МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК

Заведующий кафедрой д-р физ.-мат. н., проф.

\_\_\_\_\_\_\_\_ Миков А. И.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_ 2016 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**ОРГАНИЗАЦИЯ БАЗЫ ЗНАНИЙ О СВОЙСТВАХ**

**СТОХАСТИЧЕСКИХ МОДЕЛЕЙ БЕСПРОВОДНЫХ**

**КОМПЬЮТЕРНЫХ AD HOC СЕТЕЙ**

Работу выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Федчун О.В.

(подпись, дата)

Факультет компьютерных технологий и прикладной математики, 4 курс Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Научный руководитель д-р физ.-мат. н., проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Миков А.

И.

(подпись, дата)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фисун

Ю.С. (подпись, дата)

Краснодар 2016

СОДЕРЖАНИЕ

Введение......................................................................................................... 5

1. Анализ задачи ..........................
   1. Основные характеристики ad hoc сети ..................................... 8
      1. Вероятность связности сети .......................................... 8
      2. Математическое ожидание степени узла ..................... 9
      3. Математическое ожидание количества связей в сети 10
      4. Распределение степеней узлов ................................... 11
   2. Организация базы знаний ......................................................... 11
      1. СУБД Oracle ................................................................ 13
      2. СУБД MS SQL Server ................................................. 15
      3. Сравнение Oracle и MS SQL Server .......................... 17
      4. Обоснование выбора MS SQL Server ....................... 19
   3. Хранение и формирование формул и отчётов ....................... 19
      1. Хранение в базе знаний .............................................. 19
      2. Отображение ............................................................... 22
2. Проектирование....................... 20
   1. Требования к программе .......................................................... 23
   2. Проектирование таблиц базы данных. .................................... 26
      1. Таблица EXPERIMENTS ........................................... 26
      2. Таблица POSSIBILITY\_DISTRIBUTION ................. 29
      3. Таблица DYNAMIC .................................................... 29
      4. Таблица TEMPLATES ................................................ 31
      5. Таблица STRINGS ...................................................... 33
3. Реализация ......................................................................................... 35
   1. Реализация базы данных средствами Microsoft SQL Server . 36
   2. Класс описания узлов сети ....................................................... 36
   3. Класс описания сети ................................................................. 37
   4. Класс описания эксперимента ................................................. 39
   5. Класс описания выборки экспериментов ............................... 41
   6. Класс, реализующий формирование отчётов ......................... 45
   7. Класс Класс, содержащий инструменты для отрисовки графических элементов ................................................................................ 45
   8. Класс описания взаимодействий с базой данных .................. 46
   9. Класс, описывающий логику взаимодействия с окном программы.. ...................................... 48
4. Результаты работы ............................................................................ 53
   1. Вкладка «Проведение эксперимента»..................................... 53
   2. Вкладка «Поиск эксперимента» .............................................. 59

Заключение .................................................................................................. 64

Список испольованных источников ......................................................... 65

ПРИЛОЖЕНИЕ A. Код программы .......................................................... 67

ПРИЛОЖЕНИЕ В. Разметка окна ............................................................. 99

ВВЕДЕНИЕ

Сетевая инфраструктура имеет огромное значение в повседневной жизни людей – будь то телефония или локальная сеть в офисе. В настоящий момент популярна идея комбинирования традиционных сетей с локальными областями беспроводного доступа, между которыми пользователи имеют возможность выбирать в соответствии со своими потребностями. Воплощением в жизнь вышеперечисленных тенденций являются «временные сети (ad hoc network) или сети беспроводных приёмопередатчиков, которые состоят из набора мобильных узлов, легко присоединяющихся к сети и покидающих ее с очень незначительными организационными сложностями. Узлы, которые находятся в зоне покрытия других узлов, вызывают друг друга напрямую, тогда как для обмена информацией на большем расстоянии, чем радиус действия, промежуточные узлы играют роль ретрансляторов и сигнал передается через несколько сегментов. За счёт своей особой структуры такая сеть самостоятельно организуется и восстанавливается: при нарушении целостности одного или нескольких маршрутов передачи автоматически производится поиск альтернативных - так сохраняется целостность сети. Ad hoc сети применяются в спасательных работах, чрезвычайных ситуациях, а также промышленной авторизации в качестве коллективной связи и связи общего пользования. Из преимуществ ad hoc сетей можно отметить большой показатель гибкости при монтаже или модернизации и возможность очень сильно сократить расходы на развертывание и последующее техническое обслуживание. Их применение также способствует большей децентрализации функций автоматики на предприятии.

При проектировании таких сетей важно учитывать, какими значениями ключевых характеристик (связность сети, количество связей и количество смежных узлов для устройства) они будут обладать в зависимости от того, сколько устройств должно входить в состав сети, какой уровень сигнала они имеют и в каких масштабах будет развёрнута сеть. Анализу свойств беспроводных самоорганизующихся сетей посвящено некоторое количество публикаций наших соотечественников и зарубежных авторов. [1-6, 8-9]

Однако при систематически частых и массивных исследованиях, направленных на определение значений свойств сети, было бы намного практичнее не проводить расчёты, а брать уже готовые из некого хранилища данных. В связи с актуальностью данного вопроса, целью работы является разработка базы знаний для хранения основных свойств стохастических моделей беспроводных компьютерных ad hoc сетей и, как следствие, разработка пользовательского приложения для её использования.

Для достижения цели в работе решаются следующие задачи:

* рассмотреть параметры области расположения, количество узлов и радиус зоны покрытия отдельного узла в качестве основных свойств ad hoc сети; рассмотреть ключевые характеристики сети: связность сети, количество связей и количество смежных узлов для устройства;



* рассмотреть концепции формирования баз данных;  реализовать алгоритм моделирования беспроводной децентрализованной сети, обладающей заданными основными свойствами;
* реализовать алгоритм расчёта ключевых характеристик сети;
* разработать структуру и реализовать базу знаний для хранения этих характеристик.

Научная новизна состоит в разработке схем базы знаний с учетом того, что записи в ней содержат текстовые значения (в том числе формулы), числовые значения, описания распределений вероятностей, графики (или структурированные данные для построения графиков); в разработке языка запросов к базе знаний, позволяющего отыскивать необходимые записи об экспериментах, а также создавать отчеты, содержащие тексты, числовые значения, и построенные на их основе графики.

Практическое значение работы состоит в обеспечении коллектива пользователей (исследователей, проектировщиков, студентов) локальной специализированной поисковой системой, позволяющей отыскивать результаты предыдущих исследований; в обеспечении системой хранения результатов собственных исследований и проектных решений для использования впоследствии авторами или коллегами.

1. Анализ задачи

1.1 Основные характеристики ad hoc сети

Специфичность беспроводных децентрализованных сетей кроется в том, что их топологические свойства регулярно меняются из-за перемещений в среде узлов системы или динамичности условий, в которых распространяется радиосигнал. В результате, в качестве самых важных характеристик ad hoc сети нужно рассмотреть количество связей между узлами и отдельные её компоненты.[6]

Таким образом, наиболее подходящей математической моделью для такой сети является не что иное, как геометрический граф. [7]

Работоспособность и качественные параметры ad hoc сети могут быть описаны через такие характеристики геометрического графа, как связность, количество рёбер и распределение степеней вершин.

Исследование основных характеристик сети можно производить вероятностными методами. При этом анализируется не одна конфигурация сети, а множество различных конфигураций, иными словами, большое число случайных геометрических графов, составляющих один набор для анализа и расчёта показателей сети. [2]

1.1.1 Вероятность связности сети

Вероятность связности должна показать, с какой уверенностью можно утверждать о том факте, что каждый узел сети с определенными параметрами и построенной в установленных условиях будет иметь доступ к любому другому узлу этой сети. Эта характеристика зависит от ряда факторов: размера области, радиуса зоны покрытия устройств сети, количества самих узлов и их расположения. [1]

Поэтому рассмотрим в работе вероятность связности сети, свойства которой получены формированием достаточно большого набора графов:

,

где *P*c– вероятность связности сети, *K*св – число связных графов в наборе, *K* - общее число графов.

1.1.2 Математическое ожидание степени узла

При проектировании беспроводной децентрализованной сети, помимо работоспособности, следует учитывать ещё и эффективность, а в частности: предполагаемую нагрузку на узел сети. В зависимости от количества узлов, связанных с одним передатчиком, сеть в экстремальных условиях может работать медленнее или, хуже того, потерять связность. Учитывая этот факт, математическое ожидание степени одного узла принято считать одной из важнейших характеристик сети.

Математическое ожидание случайной величины равно сумме произведений всех возможных значений величины на их вероятности.

В качестве оценки для математического ожидания, естественно предложить среднее арифметическое полученных в ходе эксперимента значений:

,

где *M*d– математическое ожидание степени вершины, *Ki* – сумма

степеней d вершин *i*-го графа, *ni* – количество *n* вершин *i*-го графа.

1.1.3 Математическое ожидание количества связей в сети

Знание ожидаемого числа количества связей, несёт собой ту же пользу, что и знание числа связанных с устройством узлов: позволит оптимизировать сеть.

,

где *M*a – оценка математического ожидания количества рёбер, *ai* –

количество рёбер в *i*-м графе, *K* – общее количество графов.

1.1.4 Распределение степеней узлов

Распределение степеней узлов рассматривается, формула 4, с целью узнать, в каком соотношении друг к другу будут находиться всевозможные показатели степеней по количеству обладающих этими степенями вершин.

,

где *qj* – вероятность встретить вершину со степенью *j*, *mj* – количество

вершин со степенью *j*, *Ki* – количество вершин в *i*-м графе.

1.2 Организация базы знаний

При проведении любого эксперимента необходимо накапливать получаемую в ходе опытов информацию, такую, как исходные данные, условия проведения, результаты испытаний и прочие сопутствующие данные. Такие наборы данных должны быть строго структурированы и иметь в сопровождении систему с помощью которой можно было бы всяким образом оперировать накопленными знаниями. Таким образом совокупность данных, а также особые правила, по которым эти данные можно получить, образуют базу знаний результатов исследований. Организация такой базы может быть реализована средствами системы управления базами данных.

База данных представляет собой организованную совокупность самостоятельных данных. Это коллекция схем, таблиц, запросов, отчетов, представлений и других объектов, которые систематизированы так, чтобы их обработка и поиск быстро выполнялись с помощью электронной вычислительной машины.

Реляционная база данных представляет собой набор элементов данных, организованных как набор формально описанных таблиц, из которых данные могут быть доступны или вновь собираются по-разному без реорганизации таблиц базы данных. Реляционная база данных была изобретена Е. Ф. Кодда в IBM в 1970 году. [10]

Таблица - это набор элементов данных, организованных в терминах строк и столбцов. Таблица также рассматривается как удобное представление отношений и является наиболее простой формой хранения данных.

Запись в таблице является совокупностью взаимосвязанных данных. Таблица состоит из нескольких записей (строк), каждая запись может быть разбита на несколько частей – полей; рисунок 1.

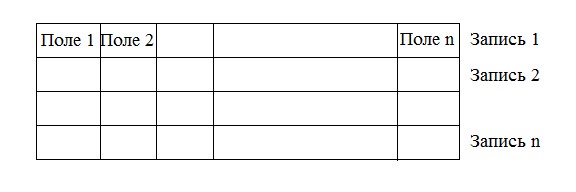


Рисунок 1 – Поля и записи в таблице базы данных

Система управления базами данных (СУБД) представляет собой совокупность программных средств, которая взаимодействует с пользователем, другими приложениями, а также базой данных сама по себе для сбора и анализа данных. СУБД общего назначения разработана, чтобы позволить определение, создание, обработку запросов, обновление и администрирование баз данных. Хорошо известные СУБД включают MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, SAP HANA и IBM

DB2. База данных, как правило, не переносима между разными СУБД, но различные СУБД могут взаимодействовать с помощью стандартов.

Простейшую схему работу с базой можно увидеть на рисунке 2.

