

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,

и^{нициалы, фамилия})

(личная подпись)

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 A.R. Mardanova _____
(initials, surname) _____
(personal signature)

SUPERVISOR Dr. Tech. Sc., Prof. I.V. Anikin _____
(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	19
3.3 Метрики измерения близости и дальности	23
3.4 Выводы	27
4 ПРЕДЛОЖЕННЫЙ ПОДХОД	29
4.1 Концептуальная модель фреймворка	29
4.2 Архитектура фреймворка	30
4.3 Кластеризация	32
4.3.1 Агломеративная иерархическая кластеризация	32
4.3.2 Моделирование кластеров	33
4.4 Сравнение траекторий	33
4.4.1 LCSS метрика для измерения схожести траекторий	34
4.4.2 Адаптивность параметров LCSS алгоритма	34
4.5 Проверка достоверности кластеров	36
4.6 Аппроксимация траекторий	38
4.6.1 Регрессионный анализ	38

4.6.2	Полиномиальная регрессия	39
5	РЕАЛИЗАЦИЯ ФРЭЙМВОРКА	41
5.1	Используемые технологии	41
5.2	Работа со входными данными	42
5.2.1	Описание входных данных	42
5.2.2	Структура файла входных данных	43
5.2.3	Обработка входных данных	44
5.2.4	Аппроксимация траекторий с использованием Полиномиальной Регрессии	46
5.2.5	Выбор ключевых точек в аппроксимированных траекториях	47
5.3	Анализ траекторий	52
5.3.1	Вычисление метрики схожести траекторий	52
5.3.2	Кластеризация	53
5.3.3	Моделирование кластеров	53
6	Результаты	54
6.1	Оценка точности	54
6.1.1	Результаты аппроксимации траекторий	54
6.1.2	Результаты кластеризации траекторий	58
7	ЗАКЛЮЧЕНИЕ	59
Литература		60
Список иллюстраций		65
Список таблиц		66
Список алгоритмов		67
Список листингов		67
ПРИЛОЖЕНИЕ		I
A.	Алгоритм парсинга входных траекторий	I
B.	Инициирование 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
ИПП	Интерфейс Прикладного Программирования
ГПИ	Графический Пользовательский Интерфейс
DTW	Dynamic Time Warping
LCSS	Longest Common SubSequence
FARS	Fatality Analysis Reporting System
ID	Identifier
SVM	Support Vector Machine
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DI	Dunn's Index
API	Application Programming Interface
AWT	Abstract Window Toolkit
GUI	Graphical User Interface

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 17 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 [?]. 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 наименьших 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 аварийные 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 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 наимпростейший 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 значи-

тельно 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 значительной 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 [4][2]. 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

This chapter presents state of the art analysis of existing approaches and discusses advantages and disadvantages of existing approaches. Each section is concluded with a summary, in which the chosen approach is specified and a short argumentation is given.

3.1 Методы обнаружения аномалий

According to [14], the task of outliers detection, which is a main objective task of this work, has been an object of interest and studies of research community originating from 19th Century. Nowadays a huge variety of different techniques for solving the task of detecting outliers and abnormalities in video traffic data are presented, and these approaches can be classified in various ways.

For example, data mining techniques in general and anomaly detection techniques as well as clustering approaches, which are one of the ways to solve the task of outliers identification, particularly can be classified as supervised, semi-supervised and unsupervised on the grounds of the manner of labeling the input data] [14][17]:

1. *Supervised*. Input data used for training contains labels for both normal and anomalous instances. As a result the algorithm can build models for both normal and abnormal classes;
2. *Semi-Supervised* [14] or *Weakly-Supervised* [5]. Input training data set contains class labels only for normal data instances. Such techniques are more widely used than supervised approaches, since anomalous data instances are usually not predictable and random and it is difficult to provide examples to cover all possible anomalous events;
3. *Unsupervised*. Does not require input data to be labeled neither for normal nor anomalous data. Such algorithms are based on the expectation that normal data instances are significantly more frequent than anomalous ones in the test data set and therefore are not applicable when this assumption is disrupted.

Alternatively, according to surveys done by Chandola in [14], Kumaran in [16] and Malik in [17], anomaly detection techniques can be classification based, nearest-neighbor based, clustering based, statistical and etc. The following offers the short overview of mentioned groups.

Основанные на кластеризации

The main concept of these methods lies in using a classifier which firstly learns to distinguish inliers and outliers and then classifies each input instance [19]. Such techniques consist of training and testing phases. Training, or learning, phase supposes learning a classifier model from a training data set, containing labeled data instances. The learned classifier is then used to classify an input trajectory as normal or anomalous by assigning a class label in a testing phase.

Depending on how testing data instances are labeled, all classification based anomaly detection techniques can be one-class or multi-class. The first type assumes that all training data instances are normal and are labeled as one class. During training phase model learns a discriminative boundary around normal instances, and a trajectory, which is not aligned with the learned normal class description, is considered as an anomalous. Single-class Support Vector Machines (SVMs) is the most commonly used classification based approach, which is applicable to the task of anomalous trajectory detection, as it was proposed by Piciarelli *et al.* [20][21]. However, this approach requires trajectory vectors to be the same length. Since raw trajectory data is usually contains different amount of trajectory points due to different speed of moving objects, it is necessary to preprocess raw trajectories to normalize them to vectors of the same length [21]. Moreover, SVMs become highly time and memory consuming while working with huge amounts of multi-dimensional data [22].

The latter category supposes learning multiple classes during training step and then using a classifier to review the input trajectory for compliance with each learned class. In literature different descriptions of training phase and training data labels are given. According to [14], training data contains only normal data instances with corresponding normal class labels, and during training phase model learns multiple discriminative boundaries around each class of normal instances. A trajectory, which is aligned with none of the learned normal class descriptions, is considered as an anomalous. In other words, an anomalous trajectory will not be accepted by neither of the classifiers. In [16] it is assumed that model is learned using training data containing labels for normal and anomalous classes. Therefore, a classifier can classify an input trajectory as belonging to a normal or anomalous class.

