МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему: Математическая модель управления транспортными потоками на основе нечеткой логики

Студент А.В. Заражевский _____

Руководитель Г.А. Тырыгина _____

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»
Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Заражевский Алексей Витальевич

1. Тема

Математическая модель управления транспортным потоком на основе нечеткой логики

2. Срок сдачи студентом законченной бакалаврской работы

24.06.2016

3. Исходные данные к бакалаврской работе

Модель управления транспортным потоком

4. Содержание бакалаврской работы (перечень подлежащих разработке вопросов, разделов)

- Введение
- Глава 1. Теоретические основы математического моделирования
- Глава 2 . Моделирование работы светофора
- Глава 3. Программная реализация модели
- Заключение
- Список используемой литературы

5. Ориентировочный перечень графического и иллюстративного материала

Презентация

6. Дата выдачи задания « 11 » января 2016 г.

Руководитель бакалаврской работы _____ Г.А. Тырыгина

Задание принял к исполнению _____ А.В. Заражевский

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента ___Заражевского Алексея Витальевича
по теме Математическая модель управления транспортными потоками на основе нечеткой логики

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Поиск материалов	8.02.2016	8.02.2016	Выполнено	
Составление библиографического списка	10.02.2016	10.02.2016	Выполнено	
Анализ материалов	12.02.2016	12.02.2016	Выполнено	
Изучение основных понятий нечеткого моделирования. Подготовка первой главы.	15.02.2016	15.02.2016	Выполнено	
Построение математической модели управления транспортным потоком. Подготовка второй главы.	7.03.2016	7.03.2016	Выполнено	

Разработка программы. Подготовка третьей главы.	21.04.2016	21.04.2016	Выполнено	
Предварительная защита	30.05.2016-11.06.2016	31.05.2016	Выполнено	
Проверка на наличие заимствований (плагиата) в системе antiplagiat.ru	17.06.2016	17.06.2016	Выполнено	
Сдача на кафедру отзыва научного руководителя и ознакомление с ним	20.06.2016	20.06.2016	Выполнено	
Сдача на кафедру комплекта документов для защиты	24.06.2016	24.06.2016	Выполнено	
Защита бакалаврской работы	27.06.2016-30.06.2016	29.06.2016	Выполнено	

Руководитель бакалаврской работы _____ Г.А. Тырыгина

Задание принял к исполнению _____ А.В. Заражевский

Аннотация

Тема: Математическая модель управления транспортными потоками на основе нечеткой логики

В связи с увеличение плотности транспортного потока на улицах городов приобретает актуальность организации регулирования потоков.

Объект исследования: транспортный поток мегаполиса.

Предмет: управления транспортным потоком на примере светофора.

Цель: программная реализация модели управления транспортным потоком.

Для этого необходимо решить следующие задачи:

- изучение аппарата нечеткой логики;
- изучение модели потока управления транспортных средств на основе аппарата нечеткой логики;
- программная реализация модели управления транспортным потоком на примере светофора.

Бакалаврская работа состоит из трех глав, введения и заключения.

Первая глава работы посвящена анализу различных классов математических моделей транспортных потоков. Также изучаются теоретические основы нечеткой логики.

Вторая глава описывает моделирование потока транспортных средств: алгоритм работы светофора, формулирование требований к программе, логика программы.

В третьей главе представлена программная реализация математической модели регулирования транспортного потока посредством управления нечетким светофором.

Бакалаврская работа представлена на 40 страницах, включает 12 иллюстраций, 2 таблицы, список используемой литературы содержит 21 источник.

Оглавление

Введение 3
Глава 1 Теоретические основы математического моделирования 5
1.1 Обзор математических моделей транспортных потоков 5
1.2 Основные принципы моделирования загрузки 7
1.3 Модели динамики транспортного потока 10
1.4 Нечеткая логика в моделировании 11
Глава 2 Моделирование работы светофора 14
2.1 Математическая модель светофора 14
2.2 Этапы работы системы управления 16
2.3 Формулирование требований к программе 23
Глава 3 Программная реализация модели 26
3.1 Описание программы 26
3.2 Программная реализация 28
Заключение 36
Список используемой литературы 38
Приложение А Листинг кода файла mainForm.cs 41

Введение

Транспортная инфраструктура — одна из важнейших инфраструктур любого города. В настоящее время наблюдается увеличение количества транспортных средств и, как следствие, увеличивается плотность транспортных потоков на улицах городов. В связи с невозможностью дальнейшего развития транспортной системы оживленных мегаполисов, организация дорожного движения приобретает высокий уровень актуальности. Пробки в основном образуются перед светофорами, перекрестками, пешеходными переходами и т.д. В связи с этим большое значение обретает задача регулирования транспортных потоков при помощи светофоров.

Объект бакалаврской работы: транспортный поток мегаполиса.

Предмет бакалаврской работы: управления транспортным потоком.

Целью бакалаврской работы является программная реализация модели управления транспортным потоком на примере светофора.

Задачами бакалаврской работы, исходя из поставленной цели, являются:

- изучение аппарата нечеткой логики;
- изучение модели потока управления транспортных средств на основе аппарата нечеткой логики;
- программная реализация модели управления транспортным потоком на примере светофора.

Первая глава работы посвящена анализу различных классов математических моделей транспортных потоков. Также изучаются теоретические основы нечеткой логики.

Вторая глава описывает моделирование потока транспортных средств: алгоритм работы светофора, формулирование требований к программе, логика программы.

В третьей главе представлена программная реализация математической модели регулирования транспортного потока посредством управления нечетким светофором.

Глава 1 Теоретические основы математического моделирования

1.1 Обзор математических моделей транспортных потоков

Транспортная инфраструктура — одна из важнейших инфраструктур, обеспечивающих жизнь городов и регионов. В последние десять лет во многих крупных городах наблюдается упадок возможности развития транспортных сетей. Поэтому возможность оптимизировать планирование сетей, организацию движения и системы маршрутов общественного транспорта становится необходимостью. Решение таких задач невозможно без математического моделирования. Главная ее задача — определение и прогнозирование всех параметров функционирования транспортной сети. К ним относятся: интенсивность движения на всех элементах сети, объемы перевозок в сети общественного транспорта, средние скорости движения, задержки и потери времени и т.д.

Математические модели анализа транспортных сетей различаются по решаемым задачам, математическому аппарату, используемым данным и степени детализации описания движения [1]. Поэтому классифицировать их в полной мере не удается. Но, основываясь на задачах, для решений которых модель применялась, можно условно разделить их на три класса:

- прогнозные модели;
- имитационные модели;
- оптимизационные модели.

Прогнозную модель можно определить следующими условиями: Пусть известны геометрия и характеристики транспортной сети и размещение потокообразующих объектов в городе. Цель модели: определить, характеристики движение транспортного потока. Конкретно такая модель включает в себя расчет объема межрайонных передвижений, интенсивность потока, распределение автомобилей и пассажиров по путям движения и прочее.

Результатом таких моделей будет прогнозирование последствий изменений в транспортной сети или в размещении объектов.

Имитационная модель воспроизводит движение, временное развитие процесса. Исходными данными таких моделей будут средние значения потоков и распределение по путям, поэтому они будут считаться известными. Отличие между этим двумя моделями состоит в том, что: прогнозирующая модель отвечает, сколько транспорта, и в каком направлении будет двигаться в данной сети, а имитационная модель раскрывает в деталях процесс движения, если известно в среднем, сколько транспорта, и в каком направлении будет двигаться в данной сети. Таким образом, прогнозирующие имитационные модели являются дополняющими друг друга. Кроме того, к имитационным моделям можно отнести широкий спектр моделей — модели динамики транспортного потока. В моделях этого класса может применяться разная техника, начиная от имитации движения отдельного автомобиля, и заканчивая описанием динамики плотности автотранспорта на различных участках дороги. Динамические модели обеспечивают большую детализацию описания движения, но требуют большого количества вычислительных ресурсов. Такие модели позволяют оценить динамику скорости движения, задержки на перекрестках, длины и динамику образования «очередей» или «заторов» и другие характеристики движения. Динамические имитационные модели нашли свое применение в таких областях, как улучшение организации движения, оптимизация светофорных циклов и прочее. В настоящее время актуальной задачей является разработка систем автоматизированного регулирования транспортным потоком в режиме реального времени. Такие системы используют информацию с датчиков в сочетании с динамическим имитационным моделированием. Развитием динамических моделей также интересуются ученые, рассматривая транспортный поток как физическое явление со сложными свойствами. Среди таких свойств — спонтанная потеря устойчивости, явления самоорганизации и коллективного поведении и прочие.

Целью моделей прогноза и имитации является воспроизведение транспортных потоков. Однако большое количество таких моделей предназначено для оптимизации функционирования транспортных сетей: задачи оптимизации маршрутов пассажирских и грузовых перевозок, выработки оптимальной конфигурации сети и прочее. Методы оптимизации представляют собой обширную область исследований.

1.2 Основные принципы моделирования загрузки

Транспортный поток складывается из отдельных участников совершаемых передвижения и пользователей транспортной сети. В понятие передвижения включается не только автотранспорт, но и пешеходное движение. Основные факторы распределения и количество передвижений это:

- потокообразующие факторы — размещение объектов, привлекающие движение, например жилые районы, места обслуживания и прочее;
- характеристики транспортной сети — количество и качество улиц и дорог, организация движения, маршруты общественного транспорта;
- поведенческие факторы — мобильность и личные предпочтения населения.

Формальное описание данных факторов при построении математической модели — транспортный граф. Его узлы соответствуют различным элементам транспортной сети, а дуги — улицам и линиям транспорта. К ним также относятся пересадки с уличного транспорта, на внеуличный транспорт. Отдельной составляющей транспортного графа является граф общественного транспорта. Его узлы обозначают остановочные пункты, а дуги — путь следования маршрута между остановками.

Для описания распределения потокообразующих объектов город делят на условные районы прибытия и отправления [3]. Каждый такой район включается в граф в виде узла, соединенный с обычными узлами специальными дугами.

Объем передвижений из одного района в другой называется межрайонной корреспонденцией. Пути передвижения в этом случае роли не играют.

Поведение пользователей моделируется при помощи формулировки критерия, на котором основывается цена альтернативного пути и способов передвижения. Этот критерий называют обобщенной ценой пути. Его увеличение снижает привлекательность пути. Обобщенная цена пути складывается из обобщенных цен всех входящих в него дуг. Также в цену пути можно добавить цену перехода с одной дуги на другую: цена поворота, цена посадки и прочее.