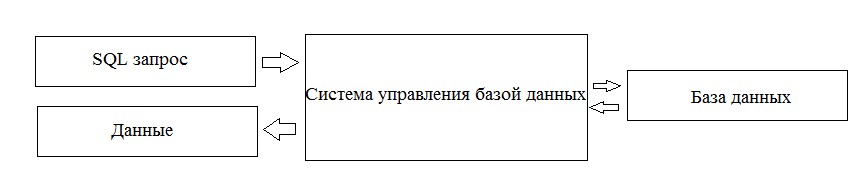


Рисунок 2 – Простейшая схема работы с БД

1.2.1 СУБД Oracle

Oracle Database (обычно упоминается как СУБД Oracle или просто как Oracle) является объектно-реляционной системой управления базами данных, производится и продается компанией Oracle Corporation.

СУБД Oracle может хранить и выполнять процедуры и функции внутри себя, такие процедуры называются хранимыми. Для Oracle разработан специальный язык PL / SQL - запатентованное процедурное расширение корпорации Oracle к SQL.

СУБД Oracle хранит данные логически в виде табличных пространств и физически в виде файлов данных Tablespaces, которые могут содержать различные типы сегментов памяти, таких, как сегменты данных, индексные сегменты. Сегменты в свою очередь, могут включать в себя один или более экстентов. Экстенты содержат группы смежных блоков данных. Блоки данных формируют основные единицы хранения данных.

Администратор баз данных может наложить максимальные квоты хранения для одного пользователя в пределах каждого табличного пространства.

Среди отличительных черт Oracle можно выделить функцию секционирования, которая была введена в Oracle 8. Функция секционирования позволяет разбивать таблицы на основании некоторого набора ключей. Конкретные разделы могут быть легко добавлены или выброшены, чтобы оптимизировать управление большими наборами данных.

Oracle отслеживает хранение компьютерных данных с помощью информации, хранящейся в табличном пространстве SYSTEM. Оно содержит словари данных, которые состоят из специального набора таблиц, содержащих в себе информацию обо всех пользовательских объектах в базе данных.

Каждая база данных Oracle имеет один или несколько физических файлов данных, которые содержат все данные базы и называются Datafiles.

Datafiles имеют следующие характеристики:

* один или несколько файлов данных формируют логическую единицу хранения базы данных (таблицу);
* файл данных может быть связан только с одной таблицей;  данные в файле считываются, при необходимости, во время нормальной работы базы данных и сохраняются в кэш-памяти Oracle Database. Например, если пользователь хочет получить доступ к некоторым данным в таблице базы данных, и, если запрашиваемая информация уже не в кэш-памяти для базы данных, то он считывается из соответствующих файлов данных и хранятся в памяти;
* измененные или новые данные не обязательно записываются в файл данных немедленно. Для того, чтобы уменьшить количество доступа к диску и для увеличения производительности, данные объединены в памяти и записываются в соответствующие файлы данных все сразу;
* буфер журнала повторов сохраняет повтор записи в журнал изменений, внесенных в базу данных. Экземпляр записывает повтор буферов журнала в журнал повторного выполнения так быстро и эффективно, насколько это возможно. Журналы помогают в восстановлении экземпляра в случае сбоя системы.

1.2.2 СУБД Microsoft SQL Server

Microsoft SQL Server является реляционной системой управления базами данных, разработанной Microsoft [10].

База данных MS SQL представляет собой набор таблиц с

типизированными столбцами. SQL Server поддерживает различные типы данных, включая первичные типы, такие как Integer, Float, Decimal, Char (в том числе символьные строки), Varchar (строки переменной длины символов), двоичный (для неструктурированных сгустков данных), текст (для текстовых данных).

Microsoft SQL Server также позволяет использовать составные типы, определяемые пользователем (UDT). В дополнение к таблицам, база данных может также содержать другие объекты, в том числе представления, хранимые процедуры, индексы и ограничения, а также журнал транзакций. База данных SQL Server может содержать максимум 231 объектов, и может охватывать несколько файлов на уровне операционной системы с максимальным размером файла 260 байт. Данные в базе данных хранятся в первичных файлах данных с расширением mdf.

Основной способ извлечения данных из базы данных SQL Server – это запрос. Запрос выражается с помощью варианта SQL под названием T-SQL (диалект Microsoft SQL Server с Sybase SQL Server). Запрос декларативно указывает, что должно быть получено. Он обрабатывается процессором запроса, который выясняет последовательность шагов, необходимых для получения требуемых данных. Последовательность действий, необходимых для выполнения запроса, называется планом запроса. В MS SQL может быть

несколько способов обработки одного и того же запроса.

SQL Server также позволяет использовать хранимые процедуры, которые могут быть определены пользователем. Хранимые процедуры параметризованных запросов T-SQL хранятся на самом сервере (и не выдаются клиентским приложением, как в случае с общими запросами). Хранимые процедуры могут принимать значения, отправленные клиентом в качестве входных параметров, и отправить обратно результаты в качестве выходных параметров. Они могут содержать определенные функции, а также другие хранимые процедуры, в том числе ту же хранимую процедуру (до установленного количества раз). В отличие от других запросов, хранимые процедуры имеют соответствующее имя, которое используется во время выполнения для включения в фактические запросы.

T-SQL (Transact-SQL) является вторичным средством программирования и управляющей SQL Server. Он предоставляет ключевые слова для операций, которые могут выполняться на SQL Server, включая создание и изменение схемы базы данных, ввод и редактирование данных в базе данных, а также мониторинга и управления самого сервера. Клиентские приложения, которые потребляют данные или управляют сервером будут использовать функциональность SQL Server путем отправки запросов и операторов, которые затем обрабатываются сервером, и результаты (или ошибки) возвращается в клиентское приложение.

1.2.3 Сравнение Oracle и MS SQL Server

Системы управления базами данных часто классифицируются в соответствии с моделью базы данных, которую они поддерживают. Самые популярные системы управления базами данных поддержали реляционную модель.

Среди доступнейших СУБД для Windows самыми популярными являются Oracle и MS SQL Server.

Пожалуй, наиболее очевидная разница между двумя РСУБД заключается в языке, который они используют. Хотя обе системы используют версию Structured Query Language или SQL, MS SQL Server использует Transact SQL, или T-SQL, который является расширением SQL первоначально разработанного Sybase и используется корпорацией Microsoft. Oracle, тем временем, использует PL / SQL или процедурный язык / SQL. Оба являются различными диалектами SQL и оба языка имеют разный синтаксис и возможности. Основное различие между этими двумя языками состоит в том, как они обрабатывают переменные, хранимые процедуры, а также встроенные функции. PL / SQL в Oracle группирует процедуры в пакеты. MS SQL Server, в свою очередь, пакеты не использует. Считается, что, PL / SQL является сложным и потенциально более мощным, в то время как T-SQL гораздо проще и удобнее в использовании.

Еще одним из самых больших различий между Oracle и MS SQL Server является контроль транзакций. Транзакция может быть определена как группа операций или задач, которые следует рассматривать как единое целое. Например, набор SQL запросов по редактированию записей, которые должны быть обновлены в то же самое время, когда неспособность обновлять какиелибо отдельные должно привести к тому, что ни одна из записей не обновляется. По умолчанию, MS SQL Server будет выполнять каждую команду в индивидуальном порядке, и впоследствии будет трудно или невозможно выполнить откат изменений, если по пути возникнут какие-либо ошибки. Для правильной групповой операции используется либо команда "BEGIN TRANSACTION" в самом начале запроса, либо COMMIT, который используется в конце. В рамках одной транзакции, ROLLBACK отменит любые изменения, сделанные в пределах блока транзакции. При правильном использовании с обработкой ошибок, откат допускает определенную степень защиты от повреждения данных.

В рамках Oracle, с другой стороны, каждое новое соединение с базой данных обрабатывается как новая транзакции. Сразу после выполнения запросов и команд, изменения происходят только в памяти и изменения никак не закрепляются, пока не будет использован COMMIT (с некоторыми исключениями, связанными с DDL командами, которые включают в себя "неявное" действие, и совершаются немедленно). После того, как выполнится COMMIT, следующая команда, по существу, инициирует новую транзакцию, и процесс начинается снова. Это обеспечивает большую гибкость и позволяет осуществлять контроль ошибок.

Последнее различие состоит в том, как СУБД организует объекты базы данных. MS SQL Server организует все объекты, такие как таблицы, представления и процедуры, по именам баз данных. Пользователи назначаются на логин, который предоставляется доступ к определенной базе данных и ее объектам. Кроме того, в SQL Server каждая база данных имеет частный, неразделенную дисковый файл на сервере. В Oracle все объекты базы данных сгруппированы по схемам, которые представляют собой набор подмножеств объектов базы данных. Все объекты базы данных являются общими среди всех схем и пользователей. Несмотря на то, что это все совместно, каждый пользователь может быть ограничен определенными схемами и таблицами с помощью ролей и разрешений.

1.2.4 Обоснование выбора СУБД MS SQL

В свете результатов проведенного анализа рассмотренных СУБД можно сказать, что для данного проекта наиболее подходящим вариантом будет использование MS SQL Server. Такой выбор обусловлен в большей степени используемым способом организации базы данных на жестком диске компьютера: во-первых, полностью отсутствует необходимость в предоставлении доступа для других проектов, а во-вторых, было бы удобно хранить базу одним файлом для быстрой транспортировки. Проста использования языка T-SQL в рамках данной работы будет большим плюсом в отличие от возможности объединения хранимых процедур в пакеты, так как планируется автоматизировать работу базы средствами языка C#.

1.3 Хранение и формирование формул и отчётов

Помимо результирующих характеристик проведения экспериментов база данных должна содержать формулы и отчёты по проведённым операциям. Если характеристики хранятся в очевидном числовом формате, то с хранением и отображением формул и отчётов возникает ряд проблем. Во-первых, формат хранения не очевиден, так как формулы имеют строгое форматирование, а отчёты состоят не только из текста, но и содержат вышеупомянутые сложноорганизованные формулы и графики.

1.3.1 Хранение в базе знаний

Сперва стоит с самого начала отказаться от идеи хранения в базе файлов в силу того, что такой формат данных даже при небольшом их количестве очень сильно нагружает базу, увеличивая её физический объём и скорость обработки. В силу этих соображений, можно прийти к выводу о том, что в базе будут храниться данные только строкового типа для отображения на экране или в документах.

Есть несколько возможных способов решения этой задачи:

1. Хранить в базе заранее подготовленные изображения формул, а затем непосредственно выводить их на экран. Этот способ самый простой в реализации, но самый нерациональный в плане гибкости и использования ресурсов. Во-первых, изображения – это очень объемный объект, который будет очень сильно нагружать базу данных, а во-вторых, в изображения нельзя подставить собственные значения переменных.
2. Загружать картинки с online сервисов, которые принимают формулы в формате LaTeX и преобразуют их в изображения нужного расширения. Данный способ хорош тем, что можно генерировать текст формул непосредственно в коде и хранить в базе только адрес к внешнему ресурсу. Но, прибегнув к такому методу, мы устанавливаем постоянную зависимость приложения от сети и работоспособности online сервиса, а также обрекаем себя на достаточно сложную реализацию.
3. Формировать формулы в формате MathML и отображать их в специализированном элементе управления, размещенном на форме. Изучение данного способа привело к открытию, суть которого заключается в том, что IE не имеет встроенной поддержки MathML, и обычно такие проблемы решаются установкой плагина MathPlayer. Изначально способ очень привлекательный, но распространять плагин для IE вместе с приложением не представляется возможным.
4. Создавать формулу с помощью компоновщиков html, хранить её в текстовом формате, а потом парсить специальным образом. Этот вариант выглядит привлекательнее всего, потому что шаблоны формул можно будет хранить в html (varchar) и парсить html во что угодно – для этого существует множество инструментов. Кроме того, в html можно будет хранить не только шаблоны формул, но и отформатированные шаблоны отчётов.

1.3.2 Отображение

В предыдущем подразделе было выяснено, что хранение формул и отчётов удобнее всего осуществлять в формате html – именно этим способом можно «убить двух зайцев одним выстрелом» и решить как проблему хранения и отображения формул, так и хранения отчётов.

Незакрытым остаётся лишь вопрос обработки хранимых в html данных для отображения. Самым удобным и популярным форматом отчётности в данный момент является PDF, поэтому будем рассматривать вопрос как задачу поиска способа конвертации из html в PDF.

Самой популярной библиотекой для решения такой задачи на С# является внешняя библиотека iTextSharp, которая позволяет парсить html и генерировать PDF документы непосредственно в коде проекта.