The advantage of two-phased classification based algorithms is a fast testing phase due to precomputed classifier model used to classify each input instance. Also such algorithms can perform well in cases when anomalous data instances form a class or cluster [19]. However, the training step requires accurately labeled training data, which is often not available.

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

Proximity based approaches decide whether a data instance is normal or anomalous based on how close or far is it located with respect to neighbors [16]. Nearest-neighbor and density based approaches are based on the assumption, that «normal data instances have dense neighborhood, while anomalous data instances occur far from their closest neighbors» [14].

In order to be able to compare the surrounding density for an instance under consideration with the density around its local neighbors, a distance (dissimilarity) or similarity measure between two data instances needs to be specified [19]. By virtue of an anomaly score calculation method, techniques can be grouped into two categories: 1) the anomaly score is calculated as a distance of a data instance to its k^{th} nearest neighbor and 2) to compute the anomaly score the relative density of each data instance is being computed [14].

These approaches has several disadvantages. First of all, in comparison with classification based anomalies detection techniques, the computational complexity of the testing phase is considerably higher, since nearest neighbors are computed by computing the distance for each test data instance with all instances from either testing and training data. In case of multi-dimensional trajectory data, the task of distance computation becomes even more complicated. Moreover, the accuracy of labeling decreases when the main assumption is violated: when normal instances have sparse neighborhood or anomalous instances have dense [14].

Основанные на кластеризации

Clustering is an efficient approach aimed to group data instances into different classes, called clusters, based on their similarity in such a way, that objects in one cluster are similar to each other and dissimilar to objects in other clusters [10][22]. ST clustering supposes grouping objects on the ground of their spatial and temporal similarity. To compare data instances before grouping them into clusters, similarity or distance between them needs to be measured.

There are three types of clustering based anomalies detection techniques with following assumptions: 1) normal data instances are associated with a cluster, while anomalous data instances are not associated with any cluster, 2) normal data instances are close to the cluster center, while abnormal instances lie far away from the closest cluster center and 3) normal data instances lie in large and dense clusters, while anomalies are associated with sparse clusters or clusters with a small cardinality [14][16]. Techniques of first type can be implemented using one of the clustering methods which do not require every data instance to belong to some cluster, for example DBSCAN [23]. Algorithms from second group consist of two phases: 1) data clustering and 2) calculating an anomaly score for each data instance. Techniques of the latter type require a threshold for cardinality size and/or density of a cluster to be defined to decide whether a cluster refers to normal or anomalous data.

The necessity to compute distance between trajectories in some of the clustering based approaches makes them similar to neighbor based approaches. As it is stated in [14], techniques are different in the way they process instances: in clustering based techniques each instance is evaluated with respect to the corresponding cluster, while in neighbor based techniques each instance is being inspected with respect to its proximate neighborhood. Consequently, the selection of distance computation method plays an important role and affects results and performance significantly.

On the other side, dividing all training data into groups makes clustering based algorithms similar to classification based algorithms. Though in classification based approaches class is assigned based on given labels, while in clustering based approaches classification is not given in advance [19].

One of the main advantages of clustering based techniques is the ability of majority of them to run in an unsupervised manner. For the case of TVS-based trajectory data acquisition the unsupervised learning methods are the most appropriate, because labeling hours of video data is a highly time-consuming task. Also, manual labeling of input data can lead to errors due to human operator intervention.

Moreover, clustering based techniques are adjustable to work with complex data types because of adaptability of clustering algorithms. However, at the same time they are computationally expensive and are used primarily for relatively low dimensional data, highly dependent on the used clustering algorithm and can not effectively deal with situations when anomalies form significant separate cluster groups [14].

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

The main concept of model based algorithms is that they represent the data as a set of parameters to create the model of a normal behavior. As an advantage, model based approaches do not ask the user to provide any input parameters, because all the parameter values can be derived from the data. Statistics based approaches can be considered as a subcategory of model based approaches, they are treated as one of the earliest algorithms and can be used as a basis by the various outlier detection techniques [17]. As it is stated in [14], the main idea of statistical approaches is that data instances occurring in high probability regions of a stochastic model assumed to be normal, while data instances from the low probability regions refer to anomalies. So, statistical approaches are based on using statistical stochastic model to fit to the given data and then applying a statistical inference test, also called discordance test, to decide if a data instance is normal or anomalous. It comes from the main concept that «based on results of applied statistical test, anomalies have low probability to be generated from the learned stochastic model» [14].

Statistical techniques in turn can be parametric or non-parametric [17]. In parametric approaches the normal data is supposed to fit the parametric distribution and probability

density function with estimated from the given data parameters [16]. One of the advantages of parametric techniques is that the data size does not affect the model: models grow only depending on a model complexity. However, the necessity to fit the data into some preselected distribution model complexifies and limits the application of such approaches: it is difficult to fit the data to one distribution. In this case it is possible to use a multiple-distribution model to match some clusters of the data with particular distributions [19]. One of the most known examples of parametric methods is Regression Method [17].

By contrast to this, non-parametric approaches are based on using non-parametric statistical models with structures, which are not defined in advance: the given data is used to determine the structure dynamically. Such approaches do not suppose making assumptions about the statistical distribution of the data [17].

Since statistical approaches are based on fitting a statistical model, the choice of it significantly affects results, computational complexity and performance. Nevertheless, the main assumption of statistical approaches that the data comes from a particular distribution can not be always satisfied, specifically for the case of a multi-dimensional data [14].

Выводы

Based on the given description of different approaches and their advantages and disadvantages, it was decided to focus on clustering based anomalies detection approaches for several reasons:

- 1) they can work in an unsupervised mode without a human intervention and do not require the input data to contain labels,
- 2) input data is allowed to contain anomalous trajectories,
- 3) clustering method can be easily applied to such a multi-dimensional data as trajectories by defining a suitable similarity measure.