Обобщенная цена определяется суммой слагаемых, обозначающих факторы, влияющие на оценку пути. Обобщенная оценка включает в себя:

- время передвижения, которое зависит от скорости движения и загрузки данного участка;
- задержки на различных элементах сети (парковка, ожидание и прочее);
- денежная стоимость (платные дороги, платный въезд);
- условные штрафы по времени, зависящие от особенностей транспортной сети и самого транспорта.

Исследования говорят, что основной фактор определения цены пути — время. Остальные факторы являются корректирующими и могут условно быть выражены в минутах, которые добавляются ко времени передвижения. Поэтому путь с наименьшей обобщенной ценой часто называют просто кратчайшим путем.

Важнейшей особенностью формирования загрузки транспортной сети является тот факт, что выбор пути одними пользователями виляет на выбор пути другими пользователями. Математически такое влияние описывается функциями зависимости цены дуги от суммарного потока по этой дуге. Это создает обратную связь в формировании загрузки: выбор пути, основанный на сопоставлении цен различных путей, формирует загрузку, в то же время цены

различных путей определяются сформированной загрузкой. В реальных условиях транспортные потоки представляют собой равновесное состояние этого процесса.

Моделирование транспортных потоков в сети мегаполиса обычно подразумевает следующие четыре этапа:

- оценка общего объема прибытия и отправления из каждого района города (Trip generation);
- расщепление по способам передвижения: пешеходное движение, движение на общественном транспорте, передвижение на личном автомобиле и прочее (Modal split);
- определение матриц корреспонденций, определяющих объем передвижений между каждой парой районов (Trip distribution);
- определение матриц корреспонденций по транспортной сети — определение количества передвижений по каждому из путей, выбираемых участниками движения (Trip assignment).

Деление на этапы происходит лишь условно, так как они взаимосвязаны и не могут решаться как отдельные задачи. Большинство моделей расчета корреспонденций используют в качестве важного фактора обобщенные цены межрайонных передвижений. Расщепление передвижений по видам зависит от соотношения цен этих видов. Расчет корреспонденций и их расщепление может быть выполнено правильно при условии, что известна загрузка сети. Следовательно, задачу необходимо решать последовательными приближениями, повторяя все шаги.

Объемы передвижений между каждой парой условных районов образуют матрицу корреспонденций, которая служит количественной характеристикой структуры передвижения. Многообразие передвижений может быть разбито на разные группы передвижений по следующим критериям:

- по различию в целях передвижений;
- по различию в выборе способов передвижения;
- по различию в предпочтениях при выборе путей передвижения.

Для каждой группы передвижения рассчитывается своя матрица корреспонденций. Входной информацией к модели расчета корреспонденций являются общие объемы прибытия и отправления в каждом из районов прибытия и отбытия. Оценка объемов прибытий и отправлений по разным группам связана с пространственным размещением потокопорождающих объектов и подвижностью населения, то есть средним количеством поездок. Эта оценка строится на основе имеющихся демографических и социально-экономических данных и результатов обследований и в основном предшествует математическому моделированию.

1.3 Модели динамики транспортного потока

Большинство существующих моделей динамики транспортных потоков можно разделить на три класса:

- макроскопические;
- кинетические;
- микроскопические.

Макроскопическими называют модели, которые описывают движение автомобиля в усредненных терминах, таких как плотность, средняя скорость, поток и прочее. При таком подходе транспортный поток становится схож с течением жидкости, поэтому модели этого класса называют гидродинамическими.

В микроскопических моделях явно моделируется движение каждого автомобиля. Такой подход позволяет достичь более точного описания движения в отличие от макроописания, но это требует больших вычислительных ресурсов.

Кинетический подход является промежуточным звеном в классификации моделей. При таком подходе транспортный поток описывается плотность распределения автомобилей в фазовом пространстве. Динамика фазовой плотности описывается кинетическим уравнением. Уравнение основано на усредненном описании эффектов взаимодействия отдельных автомобилей. Этим данный подход ближе к микроуровню, чем к макроуровню. Значение кинетических моделей состоит в том, что на их основе можно систематически выводить макроскопические модели.

Особое место в классе микромоделей занимают модели клеточных автоматов (Cellural Automata), широко развивающиеся в последние годы. В таких моделях движение автомобиля во времени и пространстве принимается чрезвычайно упрощенным, что позволяет достичь высокой вычислительной эффективности.

1.4 Нечеткая логика в моделировании

Нечеткая логика представляет собой подход на основе «степени правды», а не принятым в булевой логике «истина» и «ложь» (0 и 1). Идея нечеткой логики впервые была выдвинута доктором Лотфи Заде из Университета Калифорнии в Беркли в 1960-е годы. Он работал над проблемой понимания компьютером естественного языка. Естественный язык трудно перевести в термины 0 и 1, как и большинство вещей в нашем мире. Все ли, в конечном счете, описывается нулем и единицей — вопрос философский, но на деле большинство данных, которые мы хотим предоставить компьютеру на обработку, находятся в состоянии между этими двумя числами [13].

Нечеткая логика использует 0 и 1 как крайние степени истинности, но и подразумевает промежуточные значения между ними.

Нечеткая логика близка к работе человеческого мозга. Мы собираем данные и формируем ряд частичных истин, которые агрегируем в дальнейшие значения истинности, которые, в свою очередь, при определенных

превышениях пороговых значений, вызывают побуждение к действию. Подобный процесс используется в искусственных компьютерных нейронных сетях и экспертных системах.

Обратим внимание, что двоичная логика является лишь частным случаем рассуждений, основанных на нечеткой логике [14].

Можно выделить несколько особенностей нечеткой логики [11,12]:

1. Нечеткая логика проста в понимании. Математические представление, лежащие в основе нечеткой логики просты и интуитивно поняты.
2. Нечеткая логика обладает гибкостью. Она легко включается в функционирующую систему без нужды переписывать ее заново.
3. Нечеткая логика терпима к нечетким данным. Если разобраться, то все в мире является неточным, и на основе этих пониманий строятся нечеткие рассуждения.
4. Нечеткая логика может моделировать нелинейные функции произвольной сложности. Можно создать любую нечеткую систему, чтобы соответствовать набору данных ввода-вывода.
5. Нечеткая логика основывается на личном опыте экспертов. В отличие от нейронных сетей, которые создают непрозрачные модели на основе обучения, нечеткая логика опирается на опыт людей, которые уже разбираются в данной системе.
6. Нечеткая логика может быть использована совместно с обычными методами управления. Для нечетких систем не обязательно менять привычные системы управления. Во многих случаях нечеткие системы дополняют и упрощают их реализацию.
7. Нечеткая логика основана на естественном языке. Основой нечеткой логики является человеческий язык, а поскольку нечеткая логика строится на структурах качественного описания, используемых в повседневной речи, то она проста в использовании.

Естественный язык, используемый людьми, формировался на протяжении тысячелетии, что может свидетельствовать о его эффективности.

Исходя из вышеперечисленных достоинств, можно сделать выбор в пользу использования нечеткой логики в моделировании. Нечеткая логика основывается на здравом смысле, и именно здравый смысл помогает сделать правильный выбор

Глава 2 Моделирование работы светофора

2.1 Математическая модель светофора

Процесс моделирования включает три элемента:

- субъект (исследователь);
- объект исследования;
- модель, определяющую отношения познающего субъекта и познаваемого объекта.

Первый этап моделирования подразумевает наличие знаний об объекте. Модель должна отражать свойства, характерные черты объекта в ходе эксперимента. Сходство объекта с моделью должно быть ограниченным, так как модель утрачивает свой смысл при абсолютном сходстве с объектом или при чрезмерном отличии от объекта. Из этого следует, что при изучении одних сторон объекта следует отказаться от других его сторон.

Второй этап включает в себя исследования модели, как самостоятельный объект. В ходе проведения экспериментов, в которых модель осознанно помещают в различные условия, получают данные о поведении модели. Конечным результатом считаются эти знания.

Третий этап — перенос знаний, полученных на прошлом этапе. Происходит переход от модели к объекту. Переход происходит по определенным правилам. Знания должны быть скорректированы с учетом свойств, которые ее были отражены в модели или были изменены.

На четвертом этапе происходит практическая проверка знаний, полученных на прошлых этапах, и их использование для построения обобщающей теории.

Постановка задачи.

Существует два типа светофоров, которые широко используются [7]. Первый тип использует фиксированное время цикла смены сигналов. Второй тип комбинирует фиксированное время цикла и наличие датчиков, которые

могу повлиять на время цикла. Если датчик фиксирует, что на участке дороги нет транспорта, то светофор меняет свет, пока транспорт снова не появится. Этот тип управления подразумевает некоторые знания о перекрестке, чтобы настроить время цикла и место расположения транспорта [15].

Нечеткая система управления светофором является альтернативной системой управления, которая может использоваться в более широких спектрах моделей перекрестков [10]. Такая система использует датчики для подсчета транспорта, а не просто фиксирует приближение транспорта. Это позволяет системе учитывать плотность транспортного потока, что позволяет лучше оценить изменение ситуации на дорогах. Поскольку распределение трафика меняется, нечеткая система управления может изменять световой сигнал соответствующим образом.

Общая структура нечеткой системы управления светофором изображена на рисунке 2.1

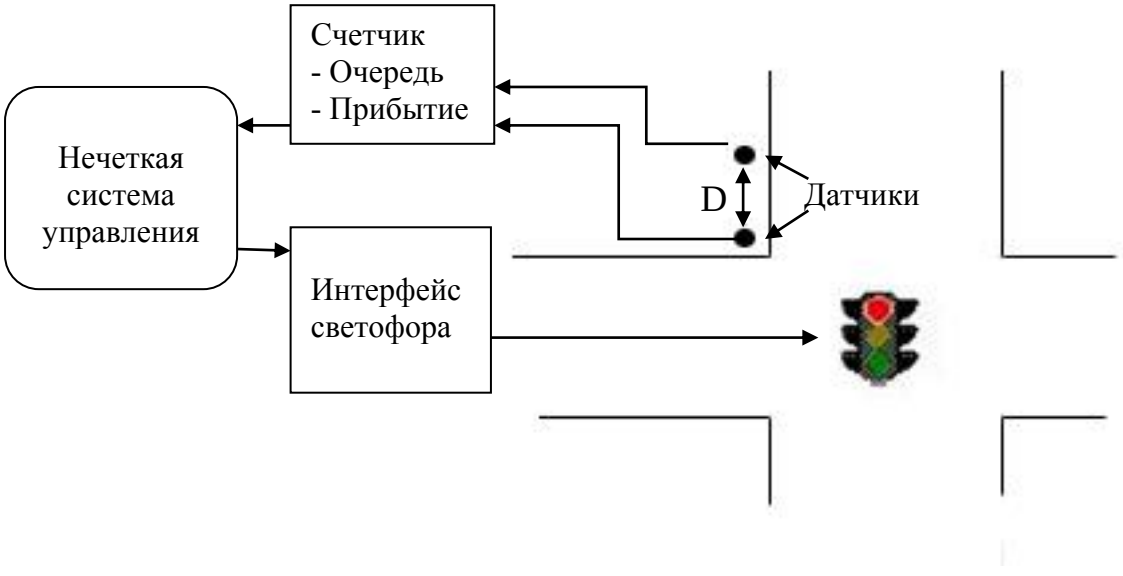

Рисунок 2.1 — Иллюстрация модели нечеткого управления транспортом на перекрестке

. Есть два датчика, расположенных на дороге для каждой полосы движения [16]. Первый датчик позади светофора подсчитывает количество автомобилей, подъезжающих на перекресток на расстоянии D от светофора. Количество автомобилей подсчитывается разницей между показаниями двух датчиков. Это отличает нечеткую систему управления от обычной системы, для которой установлен один датчик перед светофором, фиксирующий лишь приближение транспорта. Расстояние D определяется на основании схемы движения транспортного потока на конкретном участке перекрестке. Нечеткая система управления отвечает за продолжительность длины каждой фазы светофора. Существует одно состояние для каждой фазы светофора. Также существует значение по умолчанию для случая, когда транспорт отсутствует.