1. Проектирование

Одним из главных предметов исследования в данной работе является база данных, однако перед тем, как начать проектировать структуру таблиц, следует сформировать чёткие требования к программе, реализующей набор правил для ввода и вывода информации из базы данных.

2.1 Требования к программе

Разрабатываемое приложение должно позволять:

* 1. Проводить эксперименты по введённым параметрам.

Под экспериментом будем понимать генерацию такого достаточно большого количества графов, при котором накопленные статистическим путём результаты будут удовлетворять требованиям точности.

В качестве вводимых параметров для построения новой модели сети выбраны:

* + - форма области, в которой построена граф (овальная или

квадратная);

* + - высота области;  ширина области;  количество вершин в графе;  радиус действия вершины.
  1. Заносить результаты экспериментов в специализированную базу данных.

В качестве результатов проведения одного эксперимента выступают:

* + - вероятность связности;  математическое ожидание степени вершины;  математическое ожидание количества рёбер; распределение степеней вершин.



* 1. Проводить поиск в специализированной базе данных по введённым параметрам.

Чтобы упростить поиск, планируется реализовать разделение параметров поиска на три группы.

Параметры поиска по форме области:  форма;  высота: минимальное значение;  высота: максимальное значение;  ширина: минимальное значение;  ширина: максимальное значение.

Параметры поиска по свойствам эксперимента:  количество вершин: минимальное значение;  количество вершин: максимальное значение;  радиус вершин: минимальное значение;  радиус вершин: максимальное значение;  количество графов в эксперименте: минимальное значение.

* + - количество графов в эксперименте: максимальное значение.

Параметры поиска по результатам эксперимента:  вероятность связности: минимальное значение;  вероятность связности: максимальное значение;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | математическое значение; | ожидание | степени вершины: | минимальное |
|  | математическое значение; | ожидание | степени вершины: | максимальное |
|  | математическое значение; | ожидание | количества рёбер: | минимальное |
|  | математическое | ожидание | количества рёбер: | максимальное |

значение.

* 1. Выводить результаты поиска на экран в структурированном виде.

Под результатами поиска подразумеваются вывод входных данных, на которых был построен эксперимент (указаны в пункте 1) и результатов его проведения (указаны в пункте 2).

* 1. Формировать типизированные отчёты.

Отчёты должны содержать информацию, позволяющую проводить анализ найденных в базе данных или только что проведённых экспериментах.

Отчёт об эксперименте обязан содержать те данные, на которых он построен, результаты проведения, а также графики динамики формирования этих результатов и линию распределения степеней вершин.

Отчёт о найденных в базе экспериментах будет представлять собой множество экспериментов с подходящими по параметрам поиска свойствами. В нём должны быть перечислены данные по каждому участвующему в такой выборке эксперименту, а также средние значения по этой выборке.

Данные для формирования отчётов и построения графиков, которые будут в них содержаться должны храниться в базе данных.

1. Сохранять отчёты на жестком диске.

Сохранять отчёты необходимо для удобства пользователей. Отчёты должны храниться на диске в формате PDF.

1. Проводить все вышеперечисленные манипуляции через пользовательский интерфейс.

Все операции должны быть доступны пользователю через графический интерфейс.

2.2 Проектирование таблиц базы данных.

Планы структур, составляющих базу данных, представлены в таблицах

1-5.

2.2.1 Таблица EXPERIMENTS

Из предыдущего подраздела можно сделать вывод о том, что основным элементом проекта будет являться объект эксперимент, поэтому первым делом рассмотрим таблицу, в которой будут храниться данные об этой единице проекта.

В ней должны содержаться все параметры, по которым проводится эксперимент, а также результаты его проведения. Так как все эти параметры будут являться числовыми и не будут осложнять поиск и сортировку, то представляется удобным хранить все эти данные в одной таблице.

Таблица 1 – Список полей EXPERIMENTS

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Предназначение |
| ID | int | Уникальный идентификатор  записи в таблице |
| AREA\_SHAPE | int | Форма области, 0 – квадратная, 1- овальная. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| AREA\_WIDTH | float | Ширина области может быть представлена числом с плавающей точкой и характеризует ширину прямоугольной области или длину большой оси  эллиптической области |
| AREA\_HEIGHT | float | Высота области может быть представлена числом с плавающей точкой и характеризует высоту прямоугольной области или длину малой оси эллиптической области |
| NODES\_COUNT | int | Количество вершин в одном  графе |
| RADIUS | float | Радиус вершины, который так же, как и размеры области может быть представлен числом с плавающей точкой |
| GRAPHS\_COUNT | int | Число графов в одном  эксперименте |
| CONNECTED\_GRAPHS\_COUNT | int | Число связных графов – является в своём роде переменной для возможности отобразить данную величину в формулах как промежуточное |
|  |  | значение |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| DEGREES\_SUMM | int | Сумма степеней всех вершин, хранится для отображения в формулах |
| ARCS\_COUNT | int | Число рёбер во всех сформированных графах, хранится для отображения в формулах |
| CONNECTION\_POSSIBILITY | float | Вероятность связности, результат проведения  эксперимента |
| DEGREE\_EXPECTATION | float | Математическое ожидание степени вершины, результат проведения эксперимента |
| ARCS\_EXPECTATION | float | Математическое ожидание количества рёбер, результат проведения эксперимента |

В таблице EXPERIMENTS отражены все свойства и результирующие характеристики эксперимента, кроме динамики изменения результирующих характеристик в зависимости от количества сформированных графов и распределения степеней вершин, так как они представляют собой зависимости и требуют отдельных таблиц.

2.2.2 Таблица POSSIBILITY\_DISTRIBUTION

В таблице будут храниться сведения о распределении степеней вершин для всех экспериментов.

Таблица 2 – Список полей POSSIBILITY\_DISTRIBUTION

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Предназначение |
| ID | int | Уникальный идентификатор записи о распределении |
| EXPERIMENT\_ID | int | Идентификатор, по которому можно определить, к какому именно эксперименту относится пара «степень – количество вершин с такой степенью» |
| DEGREE | int | Число, отражающее степень |
| NODES\_COUNT | int | Число вершин, обладающей степенью DEGREE |

2.2.3 Таблица DYNAMIC

Таблица предназначена исключительно для построения графиков, отражающих динамику результирующих величин на протяжении проведения эксперимента, которая позволяет определить, на каком числе сформированных статистических величин показатели вероятности связности, математического ожидания степени вершины и числа рёбер становятся стабильными, а, следовательно, пригодными к рассмотрению и обеспечивающими приемлемую точность.

Таблица 3 – Список полей DYNAMIC

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Предназначение |
| ID | int | Уникальный идентификатор записи о |
| EXPERIMENT\_ID | int | Идентификатор, по которому можно определить, к какому именно эксперименту относится пара «степень – количество вершин с такой степенью» |
| GRAPHS\_COUNT | int | Текущее количество графов |
| CONNECTION\_POSSIBILITY | float | Значение вероятности связности для  GRAPHS\_COUNT |
| DEGREE\_EXPECTATION | float | Значение математического ожидания степени вершины для  GRAPHS\_COUNT |
| ARCS\_EXPECTATION | float | Значение математического ожидания количества рёбер для  GRAPHS\_COUNT |

Как и в случае с первой таблицей, нет возможности отразить динамику изменения распределения степеней вершин без использования отдельной таблицы и её дополнительной обработки. Если вспомнить, что динамика в рамках проекта существует только для отображения на графиках, то можно говорить об отсутствии необходимости реализовывать лишнюю таблицу для вышеперечисленных целей, так как отразить изменение распределения степеней невозможно на графике с двумя осями координат, а трёхмерный график не будет достаточно наглядным.

2.2.4 Таблица TEMPLATES

В разделе с требованиями к программе говорилось о необходимости включить отчёты или данные для их построения в базу данных для лучших организации и гибкости программы.

Хранить отчёты вместе с отраженными в них формулами и отображением графиков в формате PDF будет, во-первых, нерационально, во-вторых, не необходимо. Предполагается, что программа предоставляет функционал для поиска и формирования экспериментов со строго определенными параметрами, и пользователь не нуждается в таком количестве уже готовых отчётов, что найти их было бы быстрее и проще, чем сформировать заново. В связи с этим в базе будут храниться только данные для построения отчётности.

В таблицах EXPERIMENTS и DYNAMIC уже содержится материал для формирования формул и графиков, следовательно, остается лишь сформировать шаблон, по которому будет строиться тот или иной тип отчётов.

В рамках данного проекта существует всего два типа отчётов: для экспериментов и для выборки экспериментов. В будущем этот набор может расшириться, что стоит учесть при решении задачи хранения данных для формирования отчётов.

Одним из очевиднейших решений – создание таблицы TEMPLATES, где будет содержаться тела отчётов и формул. Почему шаблоны для формул будут включены в одну таблицу с шаблонами отчётов? Потому что каждая из формул в данный момент и, тем более, на случай увеличения функционала программы будет содержаться не только в одном из отчётов, что делает решение о включении самой формулы в шаблон для отчёта нерациональным. Кроме того, шаблоны для отчётов и формул будут иметь одинаковый строковый формат, что позволяет не нагружать базу и код дополнительными объектами, совершенно не упрощающими последующую обработку.

С другой стороны, поля строкового типа как раз-таки нагружают базу и в большой степени замедляют поиск, сортировку и другие простейшие манипуляции с таблицами. На данный момент в таблице TEMPLATES не планируется содержать столько записей, чтобы их тип мог как-либо замедлить работу с приложением, однако строковые значения в ней будут непозволительно велики и неделимы. Учитывая, что набор таких значений может быть увеличен при дальнейшем расширении функционала, стоит вынести тела самих шаблонов в отдельную таблицу со строками, в которой, помимо них будут храниться ещё и некоторые заголовки физические пути к файлам.

Кроме всего этого, стоит учесть, что шаблон будет включать в себя подшаблоны. Например, шаблоны формул или некоторых таблиц, которые хранятся отдельно, нужно будет включить в основное тело отчёта. Можно было бы реализовать этот процесс во внешнем коде, но было бы объективнее закончить «сборку» шаблона на этапе обращения к классу, обрабатывающему базу, поэтому в запись о самом шаблоне было решено включить ещё и список его подшаблонов. Он будет храниться в виде строки, однако каждая такая строка будет слишком мала, чтобы как-либо нагрузить таблицу.

Чтобы знать, в какое место шаблона нужно вставить чётко определенный дочерний, будут использоваться уникальные текстовые метки.

Таблица 4 – Список полей TEMPLATES

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип | Предназначение | |
| ID | int | Уникальный идентификатор для записи шаблона | |
| BODY\_STRING\_ID | int | Идентификатор записи строки, которой содержится тело шаблона | в |

Продолжение таблицы 4

|  |  |  |
| --- | --- | --- |
| CHILDREN | varchar(50) | Список идентификатор дочерних шаблонов, хранится в базе через запятую |
| MARK\_STRING\_ID | int | Поле имеет значение не NULL только в том случае, если запись относится к дочернему шаблону. Это ссылка на строку, в которой указано, вместо какой метки нужно вставить дочерний шаблон в тело основного. |

2.2.5 Таблица STRINGS

Данная таблица будет использована для хранения больших строковых данных, вроде тел шаблонов, а также путей к файлам, имён файлов и текстовых меток.

Таблица 5 – Список полей таблицы STRINGS

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Предназначение |
| ID | int | Уникальный идентификатор для записи строки |
| NAME | varchar(50) | Имя строкового значения для удобства ориентирования в таблице |
| TYPE | varchar(50) | Тип строкового значения для удобства ориентирования в таблице с множеством типов записей: шаблоны, формулы, имена, метки |

Продолжение таблицы 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| VALUE | varchar(MAX) | Непосредственно переменной | само | значение |

На рисунке 3 представлена общая схема базы данных в проекте.