That means that a clustering method and a similarity measure need to be specified.

3.2 Классификация методов кластеризации

Clustering is a highly researched form of data mining, and huge variety of clustering methods has already been proposed in literature [10]. State-of-the-art analysis of related research papers revealed that all traditional clustering approaches are usually categorized into five types: partitioning, hierarchical, density-based, model-based and grid-based methods [5][10]. Next paragraphs will briefly discuss each of the categories with highlighting main assumptions and concepts.

Методы, основанные на разделении или декомпозиции

Such methods are based on partitioning the trajectories data set randomly and then regrouping clusters by reassigning objects from one partition to another to minimize the objective function. They require the predefined parameter, usually denoted as k , which determines the amount of final clusters, or partitions, to be created. The main requirement is that number of partitions must be smaller than number of initial data points, since each partition forms a cluster, that means that it must be non-empty and contain at least one data instance, and each data instance must be included into exactly one cluster.

One of the most well-known examples of partitioning clustering algorithms is a K -Means algorithm, where firstly k cluster centers are initialized randomly and then data points are iteratively reassigned to the closest clustering center based on the discrepancy to minimize the clustering error [24]. The clustering error is defined as the sum of the squared Euclidean distances between each data set point and the corresponding cluster center [25]. The process is stopped when there are no more changes in clustering centers.

The disadvantages of the traditional K-Means clustering method are inability to form clusters of arbitrary form, dependence on initial random cluster centers initialization and high memory consumption [10]. Also finding an appropriate partitioning technique is a challenging task.

Hierarchical methods

In hierarchical based methods the given data set is decomposed into multiple levels to organize a hierarchical tree of clusters. The resulting hierarchical structure can be depicted as a tree [24].

There are two different ways of hierarchical decomposition: 1) the bottom-up (combining) and 2) the top-down (split, divisive) decomposition. They refer to agglomerative and divisive (split) clustering approaches respectively [26].

Agglomerative hierarchical clustering algorithms start by assigning each data instance to a distinct singleton-cluster, so the number of initial clusters is equal to the exact amount of data instances in input data, and then continue uniting clusters based on theirs similarity until all the initial clusters are merged into one single cluster or into predefined amount of clusters [27]. This is done by repeatedly executing following two steps: 1) identifying the two closest clusters and then 2) merging these two clusters [26]. Proximity matrix is used to store similarity measurements between clusters and is being updated on each step by computing distances between the new cluster and the other clusters.

The divisive hierarchical clustering algorithms work in a reverse manner: initially all data instances belong to one cluster and then step by step clusters split into smaller clusters until all of them become singleton clusters or until satisfying some predefined end condition.

Hierarchical clustering is supposed to be simple, but it is necessary to choose between agglomerative and split methods. Divisive clustering is more expensive in computation, therefore, it is less common than agglomerative approaches. Irreversibility of both splitting or uniting processes in traditional hierarchical clustering algorithms is also a particularity of such algorithms [10].

Since approach includes clusters joining, a significant task of agglomerative clustering algorithms is defining and computing the similarity or distance between clusters. This similarity can also be referred to as an inter-cluster or between-cluster distance. For the case of single-trajectory clusters the similarity between them is simplified and is equal to the similarity between respective trajectories. For multiple-trajectory clusters the similarity is computed according to a chosen linkage method. In literature following linkage methods are given as mostly common: single link, complete link, average link [24][28]. The choice of the linkage method depends on the application domain [26]. In the case of the single link distance between two clusters is defined as the minimum distance between two trajectories in these clusters, that means that the similarity between of two clusters is determined by two closest trajectories. The complete link linkage method implies taking the maximum distance between two trajectories in two clusters as an inter-cluster distance, so it is defined using the farthest distance of trajectory pairs. The average link supposes calculating averaged paired distance between all trajectory pairs in these two clusters.

A convenience of agglomerative hierarchical clustering approaches is that they do not require the number of resulting clusters to be predefined, so they are appropriate for clustering vehicle trajectories, because number of clusters of normal or anomalous trajectories is not known in advance. However, the most well-known disadvantage of hierarchical clustering algorithms is that they are not robust and can suffer from noise and anomalies.

Density-based methods

In comparison with the partitioning and hierarchical clustering approaches, density-based methods objects inspect similarity based on the density of the data [22]. The area is being added to the nearest cluster, while density of the points in the area remains greater than the predefined threshold [10]. Clusters form dense regions of objects and they are separated by sparse regions with low density.

The main advantage of density-based clustering approaches is that they are able to form clusters of arbitrary forms, extend beyond spherical [10]. Also they are appropriate for clustering huge data sets of trajectories in an unsupervised manner and do not require the amount of clusters to be known in advance [5][22]. However, the results quality highly dependent on the amount of trajectories in training data set, available for analysis.

The most well-known and commonly used density-based algorithm is a DBSCAN, proposed by M. Ester *et al.* in [23]. According to it, input data points are categorized as follows:

core data, density-reachable data and outliers based on parameters ε , $minPts$ and the density threshold. Neighbor parameter ε and $minPts$ specify the maximum remoteness and minimum amount of satisfying points while choosing the core points: at least $minPts$ points must be present within distance ε from the core point, these points are marked as directly reachable from the chosen core point. Aforementioned parameters need to be predefined by the user, but it is difficult to determine them correctly. Each cluster must contain at least one core point. Points are denoted as anomalous if they are not reachable from any of the other points.

Shrinkage-based or Grid-based methods

The main idea of grid-based algorithms lies in applying a multi-resolution grid data structure: the data space is quantized into a finite number of cells (units) that form a multi-resolutinal grid structure. Each cell stores summary information about data objects within its subspace [22]. Since clustering operations are performed on the created grid, and also important trajectories characteristics can be computed in each of the spatial grid cells, the quality of data compression influences the quality of results significantly [7]. Density of closely located dense cells can help to determine clusters. A trajectory can be considered as an anomalous if it differs from the expected trajectory in a number of covered grid cells [22].