2.2 Этапы работы системы управления

Процесс управления транспортным потоком разделен на шаги, соответствующие одному циклу программы [4]. Каждый шаг включает в себя следующие этапы:

1. Определение четких входных переменных — определение количества машин на различных направлениях движения.
2. Фаззификация значений входных переменных — переход от натуральных значений к лингвистическим переменным.
3. Выборка решений на основе лингвистических переменных и ряда нечетких правил.
4. Дефаззификация значений — переход от лингвистических переменных к натуральным значениям.
5. Обновление данных.

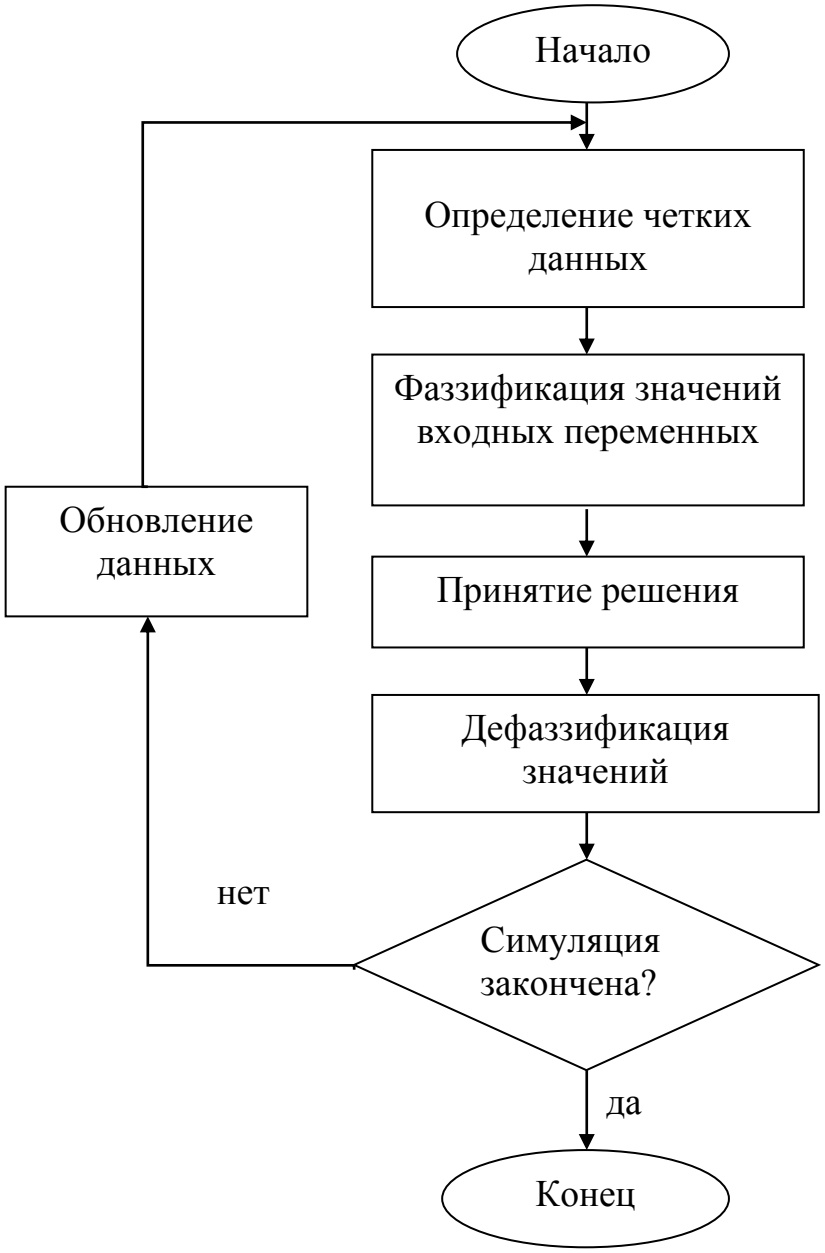

Рисунок 2.2 — Блок-схема этапов управления транспортным потоком

Необходимо разработать нечеткую логическую систему управления для изолированного перекрестка двух дорого: север-юг, запад-восток. В системе существует две входные переменные: количество трафика на стороне прибытия (Arrival) и количество трафика со стороны скапливающейся очереди (Queue) [17]. Допустим, зеленый горит для дороги север-юг, тогда эта дорога будет стороной прибытия, а дорога запад-восток — стороной скапливающейся очереди, и наоборот. Нечеткой выходной переменной будет время, на которое

необходимо увеличить продолжительность зеленого света. Таким образом, могут быть сформулированы правила нечеткого регулятора, которые будут принимать решение на основе текущих условий движения увеличивать продолжительность зеленого света или нет. Если увеличения нет, то светофор сразу меняет состояние, пропуская транспорт по противоположной стороне.

Для управления светофором есть четыре функции принадлежности для каждой из входных и выходных переменных [6]. В таблице 2.1 представлены термы лингвистических переменных и их сокращения.

Таблица 2.1 — Нечеткие переменные прибытия, очереди и увеличения времени в системе нечеткого управления

Прибытие		Очередь		Увеличение времени	
Почти ноль	ZP	Почти ноль	ZP	Ноль	Z
Мало	S	Мало	S	Малое	S
Средне	M	Средне	M	Среднее	M
Много	B	Много	B	Большое	L

Графическое представление функций принадлежности лингвистических переменных представлено на рисунке 2.2 . Можно заметить, что ось ординат является степенью числа членов каждой из нечеткой переменной. Для входных нечетких переменных ось абсцисс — показания датчиков, считываемые как количество автомобилей в штуках. Для выходных нечетких переменных ось абсцисс — отрезок времени, на который нужно продлить время данного состояния светофора в секундах.

Из рисунках 2.3 и 2.4 можно отметить, что функция принадлежности входных переменных «Прибытие» и «Очередь» равна единице для значения для значения «много» начиная с отметки 6, «средне» на отметке 4 и так далее.

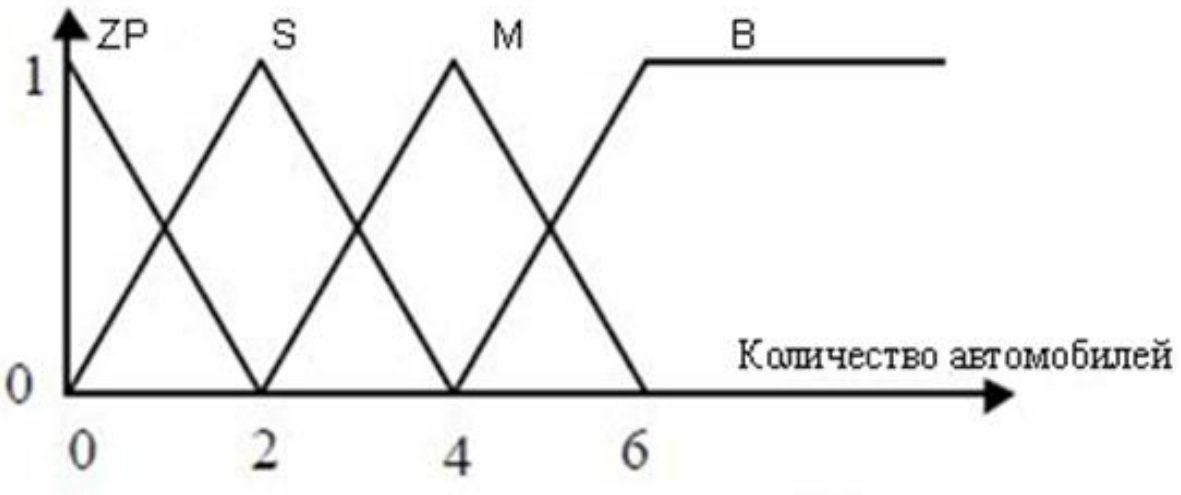

Рисунок 2.3 — График функции принадлежности входной переменной «Прибытие»

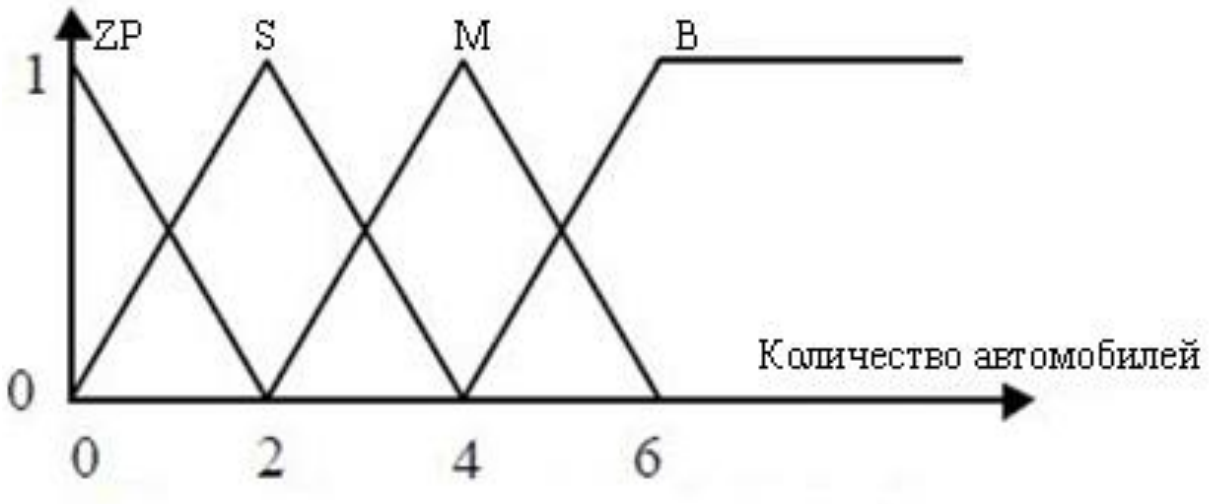

Рисунок 2.4 — График функции принадлежности входной переменной «Очередь»

График функции принадлежности выходной переменной «Увеличение времени» изображен на рисунке 2.5 [9]. Ее функция принадлежности равна единице для значения для значения «большое» начиная с 6 секунд, значение «среднее» находится в районе 4-х секунд и так далее. Конфигурация этих

функций принадлежности осуществляется в соответствии с экспертным наблюдением системы и окружающей среды.