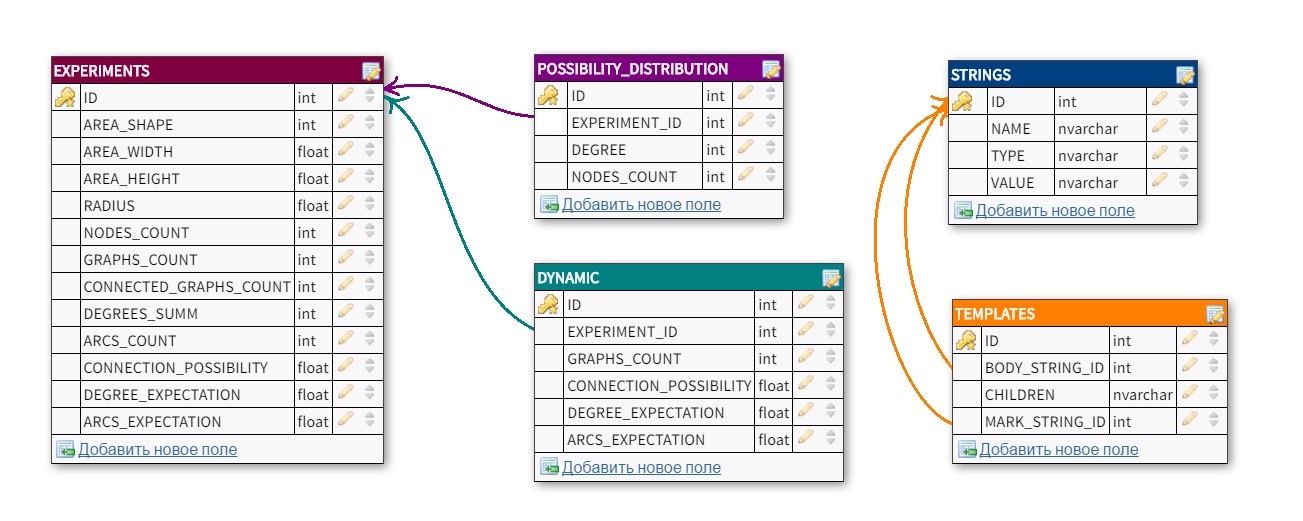


Рисунок 3 – Схема базы данных проекта

1. Часть реализации

Программа реализована на языке С# и была разработана с помощью средств Visual Studio 2015. Диаграмма классов проекта изображена на рисунке 4.

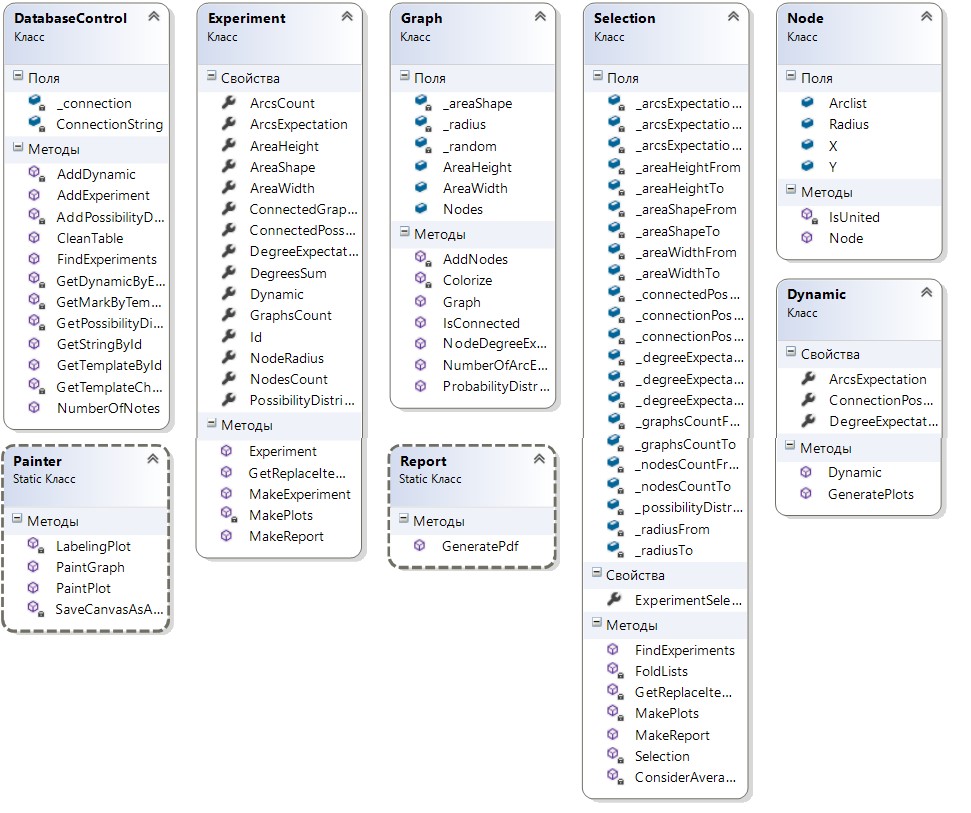


Рисунок 4 – Диаграмма классов проекта

* 1. Реализация базы данных средствами Microsoft SQL Server

Visual Studio 2015 позволяет добавить элемент базы данных как составляющую проекта, рисунок 4.

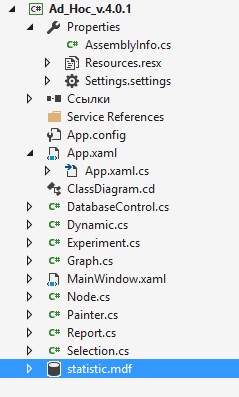


Рисунок 4 – Файл базы данных входит в состав проекта

* 1. Класс описания узлов сети

Класс был создан в качестве структуры для хранения и генерации вершин случайного графа.

Поля:

* + - * + float Radius – содержит радиус вершины;  float X – содержит абсциссу вершины;  float Y – содержит ординату вершины;
        + List<Node> Arclist - представляет собой список вершин, смежных данной.

Методы:

* + - * + Node (int areaShape, float areaHeight, float areaWidth, float radius, ref

List<Node> nodes, Random random) – конструктор класса, принимающий на вход форму и размеры области, радиус вершины, список вершин графа, которому будет принадлежать вершина и объект класса Random для генерации случайных чисел. Этот конструктор формирует свойства вершины и связывает её с вершинами из списка nodes в зависимости от того, должны ли они быть связаны; bool IsUnited (Node node1, Node node2) – возвращает истину, если вершины должны быть связаны.



* 1. Класс описания сети

Класс представляет собой структуру для хранения случайного графа как составляющей одного эксперимента, а также содержит методы, позволяющие собирать статистику.

Поля: int \_areaShape – форма области, необходима для генерации вершин; float AreaWidth – ширина области, используется для генерации



вершин и для отрисовки графа классом Painer;

float AreaHeight - высота области, используется для генерации вершин и для отрисовки графа классом Painer;

* + - * + float \_radius – радиус вершин, передаётся в конструктор вершины;
        + List<Node> Nodes – список вершин, принадлежащих графу, обозначен как свойство, потому что используется при отрисовке; Random \_random – объект класса Random для корректной генерации случайных чисел.



Методы: bool IsConnected () – определяет, является ли граф связным, используя алгоритм раскраски графа;



* + - * + void Colorize (List<bool> list, int i) – функция раскраски графа;  void AddNodes (int count) – добавляет в граф количество вершин N

= count;

* + - * + void NodeDegreeExpectation (ref int degreeSum, ref int numberOfNodes) – «пропускает» через себя сумму степеней и количество вершин в рамках эксперимента, прибавляя к ним сумму степеней и количество вершин для текущего графа; void NumberOfArcExpectation (ref double numberOfArcs) -



«пропускает» через себя количество рёбер в рамках эксперимента, прибавляя к ним количество рёбер текущего графа;

* + - * + void ProbabilityDistribution (ref List<int> degrees) - «пропускает» через себя список распределения степеней вершин в рамках эксперимента и включает туда распределение степеней вершин для текущего графа.

* 1. Класс описания эксперимента

Этот класс является своего рода контейнером для записей из таблицы EXPERIMENTS. Этот класс не только содержит все ключевые свойства и результаты проведения эксперимента, но и осуществляет генерацию новых экспериментов.

Поля:

* + - * int Id – уникальный идентификатор записи, служит для поиска эксперимента в базе данных; int AreaShape – форма области, используется для добавления в базу данных; float AreaHeight – высота области, используется для добавления в базу данных; float AreaWidth – ширина области, используется для добавления в базу данных; float NodeRadius – радиус вершины, используется для добавления в базу данных; int GraphsCount – количество графов, используется для добавления в базу данных; int NodesCount – количество вершин, используется для добавления в базу данных; int ArcsCount – количество рёбер, используется для добавления в базу данных; int ConnectedGraphsCount – количество связных графов, используется для добавления в базу данных; int DegreesSum – сумма степеней вершин, используется для добавления в базу данных; float ConnectedPossibility – вероятность связности, результат проведения эксперимента, заносится в базу данных;



float DegreeExpectation – математическое ожидание степени вершины, результат проведения эксперимента, заносится в базу данных;

* + - * float ArcsExpectation – математическое ожидание числа рёбер, результат проведения эксперимента, заносится в базу данных; Dynamic Dynamic – объект класса Dynamic, который содержит динамику изменения результирующих величин, используется для построения графиков и занесения данных в базу знаний; List<int> PossibilityDistribution – список распределения степеней вершин, степени – это порядковый номер, а количество вершин с данной степенью – это непосредственно сам элемент списка.



Методы:

* + - * Experiment (int id, int areaShape, float areaHeight, float areaWidth, float nodeRadius, int graphsCount, int nodesCount, int arcsCount, int connectedGraphsCount, int degreeSum, List<int> possibilityDistribution, Dynamic dynamic) – конструктор, принимающий на вход все параметры области, параметры эксперимента и динамику. Внутри происходит заполнение полей объекта класса и подсчёт результирующих характеристик: вероятности связности, математического ожидания степени вершины и математического ожидания количества рёбер;
      * Experiment MakeExperiment (int areaShape, float areaHeight, float areaWidth, float radius, int nodesCount, int graphsCount, Random random) – проводит эксперимент на указанных входных

параметрах области и заносит созданный объект в базу данных; Dictionary<string, string> GetReplaceItemsForReport () – данный метод предназначен для работы с отчётами. Каждый шаблон отчёта содержит строковые метки, вместо которых должны отображаться данные, соответствующие текущему эксперименту, GetReplaceItemsForReport возвращает словарь, ключом в котором является заменяемая метка, а значением – значение переменной, которое нужно вставить в текст шаблона; void MakePlots () – использует класс Painter, чтобы отрисовать графики динамики и распределения степеней вершин; void MakeReport () – формирует отчёт для текущего эксперимента.



* 1. Класс описания выборки экспериментов

Класс Selection предназначен для хранения и оперирования выборками экспериментов, найденных по определенным параметрам.