The main advantage of grid-based clustering algorithms is an improved performance: increased processing speed and processing time becomes independent on the size of the data set, only the number of cells in each dimension in the quantized space affects the processing time [10].

Model-based methods

In comparison with the above methods, which analyze distance among data objects, in model-based approaches data is supposed to be generated by a mixture of probability distributions, where each component of mixture represents a cluster. So a mathematical model is assigned to each cluster, and then method attempts to find the best fitting data for the chosen model. In this way such methods seek to increase the adaptability between given data and some statistical models [10][22]. The idea of model-based algorithms is that in order to locate clusters they describe the spatial distribution of the input data points by building density functions. The model-based approaches are typically used in feature-specific clustering and depend on the selected features and model [5].

It is emphasized, that model-based approaches show good performance while working with complex data types. This category usually includes statistical and neural network methods [10].

Graph-based methods [7]

Another category of clustering methods in application to vehicle trajectories data. Liu *et al.* in [29] presented a graph-based approach to solve the problem of detection of outliers in traffic data streams. A graph structure was used to store the traffic: nodes represent regions while edge

weights depict the traffic flow. Edge anomalies in the graph denote the traffic abnormalities, and causal outlier tree can then be used to further analyze these outliers to find causal interactions.

Another higher-level classification of clustering methods can consist of only two subclasses on the ground of properties of generated clusters: hierarchical and partitioning approaches [24]. Hierarchical algorithms group objects into clusters from singleton cluster to cluster containing all data instances or in a reverse direction. While partitioning clustering algorithms divide given data set into a predefined number of clusters in a single-layer structure.

In order to perform clustering, the similarity between two trajectories needs to be defined. Different existing distance measures will be reviewed in following paragraphs.

Выводы

Based on the given description of clustering approaches, their limitations, advantages and disadvantages, it was decided to focus on a hierarchical clustering approach, more specifically on an agglomerative hierarchical clustering, because it can deal with limitations of a given input data, that are: absence of input labels, unknown number of resulting clusters, presence of both normal and anomalous trajectories in input data.

3.3 Метрики измерения близости и дальности

As it was mentioned before, clustering based approaches require a similarity measure to be defined between two trajectories. Apart from that, distance and similarity measures are also used to compare a trajectory with a cluster or a pair of clusters between each other. A similarity measure highly dependent on the format of a trajectory. A trajectory data, represented as a multidimensional data, can contain quantitative or qualitative features, continuous or binary. In such a classification, distance measure functions are more appropriate to work with continuous features, while similarity measures – with qualitative features [24]. Input trajectory-vectors in this work contain spatial information along with temporal, which can be termed as qualitative continuous data. That means that distance measure functions are more appropriate in this case. Moreover, distance and similarity functions can be classified as 1) working with raw representations of trajectories without any preprocessing steps and 2) working with preprocessed trajectories representations. Preprocessing can include unifying the length of trajectories or reducing the dimensionality of trajectory-vectors [28].

Some of the most known and widely used traditional similarity measures are following: Euclidean Distance, Fréchet Distance, DTW, LCSS.

Евклидово расстояние (Euclidean Distance)

Euclidean distance between two trajectory vectors is calculated as a sum of squared differences of corresponding spatial coordinates [18]:

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

where both trajectories consist of m tracking points and are represented by two-dimensional vectors $T_i = \{t_i^1, t_i^2, \dots, t_i^m\}$ and $T_j = \{t_j^1, t_j^2, \dots, t_j^m\}$. Tuples $(t_{i_x}^k, t_{i_y}^k)$ represent spatial coordinates for a k -th tracking point of i -th trajectory from a data set.

However, Euclidean distance works only with trajectories with equal number of tracking points. Since usually vehicles move with different speed and behavior, trajectory length is always different and that means that raw trajectories need to be preprocessed and reduced to the same size [28]. Also, traditional Euclidean distance requires two-dimensional data, meaning that it is not able to process temporal information, and is dependent on the trajectory direction: the reversed direction can cause incorrect distance measurement, that in its turn leads to errors in clustering. Also, it fails while working with trajectories moving in a similar way but with different speeds and in the case of different sampling rates [30].

Расстояние Фреше (Fréchet Distance)

Fréchet Distance is based on Euclidean distance. It considers the positional and sequential relationship of trajectory points while calculating the similarity. The main idea of this approach is computing Euclidean distance for each pair of points from two trajectories and then designating the maximum Euclidean distance as a Fréchet Distance between them [10][31]. However, since only the maximum among distance is considered, the approach is sensitive to the presence of outliers.

Алгоритм динамической трансформации шкалы времени (DTW)

Алгоритм динамической трансформации шкалы времени (Dynamic Time Warping, DTW). Dynamic Time Warping (DTW) is one of the algorithms for measuring the similarity between two temporal time series sequences, which may vary in speed. The objective of time series comparison methods is to produce a distance metric between them two. DTW method aims to find an alignment between time-dependent sequences, such as trajectories, and is able to process trajectories of different lengths [10].

According to [10], DTW distance is calculated as follows (Formula 3.3.2):

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

where $D_D(T_i, T_j)$ refers to DTW distance between two trajectory segments with lengths m and n , $dist(a_i, b_j)$ means the Euclidean Distance between two trajectory points. Function $Rest(T_i)$ takes the remaining part of a trajectory after excluding the point a_i . It can be seen, that in case of zero-length trajectories the DTW distance is equal to 0, for the case then only one of two trajectories is non-empty, the distance between them is considered to be infinite. For two non-empty trajectories, the minimum distance between them is calculated in a recursive way.

Though the important advantage of the DTW method is its ability to process trajectory vectors of distinct lengths, DTW distance is not robust to noise and requires trajectory points to be continuous. Also DTW distance computation is highly time consuming and complex due to necessity to compare distances between each pair of trajectories.