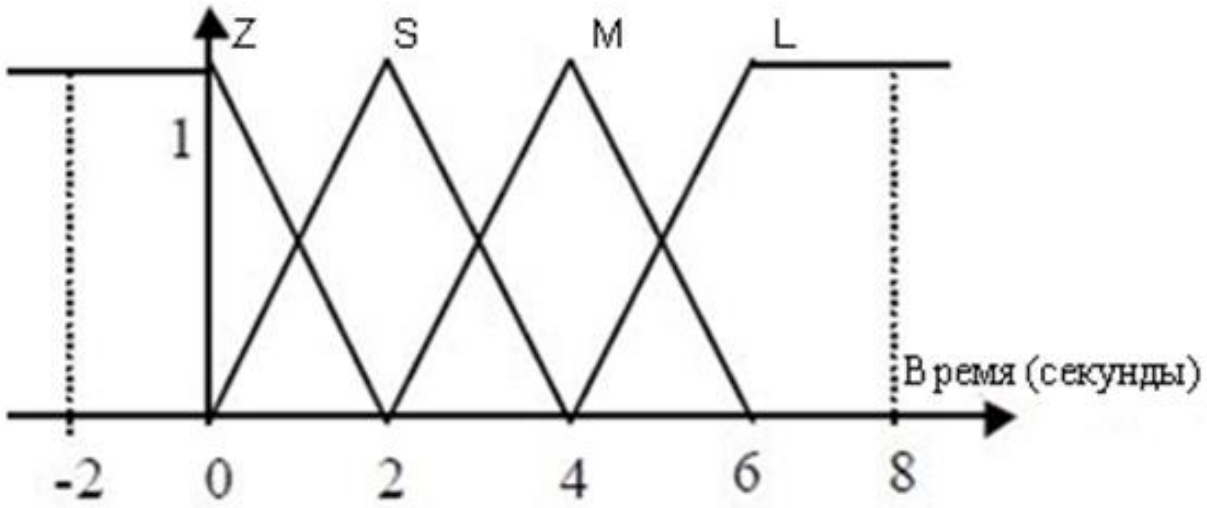

Рисунок 2.5 — График функции принадлежности выходной переменной «Увеличение времени»

В различных ситуациях ширина и центр функций принадлежности этих нечетких подмножеств могут быть изменены и настроены в соответствии с различными ситуациями и условиями движений[8]. Например, если перекресток слишком перегружен, подмножества входных переменных можно увеличить. И, наоборот, для менее загруженных перекрестков, ширина функции принадлежности может быть уменьшена. Следует отметить, что в управлении на нечеткой логике переход от одного нечеткого подмножества к другому осуществляется плавно, то есть необходимо перекрывать подмножествами друг друга. Если нет перекрытия, то управление будет напоминать ступенчатый контроль. С другой стороны, если перекрытий слишком много, то и нечеткости становится слишком много, что дает большую размытость в действиях системы управления. Нормальным перекрытием нечетких подмножеств считается примерно 25% перекрытия [6,18].

Механизм логического вывода в нечеткой логике системы управления напоминает процесс рассуждения человека [2]. Здесь происходит ассоциативная связь нечеткой логики с искусственным интеллектом. Люди неосознанно используют правила в своих действиях. Например, если транспорт на перекрестке будет регулировать сотрудник полиции, то он может руководствоваться с следующими соображениями:

ЕСЛИ поток с севера города БОЛЬШОЙ И поток с востока МАЛЕНЬКИЙ, ТО пропускать поток с севера ДОЛЬШЕ.

В другом случае он может решить:

ЕСЛИ поток с севера города СРЕДНИЙ И поток с востока СРЕДНИЙ, ТО пропускать поток с обеих сторон СРЕДНЕ.

Нечеткая логика позволяет приближенное значения в правилах такие, как МНОГО, МАЛО, СРЕДНЕ, НОРМАЛЬНО, БОЛЬШЕ и т.д. Благодаря установке степени значимости, описанных в лингвистических переменных, такие нечеткие решения можно предоставить компьютеру.

При разработке системы управления, основанной на нечеткой логике, используется примерно такие же правила:

Если машин со стороны прибытия много (B) и машин со стороны очереди мало (S), то время зеленой фазы сделать больше (L).

Если машин со стороны прибытия нет (PZ) и машин со стороны очереди мало (S), то время зеленой фазы не увеличивать (Z).

Обозначать эти правила можно следующим образом:

ЕСЛИ Прибытие — PZ И Очередь — PZ, ТО Увеличение времени — Z;

ЕСЛИ Прибытие — PZ И Очередь — S ТО Увеличение времени — Z;

ЕСЛИ Прибытие — PZ И Очередь — M, ТО Увеличение времени — Z;

ЕСЛИ Прибытие — PZ И Очередь — B, ТО Увеличение времени — Z;

ЕСЛИ Прибытие — S И Очередь — PZ, ТО Увеличение времени — S;

ЕСЛИ Прибытие — S И Очередь — S, ТО Увеличение времени — S;

ЕСЛИ Прибытие — S И Очередь — M, ТО Увеличение времени — Z;

ЕСЛИ Прибытие — S И Очередь — B, ТО Увеличение времени — Z;

ЕСЛИ Прибытие — M И Очередь — PZ, ТО Увеличение времени — M;

ЕСЛИ Прибытие — M И Очередь — S, ТО Увеличение времени — M;

ЕСЛИ Прибытие — M И Очередь — M, ТО Увеличение времени — S;

ЕСЛИ Прибытие — M И Очередь — B, ТО Увеличение времени — Z;

ЕСЛИ Прибытие — B И Очередь — PZ, ТО Увеличение времени — L;

ЕСЛИ Прибытие — B И Очередь — S, ТО Увеличение времени — M;

ЕСЛИ Прибытие — B И Очередь — M, ТО Увеличение времени — M;

ЕСЛИ Прибытие — B И Очередь — B, ТО Увеличение времени — S;

где «Прибытие» и «Очередь» — входные данные, а «Увеличение времени» зеленой фазы — логический вывод. Такие правила несложно составить, опираясь на условия движения на перекрестке. Для моделирования запишем их в матрицу, изображенную в таблице 2.2.

Таблица 2.2 — Конфигурация нечетких правил системы управления в матричной форме

	Прибытие				
Очередь		**PZ**	**S**	**M**	**B**
	PZ	Z	S	M	L
	S	Z	S	M	M
	M	Z	Z	S	M
	B	Z	Z	Z	S

Размер матрицы и количество правил равно количеству комбинаций функций принадлежности каждой из входных переменных. Например, в данном случае входных переменных две, каждая из которых содержит по четыре функции принадлежности, тогда количество правил равно шестнадцати. Иногда заполнять все правила в матрице не обязательно, но в данном случае, это необходимо.

В нечеткой системе управления, когда выполняется правило, функция принадлежности выходной переменной (в данном случае «Увеличение времени») устанавливается исходя из условий нечетких подмножеств (в данном случае «Прибытие» и «Очередь»). Нечеткая система управления с помощью максминой композиции находит степень истинности каждого из подзаключений правил. Когда степень значимости каждой нечеткой выходной переменной найдена, они объединяются в одно нечеткое множество. Дальше следует этап деффазификации, результатом которого получается действительное число [19].

2.3 Формулирование требований к программе

При разработке нечеткой системы управления светофором были сделаны следующие допущения:

- перекресток представляет собой изолированное пересечение четырех путей с поступающим транспортом с севера, запада, юга и востока;
- когда транспорт с юга и севера пересекает перекресток, транспорт с запада и востока останавливается и наоборот;
- для поворотов налево и направо не предусмотрено особых случаев;
- системе управления поступает информация о плотности трафика с каждой стороны перекрестка отдельно;
- дорога Север-Юг рассматривается, как главная.

Для создания модели перекрестка, соответствующей перечисленным свойствам, необходимо написать программу на языке программирования C#. Ее интерфейс демонстрируется на рисунке 2.6.