Поля:

* + - int \_areaShapeFrom - минимальное значение формы области, в нашем случае – это 0;
    - int \_areaShapeTo – максимальное значение параметра формы области, в нашем случае – это 1;
    - float \_areaHeightFrom – минимальное значение высоты области, используется для поиска в базе данных; float \_areaHeightTo – максимальное значение высоты области, используется для поиска в базе данных; float \_areaWidthFrom – минимальное значение ширины области, используется для поиска в базе данных;



* + - float \_areaWidthTo – максимальное значение ширины области, используется для поиска в базе данных;
    - float \_radiusFrom – минимальное значение радиуса вершины, используется для поиска в базе данных; float \_radiusTo – максимальное значение радиуса вершины, используется для поиска в базе данных;

int \_nodesCountFrom – минимальное значение количества вершин, используется для поиска в базе данных; int \_nodesCountTo – максимальное значение количества вершин, используется для поиска в базе данных; int \_graphsCountFrom – минимальное значение количества графов, используется для поиска в базе данных; int \_graphsCountTo – максимальное значение количества графов, используется для поиска в базе данных; float \_connectionPossibilityFrom – минимальное значение вероятности связности, используется для поиска в базе данных; float \_connectionPossibilityTo – максимальное значение вероятности связности, используется для поиска в базе данных; float \_degreeExpectationFrom – минимальное значение математического ожидания степени вершины, используется для поиска в базе данных;



* + - float \_degreeExpectationTo – максимальное значение математического ожидания степени вершины, используется для поиска в базе данных;
    - float \_arcsExpectationFrom – минимальное значение математического ожидания количества рёбер, используется для поиска в базе данных;
    - float \_arcsExpectationTo – максимальное значение математического ожидания количества рёбер, используется для поиска в базе данных;
    - List<Experiment> ExperimentSelection – список экспериментов, участвующих в выборке; float \_connectedPossibilityAverage – содержит среднее значение вероятности связности для экспериментов, входящих в выборку;



* + - float \_degreeExpectationAverage – содержит среднее значение математического ожидания степени вершины для экспериментов, входящих в выборку;
    - float \_arcsExpectationAverage – содержит среднее значение математического ожидания количества рёбер для экспериментов, входящих в выборку;
    - List<float> \_possibilityDistributionAverage – содержит среднее значение распределения степеней вершин для экспериментов, входящих в выборку; Методы:
    - Selection (int areaShapeFrom, int areaShapeTo, float areaHeightFrom, float areaHeightTo, float areaWidthFrom, float areaWidthTo, float radiusFrom, float radiusTo, int nodesCountFrom, int nodesCountTo, int graphsCountFrom, int graphsCountTo, float connectionPossibilityFrom, float connectionPossibilityTo, float degreeExpectationFrom, float degreeExpectationTo, float arcsExpectationFrom, float arcsExpectationTo, List<Experiment> experimentSelection) – конструктор, заполняет значения полей и считает средние значения результирующих величин для выборки экспериментов experimentSelection;
    - Selection FindExperiments (int areaShape, float areaHeightFrom, float areaHeightTo, float areaWidthFrom, float areaWidthTo, float radiusFrom, float radiusTo, int nodesCountFrom, int nodesCountTo, int graphsCountFrom, int graphsCountTo, float connectionPossibilityFrom, float connectionPossibilityTo, float degreeExpectationFrom, float degreeExpectationTo, float arcsExpectationFrom, float arcsExpectationTo, string operationCode) – статический метод, который используется для поиска выборки экспериментов, которая удовлетворяет указанным в параметрах условиям. Помимо условий, в метод передаётся строковое значение operationCode. OperationCode представляет собой последовательность из трёх чисел от 0 до 1: каждое из этих чисел означает, включен ли определенный набор параметров поиска (параметры области, параметры проведения эксперимента, параметры результатов эксперимента) в условия. Так, например, при значении operationCode = “001” при поиске будут учитываться только параметры результатов проведения экспериментов;
    - void СonsiderAverages () – вычисляет средние значения вероятности связности, математического ожидания степени вершины, математического ожидания количества рёбер и распределения степени вершин для выборки экспериментов;
    - List<int> FoldLists (List<int> list1, List<int> list2) – возвращает список целочисленных значений, каждый элемент которого был получен сложением соответствующих друг другу элементов из целочисленных списков list1 и list2. Метод используется для подсчёта распределения средних значений степеней вершин для выборки экспериментов;
    - Dictionary<string, string> GetReplaceItemForReport () - данный метод предназначен для работы с отчётами. Каждый шаблон отчёта содержит строковые метки, вместо которых должны отображаться данные, соответствующие текущему эксперименту, GetReplaceItemsForReport возвращает словарь, ключом в котором является заменяемая метка, а значением – значение переменной, которое нужно вставить в текст шаблона; void MakePlots () – отрисовывает графики, которые будут помещены в отчёт; void MakeReport () – генерирует отчёт для выборки экспериментов.



* 1. Класс, реализующий формирование отчётов

Класс предназначен для хранения методов, связанных с конвертацией отчётов. Изначально класс планировался как универсальный сборник статических методов, позволяющих строить отчёты, но впоследствии такое решение не было сочтено за рациональное, поскольку методы формирования отчётов будет практичнее привязывать к объектам, для которых они формируются.

Тем не менее, остался один универсальный для построения любого отчёта, поэтому было решено вынести его в отдельный класс с учётом того, что в будущем этот класс может расшириться.

Методы:

void GeneratePdf (string template, string fileName) – принимает шаблон отчёта и имя файла, в который его нужно сохранить, а потом парсит html в PDF, используя методы библиотеки iTextSharp.



* 1. Класс, содержащий инструменты для отрисовки графических

элементов

Painter создавался с целью собрать в себе методы, ответственные за отрисовку графических составляющих проекта: рисунков и графиков. Класс обозначен, как статический, следовательно, методы можно вызывать, не инициализируя при этом экземпляр.

Методы:

void PaintGraph (Graph graph, Panel canvas) – получает на вход объект Graph и рисует его на панели canvas. Этот метод не является необходимым, но был создан для придания наглядности некоторому функционалу программы, в частности, проведению нового эксперимента. После проведения очередного эксперимента рисуется граф, соответствующий параметрам, введённым пользователем: так студент или преподаватель может увидеть изображение смоделированной им сети; void PaintPlot (IReadOnlyList<float> additional, string fileName) – метод принимает список значений и имя файла. PaintPlot отрисовывает зависимость additional и сохраняет графический объект в физический файл на жестком диске, чтобы в будущем включить этот файл в отчёт; void SaveCanvasAsAnImage (FrameworkElement canvas, string fileName) – метод сохраняет графический программный объект в физическое изображение.



* 1. Класс описания взаимодействий с базой данных

Данный класс выполняет функцию моста между базой данных и остальными классами программы. С помощью его методов производятся все манипуляции над таблицами: чтение, запись, редактирование.

Так как данный класс не может быть статическим в силу наличия в нём полей, то необходимо будет создавать объект типа DatabaseControl для вызова какого-то из методов. Чтобы не создавать объекты во всех классах, где есть обращения к методам DatabaseControl, он создаётся один раз в инкапсулирующем приложение Windows Presentation Foundation классе Application[11] в качестве публичного поля.

Поля:

* + - * string ConnectionString – поле, которое хранит строку подключения к базе данных; qlConnection \_connection – поле содержит непосредственно подключение к самой базе данных.



Методы:

* + - * int NumberOfNotes (string tableName) – возвращает число записей в таблице с именем tableName;
      * void AddExperiment (Experiment experiment) – добавляет в базу данных эксперимент experiment, к слову, ради этого метода большинство полей класса Experiment являются свойствами, т.к. доступными в любой области проекта; void AddPossibilityDistribution (IReadOnlyList<int> possibilityDistribution) – существует как вложенный метод и выполняется «параллельно» с методом AddExperiment, чтобы внести в таблицу POSSIBILITY\_DISTRIBUTION соответствующее эксперименту распределение; void AddDynamic (Dynamic dynamic) - существует как вложенный метод и выполняется «параллельно» с методом AddExperiment, чтобы внести в таблицу DYNAMIC соответствующую



эксперименту динамику вероятности связности, математического ожидания степени вершины и математическое ожидание количества рёбер;

* + - * public List<Experiment> FindExperiments (int areaShapeFrom, int areaShapeTo, float areaHeightFrom, float areaHeightTo, float areaWidthFrom, float areaWidthTo, float radiusFrom, float radiusTo, int nodesCountFrom, int nodesCountTo, int graphsCountFrom, int graphsCountTo, float connectionPossibilityFrom, float connectionPossibilityTo, float degreeExpectationFrom, float degreeExpectationTo, float arcsExpectationFrom, float arcsExpectationTo) – метод находит в базе все записи, удовлетворяющие введённым параметрам, помещает их в список и выдаёт как результат – список экспериментов;
      * List<int> GetPossibilityDistributionByExperimentId (int id) – метод существует для извлечения массива распределения степеней вершин какого-то определенного эксперимента. Используется вместе с методом FindExperiments;
      * Dynamic GetDynamicByExperimentId (int id) - метод существует для извлечения массивов динамики какого-то определенного эксперимента. Используется вместе с методом FindExperiments; string GetTemplateById (int id) – вытаскивает из базы данных полностью собранный для внутренних подстановок шаблон; string GetTemplateChildrenById (int id) – вытаскивает список дочерних шаблонов для текущего шаблона;



* + - * string GetMarkByTemplateId (int id) – получает текстовую метку, вместо которой в родительский шаблон нужно вставить один из дочерних; string GetStringById (int id) – извлекает строковое значение из таблицы STRINGS.



* 1. Класс, описывающий логику взаимодействия с окном программы

Один из самых основных классов в приложении с пользовательским интерфейсом, так как именно он ведет обработку действий пользователя, контролирует его деятельность, сообщает об ошибках и связывает программные объекты с данными, которые поступают от пользователя.

Поля:

* + - * Random \_random – экземпляр класса рандом для генерации случайных чисел;
      * string \_operationCode – представляет собой последовательность из трёх чисел от 0 до 1: каждое из этих чисел означает, включен ли определенный набор параметров поиска (параметры области, параметры проведения эксперимента, параметры результатов эксперимента) в условия. Так, например, при значении operationCode = “001” при поиске будут учитываться только параметры результатов проведения экспериментов;
      * Experiment \_lastExperiment – хранит последний проведенный за запуск приложения эксперимент. Необходим для формирования и просмотра свежих отчётов по экспериментам;
      * Selection \_lastSelection - хранит последнюю проведенную за запуск

приложения выборку экспериментов. Необходим для формирования и просмотра свежих отчётов по выборке

экспериментов;

* + - * int \_areaShape – поле предназначено для хранения формы области;  float \_areaHeight – поле предназначено для хранения высоты области; float \_areaWidth – поле предназначено для хранения ширины области; float \_radius – поле предназначено для хранения радиуса действия вершины; int \_nodesCount – поле предназначено для хранения количества вершин в графе; int \_graphsCount – поле предназначено для хранения количества формируемых в эксперименте графов;



float \_areaHeightFrom – поле предназначено для хранения минимального значения высоты области; float \_areaHeightTo – поле предназначено для хранения максимального значения высоты области; float \_areaWidthFrom – поле предназначено для хранения минимального значения ширины области; float \_areaWidthTo – поле предназначено для хранения максимального значения ширины области; float \_radiusFrom – поле предназначено для хранения минимального значения радиуса действия вершины; float \_radiusTo – поле предназначено для хранения максимального значения радиуса действия вершины; int \_nodesCountFrom – поле предназначено для хранения минимального значения количества вершин в графах; int \_nodesCountTo – поле предназначено для хранения максимального значения количества вершин в графах; int \_graphsCountFrom – поле предназначено для хранения минимального значения количества графов в эксперименте; int \_graphsCountTo – поле предназначено для хранения максимального значения количества графов в эксперименте; float \_connectionPossibilityFrom– поле предназначено для хранения минимального значения вероятности связности;



* + - * float \_connectionPossibilityTo – поле предназначено для хранения максимального значения вероятности связности;
      * float \_degreeExpectationFrom – поле предназначено для хранения минимального значения математического ожидания степени вершины;

float \_degreeExpectationTo – поле предназначено для хранения максимального значения математического ожидания степени вершины;

* + - * float \_arcsExpectationFrom– поле предназначено для хранения минимального значения математического ожидания количества рёбер;
      * float \_arcsExpectationTo – поле предназначено для хранения максимального значения математического ожидания количества рёбер.

Примечание: Все значения полей, кроме \_lastExperiment и \_lastSelection

«тянутся» из соответствующих элементов управления TextBox на вкладках проведения и поиска экспериментов.

Методы:

* + - * MainWindow () – конструктор, который инициализирует все элементы, определения которых находятся в MainWidow.xaml; bool ReadAndValidateDataFromMakeForm () – считывает и проверяет корректность значений из элементов управления TextBox на вкладке проведения экспериментов. Возвращает значение true, если все данные введены пользователем корректно; bool ReadAndValidateDataFromSearchForm () – считывает и проверяет корректность значений из элементов управления TextBox на вкладке поиска экспериментов. Возвращает значение true, если все данные введены пользователем корректно; void MakeExperiment\_Click (object sender, RoutedEventArgs e) – обработчик кнопки «Провести эксперимент», внутри генерируется объект Experiment и помещается в \_lastExperiment;



* + - * void SeachExperiment\_Click (object sender, RoutedEventArgs e) – обработчик кнопки «Найти эксперимент», внутри генерируется

объект Selection и помещается в \_lastSelection;

void CheckBoxSearchParametrs\_Click(object sender,

RoutedEventArgs e) – обрабатывает отметки элементов управления CheckBox, затемняя или осветляя область используемых или не используемых параметров;

* + - * void RememberProperties () – метод вызывается при закрытии окна и помещает содержимое каждого из элементов управления TextBox в параметры проекта, чтобы введенные при закрытии окна данные были воспроизведены при открытии окна; void SetProperties () – метод, который при инициализации окна собирает все переменные из свойства проекта и «забивает» ими содержимое элементов управления TextBox; void ShowResults () – выводит на экран результат поиска экспериментов, то есть выборку \_lastSelection; void Window\_Closing(object sender, EventArgs e) – обработчик закрытия окна, вызывает метод RememberProperties (); void MakeReport\_Click (object sender, RoutedEventArgs e) – обработчик кнопки «Создать отчёт», запускает формирование отчёта от объекта \_lastExperiment или \_lastSelection; void ViewReport\_Click (object sender, RoutedEventArgs e) – обработчик кнопки «Просмотреть отчёт», запускает последний сформированный PDF-отчёт.