Метод наибольшей общей подпоследовательности (LCSS)

Longest Common SubSequence (LCSS) distance tries to match two trajectory sequences based on the longest common sub-sequence between them. The LCSS algorithm works with discrete values and calculates the largest number of equivalent points between the two trajectories. The task of finding the longest common sub-sequence is usually solved recursively [10]: possible translations, or shiftings, are calculated in each dimension and used to provide the maximum LCSS [32]. The basic idea of an LCSS distance is that it allows two trajectories to stretch. In comparison with DTW and Euclidean distances, LCSS enables some elements to remain unmatched [33] and, in comparison with DTW, LCSS is more robust against presence of outliers [34].

The LCSS distance is calculated according to the Formula 3.3.3 [28]:

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

where m and n are lengths of trajectories T_1 and T_2 respectively. $LCSS_{\delta, \epsilon}(T_1, T_2)/\min(m, n)$ can be also referred to as an LCSS similarity and takes value between 0 and 1.

The $LCSS_{\delta,\epsilon}(T_1, T_2)$, the longest common sub-sequence between trajectories, represents the number of matched trajectory points between trajectories T_1 and T_2 and is defined as follows (Formula 3.3.4):

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

As it can be seen, LCSS calculation depends on two constant parameters: δ (point spacing [32]) and ϵ (point distance [32] or matching threshold [33]):

- parameter δ defines the maximum remoteness in terms of time between two trajectory points in which we can look to match a given point from one trajectory with another. Also can be defined as a value representing the maximum index difference between two input trajectories allowed in calculation [32].
- constant ϵ defines the size of proximity to look for matches in terms of spatial information. According to [32] it is a floating point number which represents the maximum allowed distance between trajectory points in each dimension to consider them as equivalent: difference between X - and Y -coordinates less than ϵ value means that points are relatively close to each other and can be considered as equivalent. LCSS distance is increased by 1 in this case.

Parameters δ and ϵ affect results significantly, therefore, the task of choosing the optimal values for them is challenging and important [28][30]. The $\text{Head}(T)$ function is defined to return the first $M - 1$ points from the trajectory T , representing the trajectory with the last trajectory point removed. According to implementation given in [32] the LCSS computation, based on a dynamic programming approach, has a complexity of $O((m + k)\delta)$. However, the algorithm requires predefined constant δ and ϵ parameter values as an input to a method. Also, due to the recursive way of computations, LCSS has a high computational cost [35].

Выводы

The LCSS distance is the most appropriate in this work, since it allows the trajectories to contain noise, have different length, objects speed and sampling rates (local time shifts in

trajectories) [28]. Moreover, among the aforementioned methods, the LCSS distance is the most robust approach against noises.

3.4 Выводы

Родственные методы

The aforementioned objective has been investigated and solved in numerous works using different methods. Since in fact normal events are common and dominate the data, and abnormal events are rare and difficult to describe explicitly, many approaches are based on an unsupervised clustering of trajectories. For this thesis work the approach proposed by Ghrab, Fendri, Hammami in [28] was chosen as a basis. It is focused on detection of abnormalities based on a trajectories clustering.

The proposed approach can be described as a two-phase approach with offline clustering to extract frequent trajectories and an online classification of an input trajectory to label it as a normal or anomalous.

The clustering is done in an unsupervised manner using an agglomerative hierarchical clustering algorithm operating on a distance matrix between trajectories. To perform clustering, the LCSS distance is used as a similarity measure. The formulas and description of the LCSS distance are given in the previous section (Formulas 3.3.3 – 3.3.4).

Преимущества

One of the advantages of the proposed method is that the chosen similarity measure does not require the trajectories to be of the same length, so that the preprocessing of the trajectories, which is a high complexity process, can be avoided. Moreover, the training data is allowed to contain normal trajectories as well as anomalous: algorithm will extract both normal and anomalous clusters. Dense clusters will represent normal trajectories classes, sparse clusters – anomalous trajectories classes.

Недостатки

However, the disadvantage of the proposed method is that LCSS distance does not take into consideration such problems of video surveillance as a view perspective and a position of a moving object.

Поставленные задачи

This thesis work will be intended to investigate an opportunity of increasing the accuracy of results by making epsilon and sigma parameters, which are used to calculate the sigma, adaptable and dependent on the perspective and a distance from the camera. This includes:

- exploring a functional dependency between epsilon and sigma parameters and a distance from the camera,

- evaluating algorithm with different values.

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.

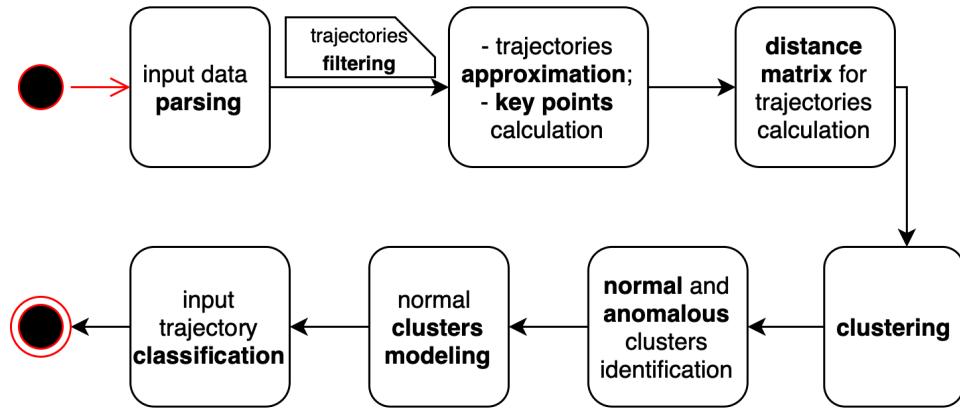


Рис. 4.1.1: Основные этапы работы фреймворка

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.