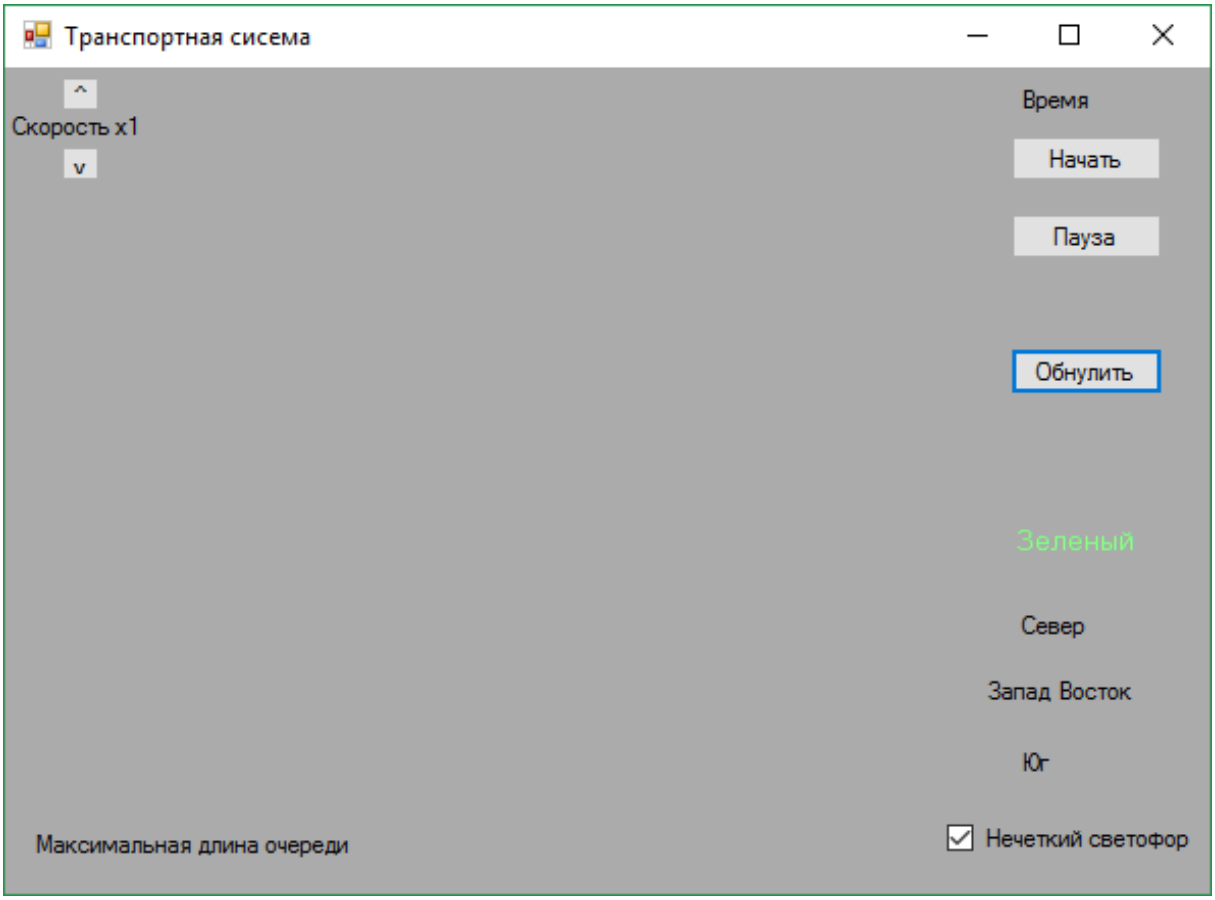

Рисунок 2.6 — Интерфейс модели транспортного потока

Сформулируем требования готовой программы. Необходимо:

- наглядное изображение ситуации на дороге в схематическом виде;
- знать количество единиц транспорта на каждом из подъездов к светофору и плотность движения на них;
- визуализировать переключение красного и зеленого света с обозначением выделенного времени на каждую из фаз (зеленую и красную) и времени суток;
- выделить кнопки управления для начала и приостановки симуляции движения;
- обнуление данных для начала новой симуляции движения;
- реализация возможности ручного изменения скорости моделирования;
- возможность переключения светофора в режим фиксированного значения;

- отображение максимального значения автотранспорта в очереди для фиксации обнаруженного затора.

Данная программа позволит моделировать регулирование движения на перекрестке посредством сигналов светофора с нечеткой системой управления.

Глава 3 Программная реализация модели

3.1 Описание программы

На рисунке 3.1 изображена программа, регулирующая движение транспортного потока посредством переключения сигналов светофора. Светофор меняет свет, основываясь на решении нечеткой системы управления, позволяя транспорту убывать с перекрестка только по одной из перпендикулярных дорог одновременно. Транспорт прибывает со всех сторон перекрестка неравномерно по заданной плотности движения потока.

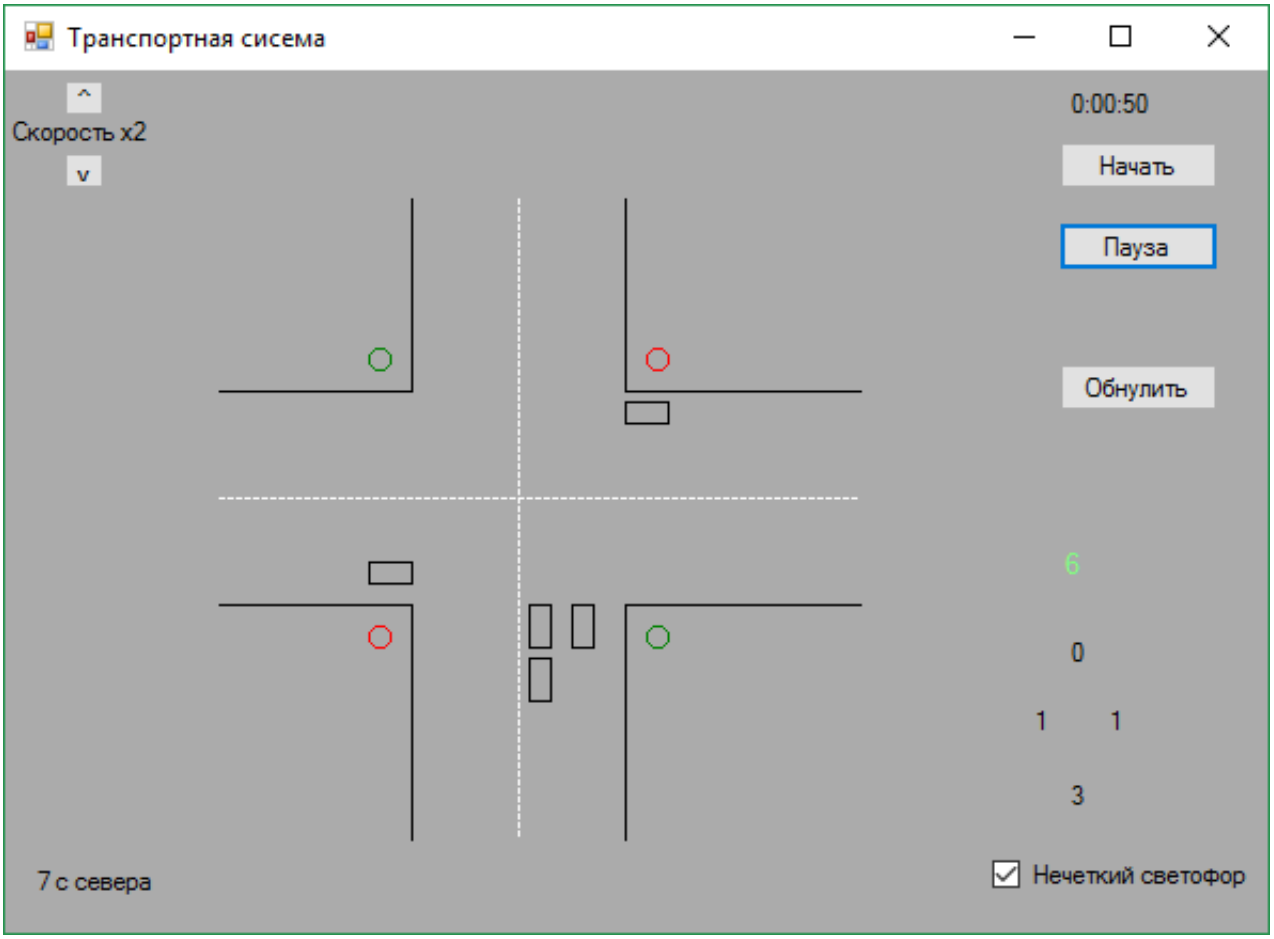

Рисунок 3.1 — Программная реализация модели управления транспортным потоком на перекрестке с помощью сигналов светофора

На основе поступающих данных о прибытии транспорта, программа решает, в каком направлении пропуск транспорта необходим больше в данный момент времени. В правом верхнем углу есть панель управления программой, включающая в себя три кнопки управления. Кнопка «Начать» запускает имитацию транспортной системы, позволяя наблюдать изменение в движении на перекрестке и в работе светофора. Кнопка «Пауза» позволяет приостановить работу программы, что позволяет проанализировать данные. После остановки программы таким образом можно без потерь данных продолжить работу программы кнопкой «Начать». Кнопка «Обнулить» останавливает работу программы, сбрасывая все данные. После чего имитацию можно начать только заново. Таймер в правом верхнем углу показывает условное время суток в симуляции.

Под кнопками управления находится зеленое число, отображающее остаток времени, выделенный на продолжительность зеленой фазы. Если система управления принимает решение об увеличении времени, число так же увеличивается.

Ниже отображены четыре числа, расположенные крестом. Это численное значение транспорта — разницы показаний, зафиксированных датчиками. Иными словами, это количество транспорта, ожидающего на перекрестке. Цифры расположены так, чтобы было понятно, с какой стороны сколько транспорт ожидает.

В правом нижнем углу есть переключатель «Нечеткий светофор». По умолчанию он включен. Этот переключатель позволяет выключить или включить нечеткую логику светофора для сравнения пропускной способности перекрестка при двух различных светофорах. В случае выключения системы нечеткого управления, время каждой фазы светофора будет фиксированным. Переключение режима светофорам таким образом не нарушает симуляцию.

В левом верхнем углу есть кнопки переключения скорости. Переключаясь между четырьмя различными скоростями, мы можем пропустить мало

интересующий нас участок времени и внимательнее рассмотреть тенденцию изменения потока на интересующем нас участке времени.

В левом нижнем углу отображается размер наибольшей очереди и сторона, с которой она была зафиксирована. Большой размер очереди горит о необходимости изменения функции принадлежности входных и выходных переменных.

В центре окна расположен блок иллюстрации. В нем мы видим перекресток с горящим в данный момент времени зеленым или красным светом с каждой стороны. Так же на подъезде к перекрестку схематично изображается очередь транспорта.

3.2 Программная реализация

Отображение объектов происходит с помощью среды CLR, использующую расширенную реализацию интерфейса графических устройств Windows (GDI — Graphics Device Interface) под названием GDI+.

Одно из преимуществ использования GDI вместо прямого доступа к оборудованию — это унификация работы с различными устройствами. Используя GDI, можно одними и теми же функциями рисовать на разных устройствах, таких, как экран или принтер, получая на них практически одинаковые изображения.