1. Результаты работы

В качестве результатов работы демонстрируется приложение, интерфейс которого позволяет работать с разработанной базой данных: добавлять туда данные, выбирать их и искать среди них подходящие.

Интерфейс приложения разделен на две части: проведение эксперимента и поиск эксперимента, которые представлены соответствующими вкладками, рисунки 5, 13.

* 1. Вкладка «Проведение эксперимента»

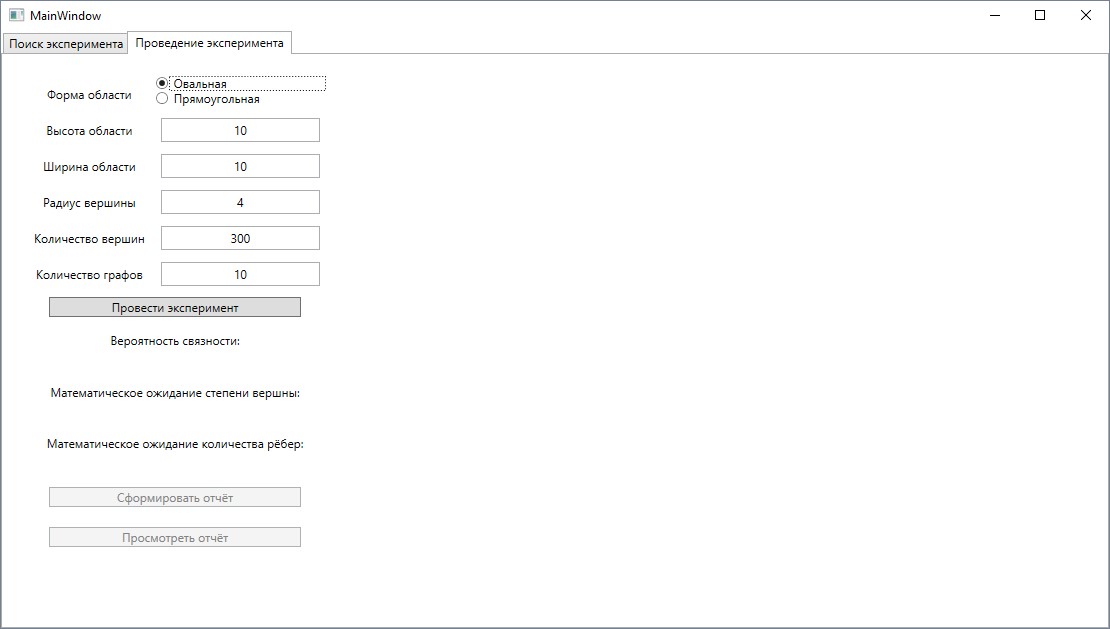


Рисунок 5 – Вкладка «Проведение экперимента»

На данной форме представлен функционал, позоляющий проводить эксперимент по следющим параметрам:

 форма области;  высота области;  ширина области;  радиус распространения сигнала вершины;  количество вершин;  количество графов.

Результаты проведения эксперимента (вероятность связности, математическое ожидание степеней вершин, математическое ожидание количества рёбер) пустые, а кнопки «Сформировать отчёт» и «Просмотреть отчёт» недоступны, так как не проведено ни одного экперимента.

Чтобы провести эксперимент, нужно нажать на кнопку «Провести эксперимент». Если какие-то данные введены некорректно, то программа выдаст сообщение об ошибке, рисунок 6.

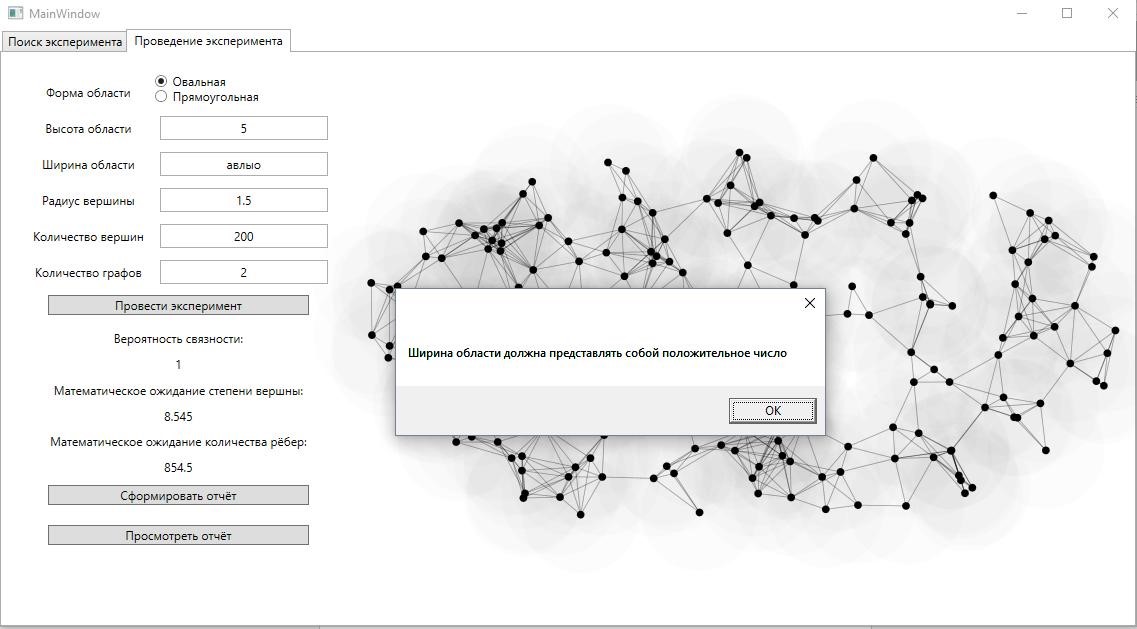


Рисунок 6 – Диалоговое окно ошибки на вкладке проведения эксперимента

Если данные введены корректно, то будет проведен эксперимент. Результаты его проведения будут отображены в нижнем левом углу, в правой части экрана появится эскиз смоделированной сети, а кнопка «Сформировать отчёт» станет доступной. Кнопка «Просмотреть отчёт» всё ещё недоступна, рисунок 7.

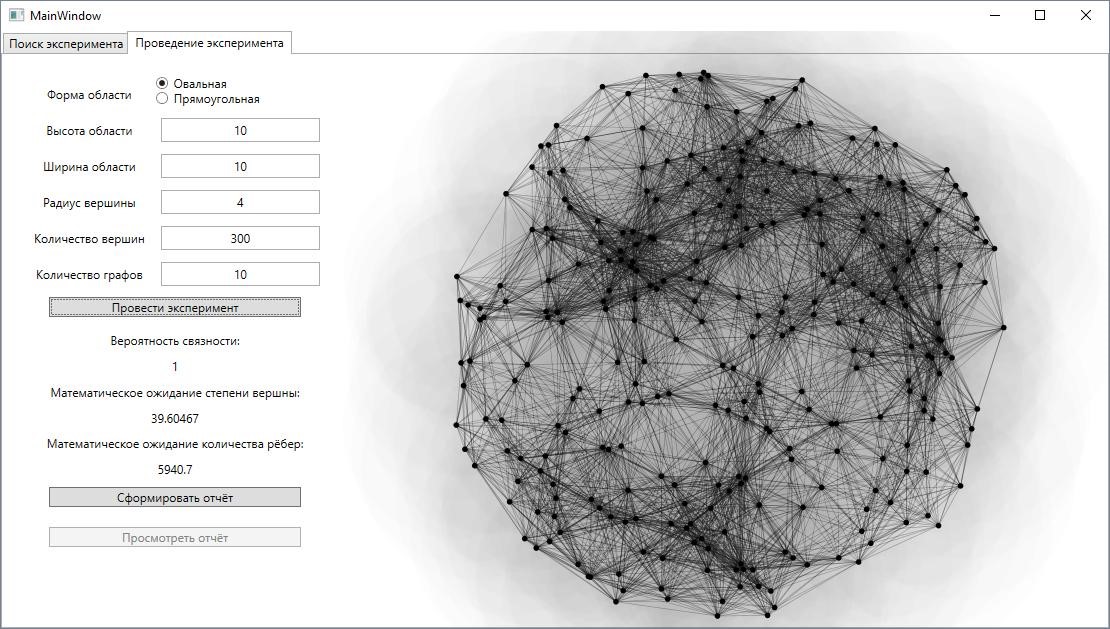


Рисунок 7 – Результаты проведенного эксперимента

При нажатии на кнопку «Сформировать отчёт» на жестком диске компьютера будет создан отчёт по проведенному эксперименту. Чтобы открыть файл отчёта, пользователю нужно кликнуть по кнопке «Просмотреть отчёт». На рисунке 8 представлен заголовок отчёта, вводная часть и параметры эксперимента, для которого он строился.

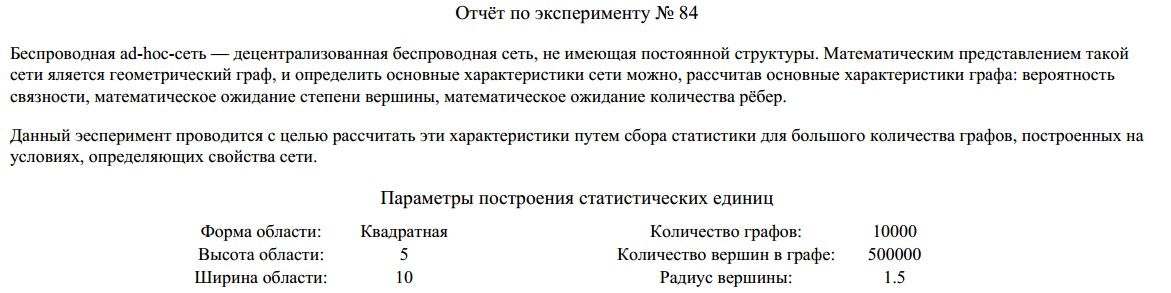


Рисунок 8 – Заголовок отчёта по эксперименту

Далее в отчёте рассматриваются основные характеристики сети: вероятность связности, математическое ожидание степени вершины, математическое ожидание количества рёбер и распределение степеней вершин. Для этих величин отображены формулы и динамика их изменения в процессе проведения эксперимента, рисунки 9-12.

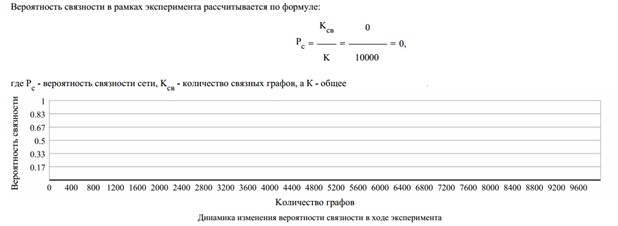


Рисунок 9 – Вероятность связности в отчёте для эксперимента

На графике видно, что для эксперимента с параметрами, изображенными на рисунке 8, вероятность связности как была равна нулю на первом графе, так им и осталась к формированию десятитысячного.

