance for t1 and t2
125 */
126 public Double calcLCSSDist(Trajectory t1, Trajectory t2) {
127     int m = t1.length();
128     int n = t2.length();
129
130     double delta = getDelta(m, n);

```

```

131     double epsilonX = getEpsilonX(m, n);
132     double epsilonY = getEpsilonY(m, n);
133
134     double dist = 1 - calcLCSS(t1, t2, delta, epsilonX, epsilonY) /
135     min(m, n);
136     trajLCSSDistances[t1.getId()][t2.getId()] = dist;
137     clustLCSSDistances[t1.getId()][t2.getId()] = dist;
138     return dist;
139 }
140
141 /**
142 * Calculates LCSS for two input trajectories
143 * Bigger the LCSS - the better
144 *
145 * @param t1      first trajectory
146 * @param t2      second trajectory
147 * @param delta    δ parameter: how far we can look in time to match
148 * a given point from one T to a point in another T
149 * @param epsilonX ε parameter: the size of proximity in which to
150 * look for matches on X-coordinate
151 * @param epsilonY ε parameter: the size of proximity in which to
152 * look for matches on Y-coordinate
153 * @return LCSS for t1 and t2
154 */
155
156 private Double calcLCSS(Trajectory t1, Trajectory t2, Double delta,
157 Double epsilonX, Double epsilonY) {
158     int m = t1.length();
159     int n = t2.length();
160
161     if (m == 0 || n == 0) {
162         return 0.0;
163     }
164
165 //     check last trajectory point (of each trajectory-part recursively
166 // )
167 //     delta and epsilon as thresholds for X- and Y-axes respectively
168 //     Then the abscissa difference and ordinate difference are less
169 //     than thresholds (they are relatively close to each other), they are
170 //     considered similar and LCSS distance is increased by 1
171     else if (abs(t1.get(m - 1).getX() - t2.get(n - 1).getX()) <
172     epsilonX
173             && abs(t1.get(m - 1).getY() - t2.get(n - 1).getY()) <
174     epsilonY
175             && abs(m - n) <= delta) {

```

```
166         return 1 + calcLCSS(head(t1), head(t2), delta, epsilonX,
167     epsilonY);
168     } else {
169         return max(
170             calcLCSS(head(t1), t2, delta, epsilonX, epsilonY),
171             calcLCSS(t1, head(t2), delta, epsilonX, epsilonY)
172         );
173     }
174 }
175 /**
176 * Calculates shortened trajectory by excluding last trajectory
177 point
178 *
179 * @param t trajectory
180 * @return trajectory without last trajectory point
181 */
182 private Trajectory head(Trajectory t) {
183     Trajectory tClone = t.clone();
184     tClone.getTrajectoryPoints().remove(tClone.length() - 1);
185     return tClone;
186 }
187 /**
188 * calc δ
189 *
190 * @param m length of first trajectory
191 * @param n length of second trajectory
192 * @return δ value
193 */
194 private Double getDelta(int m, int n) {
195     return 0.5 * min(m, n);
196 }
197 /**
198 * calc ε for X
199 *
200 * @param m length of first trajectory
201 * @param n length of second trajectory
202 * @return ε value
203 */
204 private Double getEpsilonX(int m, int n) {
205     return 0.1 * (maxX - minX);
206 }
207
208
```

```

209  /**
210   * calc ε for Y
211   *
212   * @param m length of first trajectory
213   * @param n length of second trajectory
214   * @return ε value
215   */
216  private Double getEpsilonY(int m, int n) {
217      return 0.1 * (maxY - minY);
218  }
219
220 /**
221  * At each step calc a distance matrix btwn clusters
222  * Merge two clusters with a min dist -> requires an update of the
223  * dist matrix
224  * because of the implementation: clusterId1 < clusterId2
225  *
226  * @param clusterId1 index of left joined cluster in clusters list (
227  * remained cluster)
228  * @param clusterId2 index of right joined cluster in clusters list
229  * (removed cluster)
230  */
231  private void recalcClustersDistMatrix(int clusterId1, int clusterId2)
232  {
233      for (int i = 0; i < clusterId1; i++) {
234          clustLCSSDistances[clusters.get(i).getId()][clusterId1] =
235              calcClustersDist(clusters.get(i), clusters.get(clusterId1));
236      }
237      for (int j = clusterId2; j < clusters.size(); j++) {
238          clustLCSSDistances[clusterId2][clusters.get(j).getId()] =
239              calcClustersDist(clusters.get(clusterId2), clusters.get(j));
240      }
241      clustLCSSDistances[clusters.get(clusterId1).getId()][clusters.
242      get(clusterId2).getId()] = null;
243  }
244
245 /**
246  * Calculates inter-clusters distance for two input clusters
247  * using 'single-link' linkage method:
248  * the between-cluster distance == the min distance btwn two
249  * trajectories in the two clusters
250  *
251  * @param cluster1 first cluster

```

```
246     * @param cluster2 second cluster
247     * @return distance between clusters
248     */
249     private Double calcClustersDist(Cluster cluster1, Cluster cluster2)
250     {
251         double dist = Double.MAX_VALUE;
252         for (Trajectory trajectory1 : cluster1.getTrajectories()) {
253             for (Trajectory trajectory2 : cluster2.getTrajectories()) {
254                 Double lcssDist = trajLCSSDistances[trajectory1.getId()]
255                         [trajectory2.getId()];
256                 if (lcssDist != null && lcssDist < dist)
257                     dist = lcssDist;
258             }
259         }
260     }
261 }
```