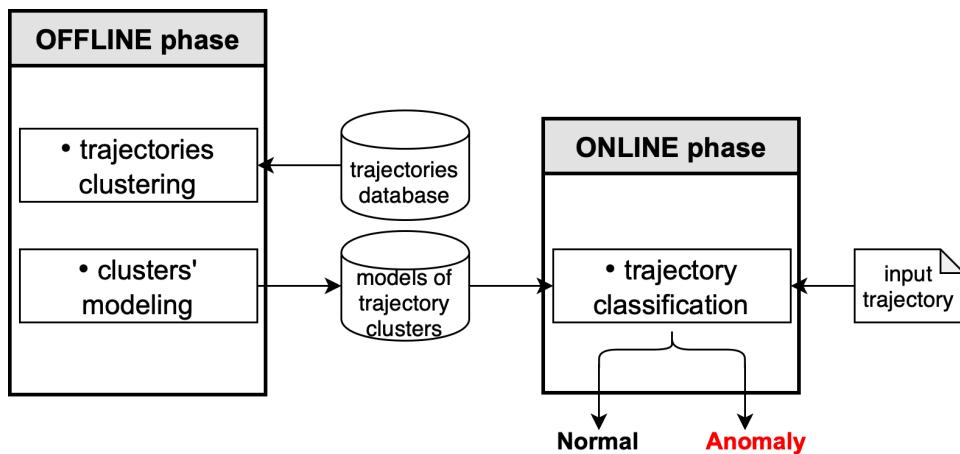


Рис. 4.2.1: Схема двухэтапного предложенного подхода

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.

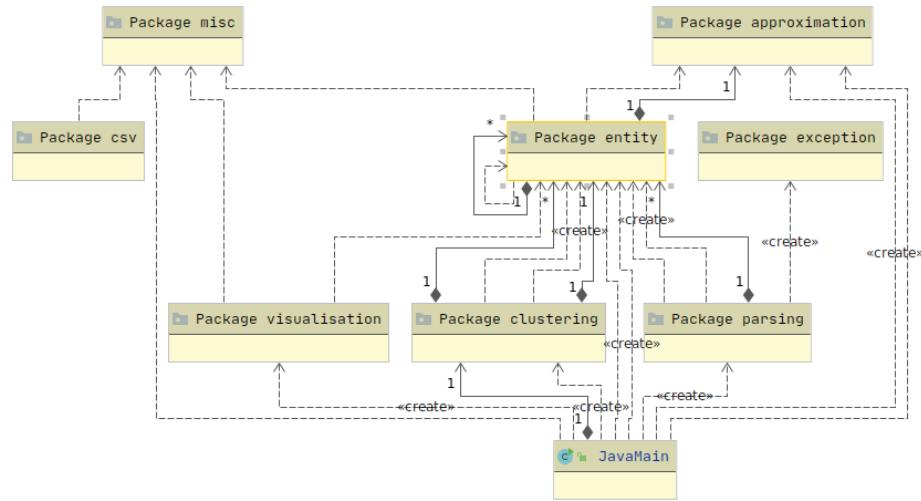


Рис. 4.2.2: Архитектура фреймворка

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 НММ (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 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.