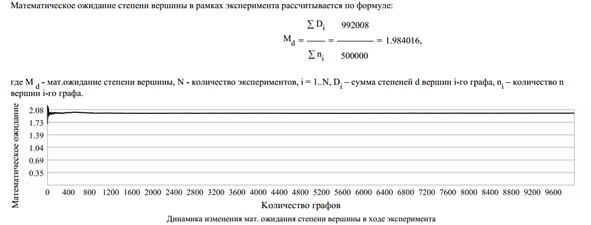


Рисунок 10 – Математическое ожидание степени вершины для эксперимента

По графику динамики можно заметить, что значение оставалось статичным уже на отметке ~1000 графов, из чего можно сделать вывод, что формирование числа моделей, превышающее данное значение в десять раз не является необходимостью

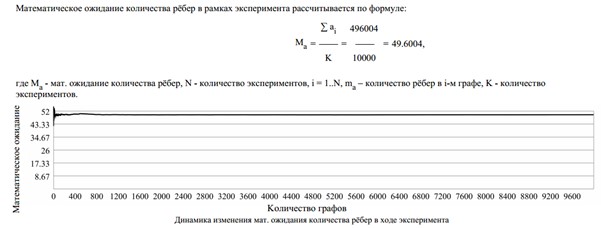


Рисунок 11 – Математическое ожидание количества рёбер в отчёте по эксперименту

На рисунке 11 чётко видно, что значение практически постоянное на той же отметке в ~1000 графов.

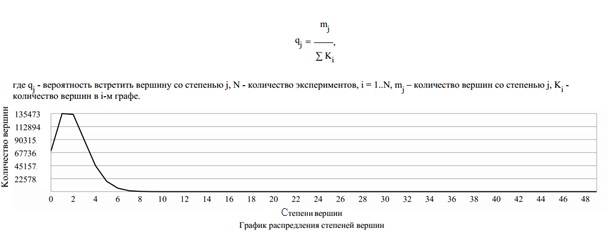


Рисунок 12 – Распределение степеней вершин в отчёте для эксперимента

4.2 Вкладка «Поиск эксперимента»

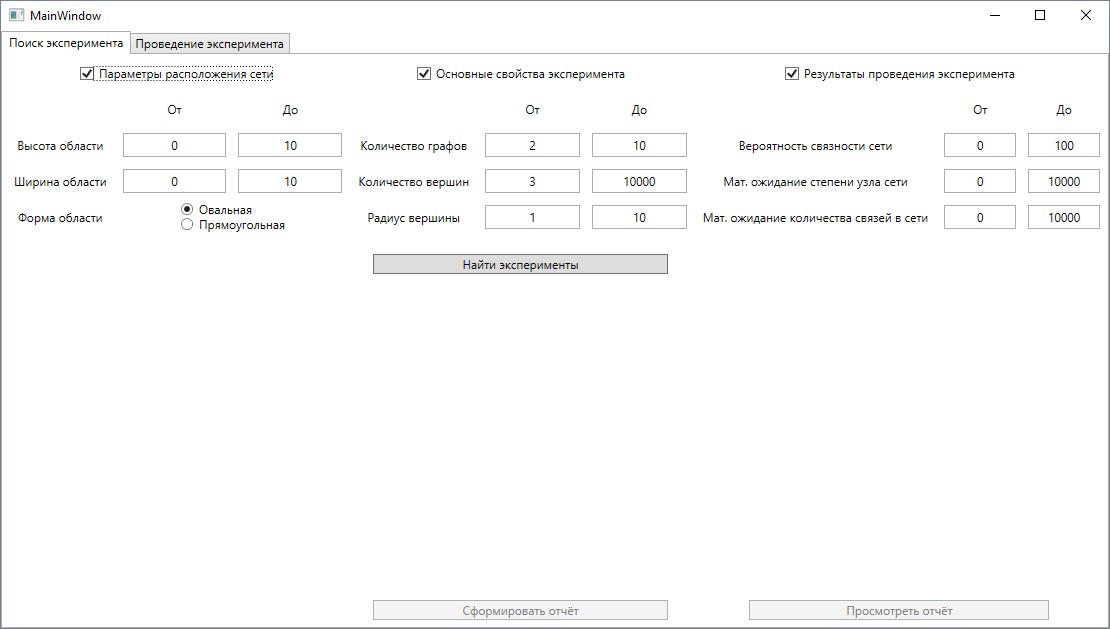


Рисунок 13 – Вкладка «Поиск эксперимента»

На данной вкладке преставлен функционал, позволяющий искать эксперименты по параметрам, разделенным на три раздела:

 параметры области;  параметры свойств эксперимента;  параметры результатов эксперимента.

Любой из разделов можно не учитывать при поиске, кликнув по соответствующему элементу управления TextBox, рисунок 14.

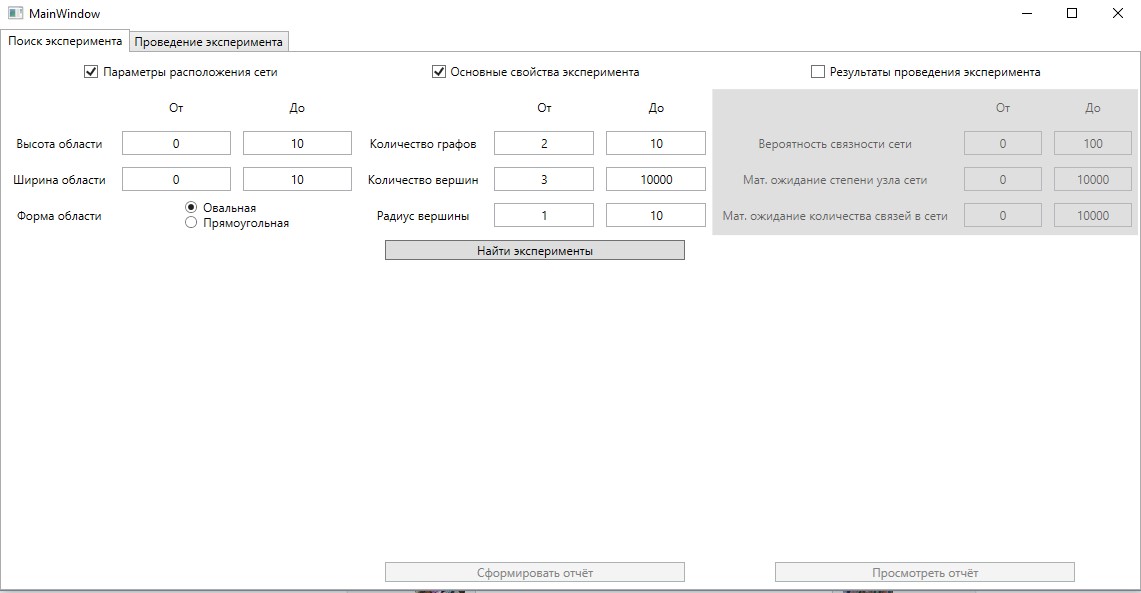


Рисунок 14 – Параметры поиска на вкладке «Поиск эксперимента»

Если же снять галочки со всех трёх разделов и попытаться найти эксперимент, программа выдаст сообщение об ошибке, рисунок 15.

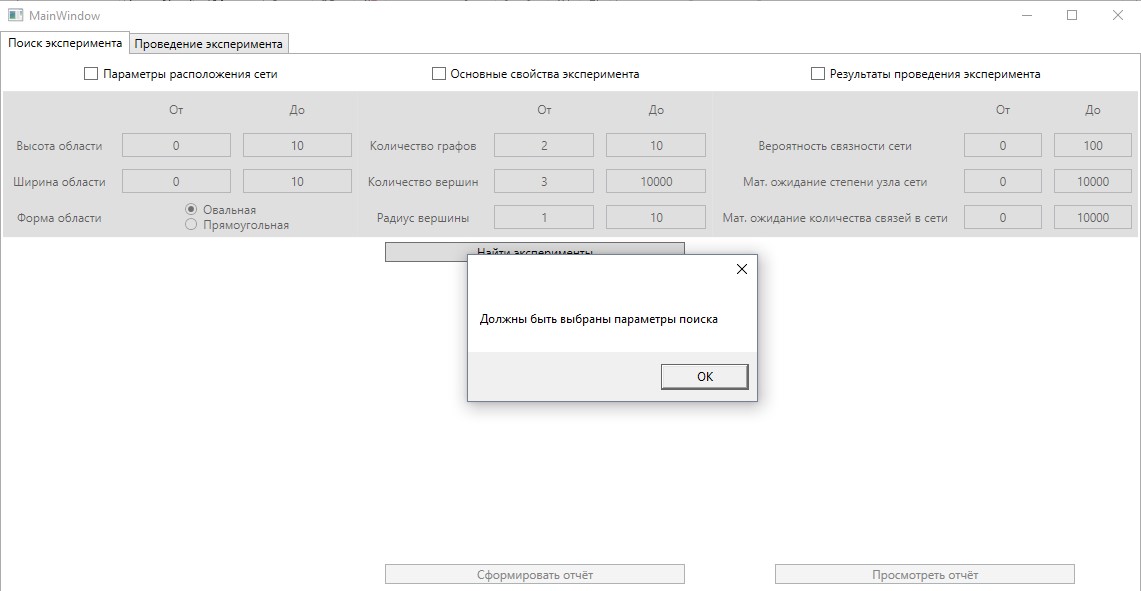


Рисунок 15 – Сообщение об ошибке при поиске без параметров

Механизм доступности кнопок работает аналигично с тем, что описан для вкладки «Проведение эксперимента».

Если пользователь ввёл все параметры корректно и кликнул по кнопке «Найти эксперимент», область в середине экрана заполнится результатми, удовлетворяющими введённым параметрам, рисунок 16.

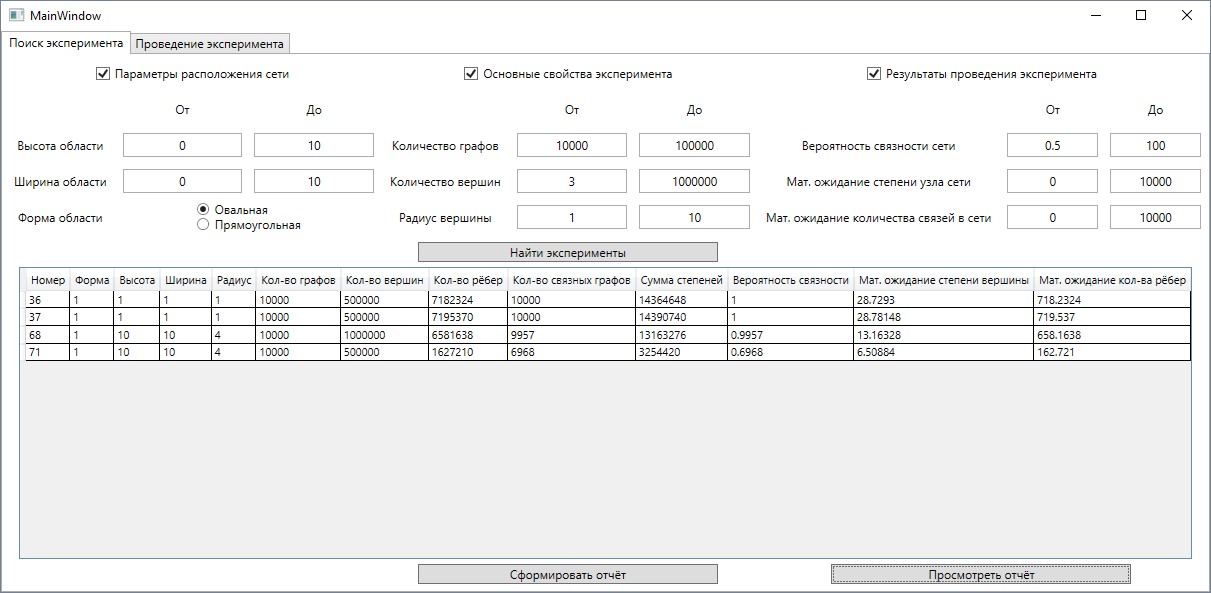


Рисунок 16 – Результаты поиска экспериментов

На рисунках 17-19 изображен отчёт по выборке экспериментов, сформированной в результате поиска.