Часть кода, реализующая отрисовку автомобилей на перекрестке в программе, представлена на рисунке 3.2. Аналогичным образом происходит отрисовка остальных элементов иллюстрации в программе.

```
System.Drawing.Pen myPen;
myPen = new System.Drawing.Pen(System.Drawing.Color.Black);
System.Drawing.Graphics formGraphics = this.CreateGraphics();
formGraphics.Clear(SystemColors.AppWorkspace);
//автомобили
N = Cn / 10;
int x, y;
if(Cn != 0) N++;
for (int i = 0; i < N; i++)
{
    x = i % 2;
    y = i / 2;
    formGraphics.DrawRectangle(myPen, 195 + 20 * x, 130 - 25 * y, 10, 20);
}
S = Cs / 10;
if(Cs != 0) S++;
for (int i = 0; i < S; i++)
{
    x = i % 2;
    y = i / 2;
    formGraphics.DrawRectangle(myPen, 245 + 20 * x, 250 + 25 * y, 10, 20);
}
E = Ce / 10;
if (Ce != 0) E++;
for (int i = 0; i < E; i++)
{
    x = i / 2;
    y = i % 2;
    formGraphics.DrawRectangle(myPen, 290 + 25 * x, 155 + 20 * y, 20, 10);
}
W = Cw / 10;
if (Cw != 0) W++;
for (int i = 0; i < W; i++)
{
    x = i / 2;
    y = i % 2;
    formGraphics.DrawRectangle(myPen, 170 - 25 * x, 230 - 20 * y, 20, 10);
}
```

Рисунок 3.2 — Реализация графического отображения автомобилей

Для фаззификации входных данных и последующей дефаззификации выходных данных используется три нечетких множества. На рисунке 3.3 изображен фрагмент кода, хранящих нечеткие множества лингвистических переменных «очередь», «прибытие» и «увеличение».

```
carsQue[0] = new Tuple<string, int, int, int>("ZP", 0, 0, 2);
carsQue[1] = new Tuple<string, int, int, int>("S", 0, 2, 4);
carsQue[2] = new Tuple<string, int, int, int>("M", 4, 6, 8);
carsQue[3] = new Tuple<string, int, int, int>("B", 6, 8, 10);

carsArr[0] = new Tuple<string, int, int, int>("ZP", 0, 0, 2);
carsArr[1] = new Tuple<string, int, int, int>("S", 0, 2, 4);
carsArr[2] = new Tuple<string, int, int, int>("M", 4, 6, 8);
carsArr[3] = new Tuple<string, int, int, int>("B", 6, 8, 10);

timeSet[0] = new Tuple<string, int, int, int>("Z", 0, 0, 2);
timeSet[1] = new Tuple<string, int, int, int>("S", 0, 2, 4);
timeSet[2] = new Tuple<string, int, int, int>("M", 2, 4, 6);
timeSet[3] = new Tuple<string, int, int, int>("B", 4, 6, 8);
```

Рисунок 3.3 — Лингвистические переменные «очередь», «прибытие» и «увеличение времени»

Каждый элемент массива хранит четверку данных. Первое значение — буквенное обозначение. Второе и четвертое соответствуют абсциссе левой и правой сторон основания треугольника, а третье — значению абсциссы вершины треугольника. Этот массив может быть настроен под особенности конкретного перекрестка. Например, если плотность потока на данном перекрестке будет больше, то числа во всех трех массивах следует увеличить.

На рисунке 3.4 изображена функция, фаззифицирующая входные данные, полагаясь на вышеуказанные нечеткие множества. Получая на вход количество транспорта в очереди на обеих дорогах, функция определяет степень значимости по каждому из терм-множеств. Результаты фаззификации записываются в первую строку и первый столбец двумерного массива rules.

```
private void fuzzy(int arrive , int queue)
{
    double s;
    double p;
    double set;
    for (int i = 0; i <= 4; i++)
    {
        for (int j = 0; j <= 4; j++)
        {
            rules[i, j] = 0;
        }
    }
        for(int i = 0; i < 4; i++)
        {
             if ((arrive > carsArr[i].Item2) && (arrive <= carsArr[i].Item3))
             {
                 s = carsArr[i].Item3 - carsArr[i].Item2;
                 p = arrive - carsArr[i].Item2;
                 set = 1 / s * p;
                 if (rules[0, i + 1] < set) rules[0, i + 1] = set;
             }
             if ((arrive > carsArr[i].Item3) && (arrive <= carsArr[i].Item4))
             {
                 s = carsArr[i].Item4 - carsArr[i].Item3;
                 p = arrive - carsArr[i].Item3;
                 set = 1 - (1 / s * p);
                 if (rules[0, i + 1] < set) rules[0, i + 1] = set;

             }
             if ((queue > carsQue[i].Item2) && (queue <= carsQue[i].Item3))
             {
                 s = carsQue[i].Item3 - carsQue[i].Item2;
                 p = queue - carsQue[i].Item2;
                 set = 1 / s * p;
                 if (rules[i + 1, 0] < set) rules[i + 1, 0] = set;
             }
             if ((queue > carsQue[i].Item3) && (queue <= carsQue[i].Item4))
             {
                 s = carsQue[i].Item4 - carsQue[i].Item3;
                 p = queue - carsQue[i].Item3;
                 set = 1 - (1 / s * p);
                 if (rules[i + 1, 0] < set) rules[i + 1, 0] = set;
             }
        }

}
```

Рисунок 3.4 — Программная реализация фаззификации

На рисунке 3.5 изображена функция агрегирования. В результате ее выполнения двумерный массив rules заполняется степенями истинности по каждому из правил системы нечеткого вывода.

```
private void agr()
{
    for (int i = 1; i <= 4; i++)
    {
        for (int j = 1; j <= 4; j++)
        {
            if (rules[0, j] < rules[i, 0])
            {
                rules[i, j] = rules[0, j];
            }
            else
            {
                rules[i, j] = rules[i, 0];
            }
        }
    }

}
```

Рисунок 3.5 — Программная реализация агрегирования

На рисунке 3.6 изображена матрица правил. Основываясь на ее значениях и значениях функций принадлежности двумерного массива, полученного в результате агрегации, определяется степень истинности по каждому из условий.

```
string[,] fRules = {
                   {"Z", "S", "M", "L"},
                   {"Z", "S", "M", "M"},
                   {"Z", "Z", "S", "M"},
                   {"Z", "Z", "Z", "S"},
                   };
```

Рисунок 3.6 — Матрица правил в коде

Процесс определения функции принадлежности по каждой из выходных лингвистических переменных называется аккумуляцией. Его программная реализация изображена на рисунке 3.7.

```
private void accum()
{
    double max;
    for (int k = 0; k <= 3; k++)
    {
        outVals[k] = 0;
    }
    for(int k = 0; k <= 3; k++){
        max = 0;
        for (int i = 1; i <= 4; i++)
        {
            for (int j = 1; j <= 4; j++)
            {
                if (fRules[i - 1, j - 1] == timeSet[k].Item1)
                {
                    if (max < rules[i, j])
                    {
                        max = rules[i, j];
                    }
                }
            }
        }
        outVals[k] = max;
    }
}
```

Рисунок 3.7— Реализация аккумуляции в коде программы

Получив степень значимости выходной переменной по каждой из термов, остается только дефаззифицировать значение. Множество степеней значимости записывается в массив outVals. Функция дефаззификации этих значений изображена на рисунке 3.8. Дефаззификация происходит по методу центра тяжести для одноточечных множеств. Полученное число округляется до целого. После этого оно используется системой управления для увеличения зеленой фазы.

```
private int defuzz()
{
    double d1 = 0;
    double d2 = 0;
    double h;
    double s;
    double p;
    for (int k = 0; k < timeSet[3].Item4; k++)
    {
        d[k] = 0;
        for (int i = 0; i <= 3; i++)
        {
            h = 0;
            if ((k > timeSet[i].Item2) && (k <= timeSet[i].Item3))
            {
                s = timeSet[i].Item3 - timeSet[i].Item2;
                p = k - timeSet[i].Item2;
                h = 1 / s * p;
                if (outVals[i]!=0)
                {
                    if (outVals[i] > h) d[k] = h;
                    else d[k] = outVals[i];
                }
            }
            else
                if ((k > timeSet[i].Item3) && (k <= timeSet[i].Item4))
                {
                    s = (timeSet[i].Item4 - timeSet[i].Item3);
                    p = (k - timeSet[i].Item3);
                    h = 1 - (1 / s * p);
                    if (outVals[i] != 0)
                    {
                        if (outVals[i] > h) d[k] = h;
                        else d[k] = outVals[i];
                    }
                }
                else
                    if ((k > timeSet[i].Item3) && (k <= timeSet[i].Item4))
                    {
                        s = (timeSet[i].Item4 - timeSet[i].Item3);
                        p = (k - timeSet[i].Item3);
                        h = 1 - (1 / s * p);
                        if (outVals[i] != 0)
                        {
                            if (outVals[i] > h) d[k] = h;
                            else d[k] = outVals[i];
                        }
                    }
            }
        }
        for (int k = 0; k < timeSet[3].Item4; k++)
        {
            d1 += d[k]*k;
            d2 += d[k];
        }
        if (d2 == 0) return 0;
        double z = Math.Truncate(d1 / d2);
        return Convert.ToInt32(z);
    }
```

Рисунок 3.8 — Реализация дефаззификациия в коде программы

Вышеописанные этапы повторяются, после завершения каждой зеленой фазы. В случае если на выходе будет получено число ноль, то светофор переключается в другой режим, и выполняется аналогичные расчеты для перпендикулярной дороги до тех пор, пока не будет получен ноль на выходе функции дефаззификации.

Программа прекращает имитацию автоматически спустя условных двадцать четыре часа.

Заключение

В ходе работы были изучены различные классы моделей транспортных систем, более подробно рассмотрены основные принципы модели загрузки и модели динамики транспортного потока. Были изучены понятия нечеткой логики, нечеткого моделирования.

Составлена математическая модель управления светофором, основанная на принятии нечетких решений в условиях городского движения транспорта. Система управления светофором на нечеткой логике показывает себя лучше, чем светофор с фиксированным временем за счет своей гибкости. Гибкость достигается принятием решений об увеличении продолжительности зеленой фазы, основанных на количестве зафиксированного транспорта. Светофор с фиксированным временем устанавливает время зеленой фазы независимо от плотности потока. Система управления, принимающая решение на основе наличия транспорта увеличивает продолжительность зеленой фазы на фиксированную величину времени. Такая система является расширенной версией системы управления с фиксированным временем. В нечеткой системе управления значение увеличения времени не фиксировано и нечеткое. Число фиксируемых автомобилей также преобразуется в нечеткие значения. В дополнении к нечетким величинам, система управления руководствуется лингвистическими правилами, аналогичными человеческому мышлению.

Спроектирована и затем написана программа, автоматизирующая работу регулирования движения потока на перекрестке при помощи сигналов светофора. На вход программа получает четкое значение ожидающего транспорта, фаззифицирует это значение, после чего принимает решение об увеличении времени, основываясь на базе правил. Полученное нечеткое значение дефаззифицируется, и уже натуральное число отсылается системе управления светофором для увеличения длины фазы или ее смены. Программа в режиме нечеткой системы управления обеспечивает более высокую

пропускную способность с точки зрения времени ожидания транспорта и длины возникающей очереди с каждой из сторон перекрестка. Меньшее время ожидания способствует не только экономию топлива, но и уменьшает загрязнение окружающей среды.

Список используемой литературы

1. Гасников, А.В. Введение в математическое моделирование транспортных потоков: учеб. пособие / А.В. Гасников, С.Л. Кленов, Е.А. Нурминский, Я.А. Холодов, Н.Б. Шамрай. — М.: МФТИ, 2010. — 362с.

2. Минаев, Ю.Н. Методы и алгоритмы решения задач индексации и прогнозирования в условиях неопределенности в нейросетевом логическом базисе: учеб. пособие / Ю.Н. Минаев, О.Ю. Филомонова, Б. Лиес. — М.: Горячая линия – Телеком, 2006. — 494с.

3. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы: учеб. пособие / Д. Рутковская, М. Пилиньский, Л. Рутковский. — М.: Горячая линия – Телеком, 2013. — 452с.

4. Круглов, В.В. Нечеткая логика и искусственные нейронные сети: учеб. пособие / В.В. Круглов, М.И. Дли, Р.Ю. Голунов. — М.: Физматлит, 2008. — 224с.

5. Пивкин, В.Я Нечеткие множества в системах управления: методическое пособие / В.Я. Пивкин, Е.П. Бакулин, Д.И. Кореньков, 2006. — 38с.

6. Бураков, М.В. Нечеткие регуляторы: учеб. пособие. / М.В. Бураков. — СПб.: ГУАП, 2010. — 237с.

7. Рыбин, В.В. Основы теории нечетких множеств и нечеткой логики: учеб. пособие / В.В. Рыбин. — М.: МАИ, 2007. — 96с.

8. Алтунин, А.Е. Модели и алгоритмы принятия решений в нечетких условиях: учеб. пособие / А.Е. Алтунин, М.В. Семухин. — Т.: Издательство Тюменского гос. ун-та, 2009. — 352с.

9. Зайченко, Ю.П. Нечеткие модели и методы в интеллектуальных системах: учеб. пособие / Ю.П. Зайченко. — К.: Слово, 2008. — 392с.

10. Пегат, А. Нечеткое моделирование и управление: учеб. пособие / А. Пегат. — М.: Бином, 2013. — 798с.

11. Батыршин, И.З. Основные операции нечеткой логики и их обобщения: учеб. пособие / И.З. Батыршин. — К.: Отечество, 2011. — 100с.

12. Павлов, А.Н. Принятие решений в условиях нечеткой логики: учеб. пособие / А. Н. Павлов, Б.В. Соколов. — СПб.: ГУАП, 2006. — 72с.

13. Круглов, В.В. Два подхода к самоорганизации базы правил системы нечеткого логического вывода. / В.В. Круглов, А.А. Усков. — 2006. [Электронный ресурс] Режим доступа: http://www.uskov.net/files/Uskov%20DPSO%202006%20.pdf

14. Новак, В. Математические принципы нечеткой логики / В. Новак, И.Г. Перфильева, И. Мочкорж. — 2006. [Электронный ресурс] Режим доступа: http://works.tarefer.ru/46/100085/index.html

15. Murat, Y.S. A New Approach for Fuzzy Traffic Signal Controls. / Y. S. Murat, E. Gedizlioglu. – Turkey: Pamukkale University, 2011. [Электронный ресурс] Режим доступа: http://www.iasi.cnr.it/ewgt/13conference/31_murat.pdf

16. Nellore, K. A Survey on Urban Traffic Management System Using Wireless Sensor Networks / K. Nellore, G. P. Hancke. Pretoria:University of Pretoria, 2016. [Электронный ресурс] Режим доступа: http://www.mdpi.com/1424-8220/16/2/157/pdf

17. Ge, Y. A Two-Stage Fuzzy Logic Control Method of Traffic Signal Based on Traffic Ungency Degree / Qingdao : Qingdao University of Science and Technology, 2014. [Электронный ресурс] Режим доступа: http://www.hindawi.com/journals/mse/2014/694185/

18. Alam, J. Intelligent Traffic Light Control System for Isolated Intersection Using Fuzzy Logic / J. Alam, M.K. Pandey, H. Ahmed. — Conference on Advances in Communication and Control Systems, 2013. [Электронный ресурс] Режим доступа: http://www.atlantis-press.com/php/download_paper.php?id=6306

19. Yusof, R. Intelligent Traffic Lights Control by Fuzzy Logic / Malaysia: Journal of Computer Science, 2007. [Электронный ресурс] Режим доступа: https://www.researchgate.net/publication/229029935

20. Taha, M.A. Traffic Simulation System Based on Fuzzy Logic / M. A. Taha, L. Ibrahim . — Procedia Computer Science. Washington, 2012 [Электронный ресурс] Режим доступа: http://www.sciencedirect.com/science/article/pii/S1877050912006758