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 - LCSS similarity / minimum(input lengths)
    end

```

Таким образом, можно предположить, что ε и расстояние от камеры находятся в обратной зависимости, что означает, что в итоговой формуле расстояние от камеры должно присутствовать в знаменателе формулы с некоторым коэффициентом.

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

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.

Algorithm 3: Определение адаптивных параметров LCSS

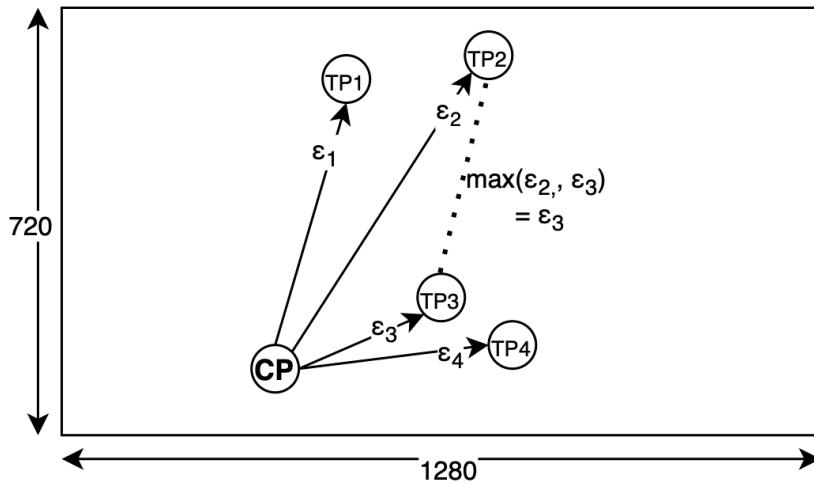
Input: First trajectory point: TP_1 ,
 Second trajectory point: TP_2

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 ε_1 and ε_2 values;
- take the $\max(\varepsilon_1, \varepsilon_2)$ as a final ε to compare TP_1 and TP_2 .

end

Рис. 4.4.1: Принцип выбора адаптивного значения ε

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 межклластерной дисперсией [39]. DI рассчитывается как отношение минимального межклластерного расстояния d_{min} к максимальному внутриклластерному диаметру d_{max} , и для k кластеров может быть определен следующим образом (4.5.1) [40]:

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

где минимальное межклластерное расстояние d_{min} в соответствии с методом связи по минимальной ссылке сводится к подсчету минимального расстояния между двумя траекториями из разных кластеров. Максимальный внутриклластерный диаметр d_{max} , или самое большое внутриклластерное 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:

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

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 рас-

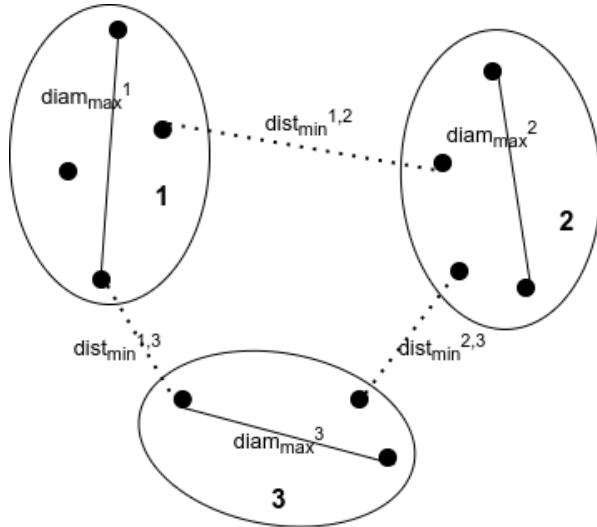


Рис. 4.5.1: Объяснение индекса DI

стояние 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 наблюдаемых 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, полиноми-

альная 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 перекрёстках 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. 5.2.1а).
2. Перекресток улиц Несмелова - Кировская Дамба, 2.txt (рис. 5.2.1б).
3. Перекресток улиц Московская - Галиаскара Камала, 3.txt (рис. 5.2.1в).
4. Перекресток улиц Московская - Парижской Коммуны, 4.txt (рис. 5.2.1г).

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

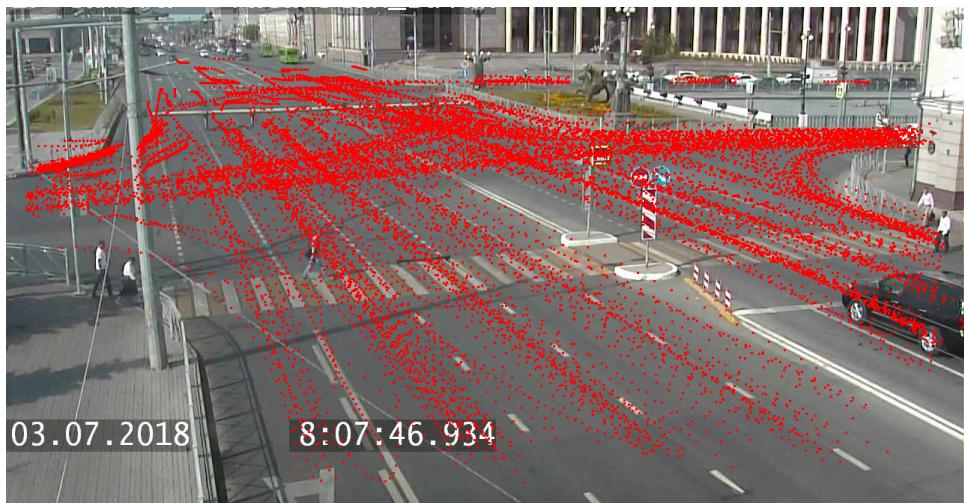


Рис. 5.2.2: Пример работы системы слежения на данных с первого перекрестка

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 инфор-



(a) игнорируемые траектории

(b) оставленные траектории

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

мацию о траекториях. Обнаружение аномалий второй категории, которая формируется из траекторий ТС, движущихся с аномально низкой или высокой скоростью, требует учета 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);
30 }
```

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

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>

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 (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 моделью (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 функции. 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;
```

```
26     }
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
7 *               given 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
15   ε) {
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 }
```

```

26     }
27 }
28
29 /**
30 * Calculates shortened trajectory by excluding last trajectory point
31 *
32 * @param t trajectory
33 * @return trajectory without last trajectory point
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.

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 результирующих 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

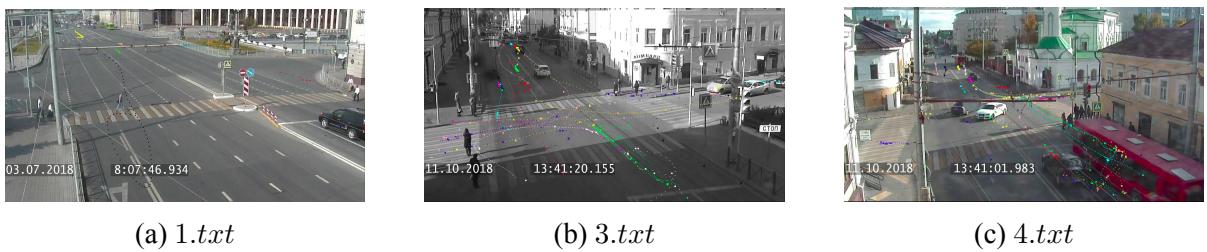


Рис. 6.1.1: Траектории, аппроксимированные полиномами 4-ой степени

степени не оказало 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

7 ЗАКЛЮЧЕНИЕ

In this work an approach for identification of trajectory anomalies in uncertain ST data was proposed and discussed. To perform an evaluation and implement the approach, the framework was developed. The source code of the implemented framework is available on GitHub repository [48]. For solving the task of outliers detection the clustering approach was used as a basis, specifically agglomerative hierarchical clustering algorithm. To calculate the similarity and dissimilarity between trajectories and clusters, the LCSS distance was chosen. However, as it was mentioned in previous chapters, LCSS distance calculation becomes infeasible for long trajectories. For that reason the approximation of input trajectories using polynomial regression was performed. According to the evaluation results, the best accuracy of approximation is achieved while using the 3th and 4th-degree polynomial functions jointly. Thereby, clustering was performed on a filtered set of approximated input trajectories using key points for each of them. The accuracy of the performed clustering was evaluated using a DI index and is equal to 0.95 value, which can be considered as a *good* result.

(In this work following results were achieved:)

As a result of this work following conclusions and deductions can be drawn:

- approximation of short trajectories with a non-constant speed requires higher-order polynomial functions for approximation
- LCSS distance allows trajectories to be of different lengths, but becomes extremely computationally expensive and complex for trajectories with more than 11-12 trajectory points
- approximation using a polynomial regression works well with the trajectory data, since it is known in advance that spatial coordinates of a trajectory are functionally dependent on the time (according to principles of physics, speed, acceleration, etc.)