Приложение А

Листинг кода файла mainForm.cs

```
sing System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace Cross
{
    public partial class Form1 : Form
    {
        int N = 0, S = 0, E = 0, W = 0;
        int Cn = 0, Cs = 0, Ce = 0, Cw = 0, Cn1 = 0, Cs1 = 0, Ce1 = 0, Cw1 = 0, Cn2 = 0,
Cs2 = 0, Ce2 = 0, Cw2 = 0;
        int hour = 0, minute = 0, sec = 0;
        int max;
        bool lights;
        bool fuzz = true;
        int lightTime = 10;
        Random rnd = new Random();
        Tuple<string, int, int, int>[] carsQue = new Tuple<string, int, int, int>[4];
        Tuple<string, int, int, int>[] carsArr = new Tuple<string, int, int, int>[4];
        Tuple<string, int, int, int>[] timeSet = new Tuple<string, int, int, int>[4];
        double[] outVals = new double[4];
        double[] d = new double[8];
        double[,] rules = new double[5, 5];
        string[,] fRules = {
                            {"Z", "S", "M", "L"},
                            {"Z", "S", "M", "M"},
                            {"Z", "Z", "S", "M"},
                            {"Z", "Z", "Z", "S"},
                            };
        float[] v = new float[5];
        static string connStr = @"Data Source=(local)\SQLEXPRESS;
                             Initial Catalog=transport;
                             User Id=root;
                             Password=root;";
        SqlConnection conn = new SqlConnection(connStr);
        int[] CarsN = new int[] {6, 8, 6, 3, 7, 4, 9, 6, 8, 7, 7, 7, 8, 7, 8, 4, 6, 6, 5,
6, 8, 8, 4, 2};
        int[] CarsS = new int[] {7, 7, 9, 4, 4, 4, 7, 4, 4, 6, 4, 4, 5, 6, 6, 5, 6, 6, 6,
4, 3, 3, 3, 2};
        int[] CarsE = new int[] {8, 5, 4, 5, 5, 7, 5, 6, 7, 7, 6, 3, 4, 4, 5, 6, 7, 5, 3,
3, 4, 4, 3, 3};
        int[] CarsW = new int[] {6, 3, 6, 6, 6, 4, 6, 6, 6, 7, 4, 5, 5, 3, 3, 5, 3, 2, 2,
5, 5, 4, 3, 2};
        public Form1()
        {
            InitializeComponent();
            try
            {
                conn.Open();
            }
```

```
            catch (SqlException se)
            {
                Console.WriteLine("Ошибка подключения:{0}", se.Message);
                return;
            }
            Console.WriteLine("Соедение успешно произведено");
            SqlCommand cmd = new SqlCommand("UPDATE dbo.detectors SET North = 0, North2 =
0, South = 0, South2 = 0, East =0, East2 = 0, West = 0, West2 = 0;", conn);
            Console.WriteLine("Вставляем запись");
            try
            {
                cmd.ExecuteNonQuery();
            }
            catch
            {
                Console.WriteLine("Ошибка, при выполнении запроса на добавление записи");
                return;
            }
            carsQue[0] = new Tuple<string, int, int, int>("ZP", 0, 0, 2);
            carsQue[1] = new Tuple<string, int, int, int>("S", 0, 2, 4);
            carsQue[2] = new Tuple<string, int, int, int>("M", 4, 6, 8);
            carsQue[3] = new Tuple<string, int, int, int>("B", 6, 8, 10);

            carsArr[0] = new Tuple<string, int, int, int>("ZP", 0, 0, 2);
            carsArr[1] = new Tuple<string, int, int, int>("S", 0, 2, 4);
            carsArr[2] = new Tuple<string, int, int, int>("M", 4, 6, 8);
            carsArr[3] = new Tuple<string, int, int, int>("B", 6, 8, 10);

            timeSet[0] = new Tuple<string, int, int, int>("Z", 0, 0, 2);
            timeSet[1] = new Tuple<string, int, int, int>("S", 0, 2, 4);
            timeSet[2] = new Tuple<string, int, int, int>("M", 2, 4, 6);
            timeSet[3] = new Tuple<string, int, int, int>("B", 4, 6, 8);
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            sec++;
            if (sec >= 60)
            {
                minute += 1;
                sec -= 60;
                if (minute >= 60)
                {
                    hour += 1;
                    minute -= 60;
                    if (hour >= 24)
                    {
                        timer1.Stop();
                        hour -= 1;
                    }
                }
            }
            if (rnd.Next(0, 10-CarsN[hour]) == 0) Cn2++;
            if (rnd.Next(0, 10-CarsE[hour]) == 0) Ce2++;
            if (rnd.Next(0, 10-CarsW[hour]) == 0) Cw2++;
            if (rnd.Next(0, 10-CarsS[hour]) == 0) Cs2++;
            SqlCommand cmd = new SqlCommand("UPDATE dbo.detectors SET North2 = " + Cn2 +
", South2 = " + Cs2 + ", East2 =" + Ce2 + ", West2 = " + Cw2 + ";", conn);
            try
            {
                cmd.ExecuteNonQuery();
```

```
}
catch
{
    Console.WriteLine("Ошибка, при выполнении запроса на добавление записи");
    return;
}
if (lights)
{
    Cn1 ++;
    if (Cn1 > Cn2) Cn1 = Cn2;
    Cs1 ++ ;
    if (Cs1 > Cs2) Cs1 = Cs2;
}
if (!lights)
{
    Ce1 ++ ;
    if (Ce1 > Ce2) Ce1 = Ce2;
    Cw1 ++ ;
    if (Cw1 > Cw2) Cw1 = Cw2;
}
getCars();
lightTime--;
if (max < Ce)
{
    max = Ce;
    label8.Text = max + " с востока";
}
if (max < Cw)
{
    max = Cw;
    label8.Text = max + " с запада";
}
if (max < Cn)
{
    max = Cn;
    label8.Text = max + " с севера";
}
if (max < Cs)
{
    max = Cs;
    label8.Text = max + " с юга";
}
if (lightTime == 0 && !lights)
{
    if (fuzz)
    {
        fuzzy((Cw + Ce), (Cn + Cs));
        agr();
        accum();
        lightTime = defuzz();
        if (lightTime == 0)
        {
            lights = true;
            lightTime = 5;
        }
    }
    else
    {
        lights = true;
        lightTime = 20;
    }
}
if (lightTime == 0 && lights)
```

```
{
    if (fuzz)
    {
        fuzzy((Cn + Cs), (Cw + Ce));
        agr();
        accum();
        lightTime = defuzz();
        Console.WriteLine(lightTime);
        if (lightTime == 0)
        {
            lights = false;
            lightTime = 5;
        }
    }
    else
    {
            lights = false;
            lightTime = 20;
    }
}
System.Drawing.Pen myPen;
myPen = new System.Drawing.Pen(System.Drawing.Color.Black);
System.Drawing.Graphics formGraphics = this.CreateGraphics();
formGraphics.Clear(SystemColors.AppWorkspace);
//автомобили
N = Cn / 10;
int x, y;
if(Cn != 0) N++;
for (int i = 0; i < Cn; i++)
{
    x = i % 2;
    y = i / 2;
    formGraphics.DrawRectangle(myPen, 195 + 20 * x, 130 - 25 * y, 10, 20);
}
S = Cs / 10;
if(Cs != 0) S++;
for (int i = 0; i < Cs; i++)
{
    x = i % 2;
    y = i / 2;
    formGraphics.DrawRectangle(myPen, 245 + 20 * x, 250 + 25 * y, 10, 20);
}
E = Ce / 10;
if (Ce != 0) E++;
for (int i = 0; i < Ce; i++)
{
    x = i / 2;
    y = i % 2;
    formGraphics.DrawRectangle(myPen, 290 + 25 * x, 155 + 20 * y, 20, 10);
}
W = Cw / 10;
if (Cw != 0) W++;
for (int i = 0; i < Cw; i++)
{
    x = i / 2;
    y = i % 2;
    formGraphics.DrawRectangle(myPen, 170 - 25 * x, 230 - 20 * y, 20, 10);
}
if (minute < 10) label5.Text = hour + ":0" + minute + ":" + sec;
else label5.Text = hour + ":" + minute + ":" + sec;
label1.Text = Cn.ToString();
label2.Text = Cw.ToString();
label3.Text = Ce.ToString();
```

```
            label4.Text = Cs.ToString();
            label7.Text = lightTime.ToString();
            formGraphics.DrawLine(myPen, 100, 250, 190, 250);
            formGraphics.DrawLine(myPen, 290, 250, 400, 250);
            formGraphics.DrawLine(myPen, 100, 150, 190, 150);
            formGraphics.DrawLine(myPen, 290, 150, 400, 150);

            formGraphics.DrawLine(myPen, 290, 250, 290, 360);
            formGraphics.DrawLine(myPen, 190, 250, 190, 360);
            formGraphics.DrawLine(myPen, 290, 60, 290, 150);
            formGraphics.DrawLine(myPen, 190, 60, 190, 150);
            //светофоры
            if (lights) myPen = new System.Drawing.Pen(System.Drawing.Color.Green);
            else myPen = new System.Drawing.Pen(System.Drawing.Color.Red);
            formGraphics.DrawEllipse(myPen, 300, 260, 10, 10);
            formGraphics.DrawEllipse(myPen, 170, 130, 10, 10);

            if (!lights) myPen = new System.Drawing.Pen(System.Drawing.Color.Green);
            else myPen = new System.Drawing.Pen(System.Drawing.Color.Red);
            formGraphics.DrawEllipse(myPen, 300, 130, 10, 10);
            formGraphics.DrawEllipse(myPen, 170, 260, 10, 10);
            //разметка
            myPen = new System.Drawing.Pen(System.Drawing.Color.White);
            myPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
            formGraphics.DrawLine(myPen, 100, 200, 400, 200);
            formGraphics.DrawLine(myPen, 240, 60, 240, 360);
            myPen.Dispose();
            formGraphics.Dispose();
            cmd = new SqlCommand("UPDATE dbo.detectors SET North =" + Cn1 +", South = " +
Cs1 + ", East ="+ Ce1 + ", West = " + Cw1 +";", conn);
            try
            {
                cmd.ExecuteNonQuery();
            }
            catch
            {
                Console.WriteLine("Ошибка, при выполнении запроса на добавление записи");
                return;
            }
            cmd = new SqlCommand("UPDATE dbo.detectors SET North =" + Cn1 + ", South = "
+ Cs1 + ", East =" + Ce1 + ", West = " + Cw1 + ";", conn);
            try
            {
                cmd.ExecuteNonQuery();
            }
            catch
            {
                Console.WriteLine("Ошибка, при выполнении запроса на добавление записи");
                return;
            }

        }
        private void button1_Click(object sender, EventArgs e)
        {
            timer1.Start();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            timer1.Stop();
        }
        private void button3_Click(object sender, EventArgs e)
        {
            timer1.Stop();
```

```
    System.Drawing.Pen myPen;
    myPen = new System.Drawing.Pen(System.Drawing.Color.Black);
    System.Drawing.Graphics formGraphics = this.CreateGraphics();
    formGraphics.Clear(SystemColors.AppWorkspace);
    hour = 0; minute = 0; sec = 0;
    lights = false;
    lightTime = 20;
    max = 0;
    Cn = 0; Cw = 0; Cs = 0; Ce = 0;
     label5.Text = "Время";
    label8.Text = "Максимальная длина очереди";
    label7.Text = "Зеленый";
    label1.Text = "Север";
    label2.Text = "Запад";
    label3.Text = "Восток";
    label4.Text = "Юг";
}
private void fuzzy(int arrive, int queue)
{
    double s;
    double p;
    double set;
    for (int i = 0; i <= 4; i++)
    {
        for (int j = 0; j <= 4; j++)
        {
            rules[i, j] = 0;
        }
    }
    for (int i = 0; i < 4; i++)
    {
        if ((arrive > carsArr[i].Item2) && (arrive <= carsArr[i].Item3))
        {
            s = carsArr[i].Item3 - carsArr[i].Item2;
            p = arrive - carsArr[i].Item2;
            set = 1 / s * p;
            if (rules[0, i + 1] < set) rules[0, i + 1] = set;
        }
        if ((arrive > carsArr[i].Item3) && (arrive <= carsArr[i].Item4))
        {
            s = carsArr[i].Item4 - carsArr[i].Item3;
            p = arrive - carsArr[i].Item3;
            set = 1 - (1 / s * p);
            if (rules[0, i + 1] < set) rules[0, i + 1] = set;

        }
        if ((queue > carsQue[i].Item2) && (queue <= carsQue[i].Item3))
        {
            s = carsQue[i].Item3 - carsQue[i].Item2;
            p = queue - carsQue[i].Item2;
            set = 1 / s * p;
            if (rules[i + 1, 0] < set) rules[i + 1, 0] = set;
        }
        if ((queue > carsQue[i].Item3) && (queue <= carsQue[i].Item4))
        {
            s = carsQue[i].Item4 - carsQue[i].Item3;
            p = queue - carsQue[i].Item3;
            set = 1 - (1 / s * p);
            if (rules[i + 1, 0] < set) rules[i + 1, 0] = set;
        }
    }
}
private void agr()
```

```
{
    for (int i = 1; i <= 4; i++)
    {
        for (int j = 1; j <= 4; j++)
        {
            if (rules[0, j] < rules[i, 0])
            {
                rules[i, j] = rules[0, j];
            }
            else
            {
                rules[i, j] = rules[i, 0];
            }
        }
    }
}
private void accum()
{
    double max;
    for (int k = 0; k <= 3; k++)
    {
        outVals[k] = 0;
    }
    for(int k = 0; k <= 3; k++){
        max = 0;
        for (int i = 1; i <= 4; i++)
        {
            for (int j = 1; j <= 4; j++)
            {
                if (fRules[i - 1, j - 1] == timeSet[k].Item1)
                {
                    if (max < rules[i, j])
                    {
                        max = rules[i, j];
                    }
                }
            }
        }
        outVals[k] = max;
    }
}
private int defuzz()
{
    double d1 = 0;
    double d2 = 0;
    double h;
    double s;
    double p;
    for (int k = 0; k < timeSet[3].Item4; k++)
    {
        d[k] = 0;
        for (int i = 0; i <= 3; i++)
        {
            h = 0;
            if ((k > timeSet[i].Item2) && (k <= timeSet[i].Item3))
            {
                s = timeSet[i].Item3 - timeSet[i].Item2;
                p = k - timeSet[i].Item2;
                h = 1 / s * p;
                if (outVals[i]!=0)
                {
                    if (outVals[i] > h) d[k] = h;
                    else d[k] = outVals[i];
```

```
                }
            }
            else
                if ((k > timeSet[i].Item3) && (k <= timeSet[i].Item4))
                {
                    s = (timeSet[i].Item4 - timeSet[i].Item3);
                    p = (k - timeSet[i].Item3);
                    h = 1 - (1 / s * p);
                    if (outVals[i] != 0)
                    {
                        if (outVals[i] > h) d[k] = h;
                        else d[k] = outVals[i];
                    }
                }
        }
    }
    for (int k = 0; k < timeSet[3].Item4; k++)
    {
        d1 += d[k]*k;
        d2 += d[k];
    }
    if (d2 == 0) return 0;
    double z = Math.Truncate(d1 / d2);
    return Convert.ToInt32(z);
}
private void getCars()
{
    SqlConnection conn = new SqlConnection(connStr);
    try
    {
        conn.Open();
    }
    catch (SqlException se)
    {
        Console.WriteLine("Ошибка подключения:{0}", se.Message);
        return;
    }
    SqlCommand cmd = new SqlCommand("SELECT * FROM detectors", conn);
    using (SqlDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection))
    {
        while (dr.Read())
        {
            Cn = dr.GetInt32(1) - dr.GetInt32(0);
            Cs = dr.GetInt32(3) - dr.GetInt32(2);
            Ce = dr.GetInt32(5) - dr.GetInt32(4);
            Cw = dr.GetInt32(7) - dr.GetInt32(6);
        }
    }
}
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        fuzz = true;
    }
    else
    {
        fuzz = false;
    }
}
private void button4_Click(object sender, EventArgs e)
{
    switch (timer1.Interval)
```

```
            {
                case 500:
                    {
                        label10.Text = "Скорость x1";
                        timer1.Interval = 1000;
                        break;
                    }
                case 100:
                    {
                        label10.Text = "Скорость x2";
                        timer1.Interval = 500;
                        break;
                    }
                case 10:
                    {
                        label10.Text = "Скорость x3";
                        timer1.Interval = 100;
                        break;
                    }
            }
        }
        private void button5_Click(object sender, EventArgs e)
        {
            switch(timer1.Interval)
            {
                case 1000:
                    {
                        label10.Text = "Скорость x2";
                        timer1.Interval = 500;
                        break;
                    }
                case 500:
                    {
                        label10.Text = "Скорость x3";
                        timer1.Interval = 100;
                        break;
                    }
                case 100:
                    {
                        label10.Text = "Скорость x4";
                        timer1.Interval = 10;
                        break;
                    }
            }
        }
    }
}
```