Update
after
obtaining
final
results

Дальнейшее развитие

Последующие исследования

The implemented algorithm is designed in an offline-learning manner, that means that models of normal trajectories are learned offline beforehand and are not updated with new upcoming data on an on-going basis. The future researches can include investigating an opportunity of updating normal trajectories database in order to make the framework more adaptable to actual traffic data.

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. Gunopulos, 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	30
4.2.1 Схема двухэтапного предложенного подхода	30
4.2.2 Архитектура фреймворка	31
4.4.1 Принцип выбора адаптивного значения ε	36
4.5.1 Объяснение индекса DI	38
5.2.1 Примеры изображений с видеокамер на перекрестках 1-4	43
5.2.2 Пример работы системы слежения на данных с первого перекрестка	43
5.2.3 Результаты фильтрации траекторий на примере данных с первого перекрестка	45
5.2.4 Результаты полиномиальной регрессии с обозначением ключевых точек	51
6.1.1 Траектории, аппроксимированные полиномами 4-ой степени	56

Список таблиц

6.1.1 Значения коэффициента R^2 для полиномов различных степеней	55
6.1.2 Сравнение минимальной, средней, максимальной скоростей ТС	57

Алгоритмы

1	Описание Агломеративной Иерархической кластеризации	32
2	Описание алгоритма LCSS метрики	35
3	Определение адаптивных параметров LCSS	36

Листинги

5.1	Подсчет скорости ТС	45
5.2	Вызов метода решения полиномиальных уравнений	48
5.3	Вычисление граничных ключевых точек траектории	49
5.4	Вычисление LCSS	52
7.1	Алгоритм парсинга входных траекторий	I
7.2	Иницирование 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         }
67         while (nextChar == '[') {
68             increaseOpeningSqBracketsCount();
69             nextChar = (char) reader.read();
70         }
71         while (trajectoryCoordinatesStarted) {
72             if (nextChar == '(') {
73                 readCoordinates(reader);
74             }
75             nextChar = (char) reader.read();
76             if (nextChar == ']') {
77                 increaseClosingSqBracketsCount();
78             }
79         }
80         if (trajectoryStarted) {
81             if (nextChar == '[') {
```

```
82         increaseOpeningSqBracketsCount();
83         readTime(reader);
84     } else {
85         throw new TrajectoriesParserException("After coordinates array
86         with timestamps was expected");
87     }
88 }
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 /**
128 * Reads time and saves it into already initialized trajectory by
129 * updating trajectoryPoint at indexOfTP position in a current
130 * trajectory
131 */
132 private void readTime(InputStream reader) throws IOException {
133     char nextChar = (char) reader.read();
134     while (nextChar != ']') {
135         while (nextChar != ',' && nextChar != ']') {
136             t.append(nextChar);
137             nextChar = (char) reader.read();
138         }
139         if (nextChar == ']') {
140             increaseClosingSqBracketsCount();
141             trajectoryPoints.get(indexOfTP).setTime(Integer.parseInt(t.
142                             toString().trim()));
143             trajectoryPoints.get(indexOfTP).setTime(Integer.parseInt(t.
144                             toString().trim()));
145             indexOfTP++;
146             t = new StringBuilder();
147             nextChar = (char) reader.read();
148         }
149     }
150
151     private void increaseOpeningSqBracketsCount() {
152         openingSqBracketNumber++;
153         processBracketsCount();
154     }
155
156 /**
157 * Adds parsed trajectory into an array of output trajectories and
158 * prepares for the next input trajectory by resetting to 0 indexes and
159 * buffers
160 */
161 private void finishProcessingTrajectory() {
162     trajectories.add(new Trajectory(indexOfT, trajectoryPoints));
163     trajectoryPoints = new ArrayList<>();
164     indexOfT++;
165     indexOfTP = 0;
166     trajectoryStarted = false;
```

```

166     increaseClosingSqBracketsCount();
167 }
168 /**
169 * Creates a new TrajectoryPoint with collected x and y
170 * Clear the buffer
171 */
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[][] trajLCSSDistances;
21     private Double[][] clustLCSSDistances;
22     private int minX, maxX, minY, maxY;
23     private TrajectoryPoint cameraPoint;
24
25     public Clustering(List<Trajectory> trajectories) {
26         clusters = new ArrayList<>();
27         trajLCSSDistances = new Double[trajectories.size()][trajectories
28             .size()];
29         clustLCSSDistances = new Double[trajectories.size()][
30             trajectories.size()];
31     }
32
33
34     public Double[][] getTrajLCSSDistances() {
35         return trajLCSSDistances;
36     }
37
38     public void setTrajLCSSDistances(Double[][] trajLCSSDistances) {
39         this.trajLCSSDistances = trajLCSSDistances;
40         for (int i = 0; i < trajLCSSDistances.length; i++) {
41             System.arraycopy(
42                 trajLCSSDistances[i], 0,
43                 clustLCSSDistances[i], 0,
44                 trajLCSSDistances.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         // join i1 and i2 clusters, add i1 traj-es to cluster i2
107         clusters.get(id1).appendTrajectories(clusters.get(id2).
108 getTrajectories());
109         // recalculate D for i1 and i2 lines -> set i2 line all to
110         NULLS
111         recalclClustersDistMatrix(id1, id2);
112         // remove i2 from 'clusters'
113         clusters.remove(id2);
114         numClusters--;
115     }
116 }
117 /**
118 * Calculates LCSS distance for two input trajectories
119 * Smaller the LCSS distance - the better (0.0 - equal trajectories)
120 *
121 * @param t1 first trajectory
122 * @param t2 second trajectory
123 * @return LCSS distance for t1 and t2
124 */
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 }
```

```

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 }
```