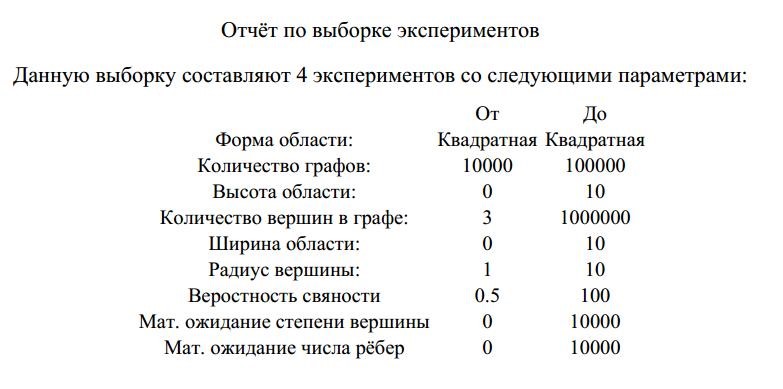


Рисунок 17 – Заголовок отчёта по выборке экспериментов

В заголовке отчёта указано, по каким характеристикам искались эксперименты.

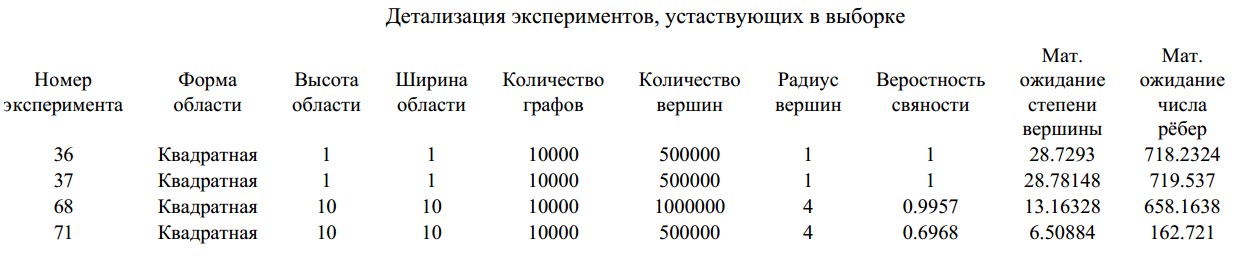


Рисунок 18 – Детализация экспериментов выборки в отчёте по выборке экспериментов

В детализации указаны локальные характеристики каждого эксперимента, входящего в выборку.

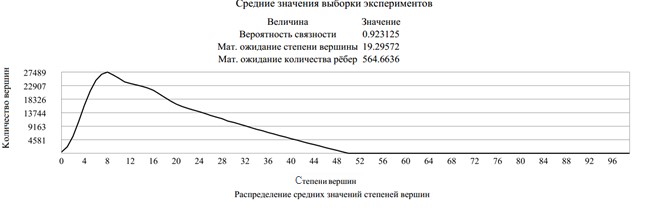


Рисунок 19 – Средние значения по выборке в отчёте по выборке экспериментов

Средние значения по выборке вычисляются с целью обобщить эксперименты, найденные по диапазонам значений параметров.

ЗАКЛЮЧЕНИЕ

В результате проведения работы были решены следующие задачи:

* рассмотрение параметров области расположения, количества

узлов и радиуса зоны покрытия отдельного узла в качестве основных свойств ad hoc сети;

* рассмотрение ключевых характеристик сети: связности, количества связей и количества смежных узлов для устройства;
* рассмотрение концепции формирования баз данных. В работе была спроектирована и реализована база данных, позволяющая хранить:  параметры сети;  основные характеристики сети;  функции как наборы значений;  переменные;  данные для построения графиков;  данные для построения отчётов.
* реализация алгоритм моделирования беспроводной

децентрализованной сети, обладающей заданными основными свойствами;

Был реализован алгоритм расчёта ключевых характеристик сети и разработана программа для взаимодействия с базой знаний для хранения основных свойств стохастических моделей беспроводных компьютерных ad hoc сетей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Миков А.И., Мезенцева А.С. Характеристики геометрических графов, моделирующих AD HOC сети // Информатизация и связь. – 2012. – № 5. – C. 85-88.
2. Миков А.И. Стохастические характеристики разрезов в графах AD HOC сетей // Информатизация и связь. – 2014. – № 1. – С. 76-79.
3. Замятина Е.Б., Миков А.И., Михеев Р.А. Особенности моделирования распределенных информационных систем // Вестник Пермского университета. Серия: Математика. Механика. Информатика. – 2013. – № 4. – С. 107-118.
4. Замятина Е.Б., Миков А.И. Программные средства системы имитации T ad.NET для обеспечения ее адаптируемости и открытости // Информатизация и связь. – 2012. – № 5. – C. 130-133.
5. Миков А.И. Связность автономных беспроводных компьютерных сетей в местностях с плохой инфраструктурой // Экологический вестник научных центров Черноморского экономического сотрудничества. – 2014. – № 1. – С. 70-75.
6. Волченская Т.В., Князьков В.Б. Введение в теорию графов // Алгоритмы и дискретные структуры. – 2008. – C. 125.
7. Hekmat R. Ad-hoc networks: Fundamental properties and network topologies; Technology, The Netherlands and Rhyzen information and consulting services, Zoetermeer, The Netherlamd, 2006.
8. Nykvist J., Phanse K. Modeling Connectivity in Mobile Ad-hoc Network

Environments, Department of Computer Science and Electrical Engineering, Lulea, Sweden. – P. 87-88.

1. Когаловский М.Р. Основы баз данных // Лаборатория знаний. – 2007. – С. 125.
2. Windows Presentation Foundation. [Электронный ресурс], URL: [www.telecom-master.ru,](https://vk.com/away.php?to=http%3A%2F%2Fwww.telecom-master.ru) [Дата обращения: 16 марта 2016].

ПРИЛОЖЕНИЕ A

Код программы

using System; using System.Collections.Generic; using System.Data; using System.Data.SqlClient; using System.Linq;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public class DatabaseControl

{

private static readonly string ConnectionString = Properties.Settings.Default.DatabaseLocation; private readonly SqlConnection \_connection = new SqlConnection(ConnectionString);

public int NumberOfNotes(string tableName)

{

var command = new SqlCommand("SELECT MAX(ID) FROM " + tableName, \_connection);

try

{

\_connection.Open(); var reader = command.ExecuteReader(CommandBehavior.SingleRow);

if (reader.Read())

{ return (int)reader[0];

}

return 0;

} finally

{

\_connection.Close();

}

}

public void AddExperiment(Experiment experiment)

{

var addExperimentNote =

new SqlCommand("INSERT INTO EXPERIMENTS (AREA\_SHAPE, AREA\_HEIGHT, AREA\_WIDTH,

NODES\_COUNT, " +

"RADIUS, GRAPHS\_COUNT, CONNECTED\_GRAPHS\_COUNT, DEGREES\_SUMM,

ARCS\_COUNT, " +

"CONNECTION\_POSSIBILITY, DEGREE\_EXPECTATION, ARCS\_EXPECTATION) VALUES ("

+

experiment.AreaShape + ", " +

experiment.AreaHeight + ", " + experiment.AreaWidth + ", " + experiment.NodesCount

+ ", " + experiment.NodeRadius + ", " + experiment.GraphsCount + ", " + experiment.ConnectedGraphsCount + ", " + experiment.DegreesSum + ", " + experiment.ArcsCount + ", " +

Math.Round(experiment.ConnectedPossibility, 5) + ", " +

Math.Round(experiment.DegreeExpectation, 5) + ", " +

Math.Round(experiment.ArcsExpectation, 5) + ")", \_connection)

{

CommandType = CommandType.Text

};

try

{

\_connection.Open(); addExperimentNote.ExecuteNonQuery();

} finally

{

\_connection.Close();

}

AddPossibilityDistribution(experiment.PossibilityDistribution);

AddDynamic(experiment.Dynamic);

}

private void AddPossibilityDistribution(IReadOnlyList<int> possibilityDistribution)

{

var experimentId = NumberOfNotes("EXPERIMENTS");

for (var i = 0; i < possibilityDistribution.Count; i++)

{

var addPossibilityDistributionNote = new SqlCommand("INSERT INTO POSSIBILITY\_DISTRIBUTION (EXPERIMENT\_ID, " +

$"DEGREE, NODES\_COUNT) VALUES ({experimentId}, {i}, {possibilityDistribution[i]})", \_connection)

{

CommandType = CommandType.Text

};

try

{

\_connection.Open(); addPossibilityDistributionNote.ExecuteNonQuery();

} finally

{

\_connection.Close();

}

}

}

private void AddDynamic(Dynamic dynamic)

{

var experimentId = NumberOfNotes("EXPERIMENTS");

for (var i = 0; i < dynamic.ConnectionPossibility.Count; i++)

{

var addDynamicNote =

new SqlCommand("INSERT INTO DYNAMIC (EXPERIMENT\_ID, GRAPHS\_COUNT,

CONNECTION\_POSSIBILITY, DEGREE\_EXPECTATION, " +

$"ARCS\_EXPECTATION) VALUES ({experimentId}, {i},

{dynamic.ConnectionPossibility[i]}, " +

$"{dynamic.DegreeExpectation[i]}, " +

$"{dynamic.ArcsExpectation[i]})", \_connection)

{

CommandType = CommandType.Text

};

try

{

\_connection.Open(); addDynamicNote.ExecuteNonQuery();

} finally

{

\_connection.Close();

}

}

}

public List<Experiment> FindExperiments(int areaShapeFrom, int areaShapeTo, float areaHeightFrom, float areaHeightTo, float areaWidthFrom, float areaWidthTo, float radiusFrom, float radiusTo, int nodesCountFrom, int nodesCountTo, int graphsCountFrom, int graphsCountTo, float connectionPossibilityFrom, float connectionPossibilityTo, float degreeExpectationFrom, float degreeExpectationTo, float arcsExpectationFrom, float arcsExpectationTo)

{

var findExperiments = new SqlCommand(

$"SELECT \* FROM EXPERIMENTS WHERE AREA\_SHAPE >= {areaShapeFrom} AND AREA\_SHAPE

<= {areaShapeTo} AND" +

$" NODES\_COUNT >= {nodesCountFrom} AND NODES\_COUNT <= {nodesCountTo} AND

RADIUS >= {radiusFrom} AND RADIUS <= {radiusTo} AND GRAPHS\_COUNT >=" +

$" {graphsCountFrom} AND GRAPHS\_COUNT <= {graphsCountTo} AND AREA\_WIDTH >=

{areaWidthFrom} AND AREA\_WIDTH <= {areaWidthTo} AND AREA\_HEIGHT >=" +

$" {areaHeightFrom} AND AREA\_HEIGHT <= {areaHeightTo} AND CONNECTION\_POSSIBILITY

>= {connectionPossibilityFrom} AND CONNECTION\_POSSIBILITY <=" +

$" {connectionPossibilityTo} AND DEGREE\_EXPECTATION >= {degreeExpectationFrom} AND

DEGREE\_EXPECTATION <= {degreeExpectationTo} AND" +

$" ARCS\_EXPECTATION >= {arcsExpectationFrom} AND ARCS\_EXPECTATION <= {arcsExpectationTo}",

\_connection) {CommandType = CommandType.Text};

var experiments = new List<Experiment>();

try

{

\_connection.Open(); var reader = findExperiments.ExecuteReader();

while (reader.Read())

{

var possibilityDistribution = new List<int>(); var dynamic = new Dynamic();

experiments.Add(new Experiment((int)reader["ID"], (int)reader["AREA\_SHAPE"],

(float)reader["AREA\_HEIGHT"], (float)reader["AREA\_WIDTH"],

(float)reader["RADIUS"], (int)reader["GRAPHS\_COUNT"], (int)reader["NODES\_COUNT"], (int)reader["ARCS\_COUNT"],

(int)reader["CONNECTED\_GRAPHS\_COUNT"], (int)reader["DEGREES\_SUMM"], possibilityDistribution, dynamic));

}

reader.Close();

} finally

{

\_connection.Close();

}

foreach (var experiment in experiments)

{

experiment.PossibilityDistribution = GetPossibilityDistributionByExperimentId(experiment.Id); experiment.Dynamic = GetDynamicByExperimentId(experiment.Id);

}

return experiments;

}

private List<int> GetPossibilityDistributionByExperimentId(int id)

{

var getPossibilityDistributon =

new SqlCommand($"SELECT \* FROM POSSIBILITY\_DISTRIBUTION WHERE EXPERIMENT\_ID = {id}", \_connection)

{

CommandType = CommandType.Text

};

var distribution = new List<int>();

try

{

\_connection.Open();

var reader = getPossibilityDistributon.ExecuteReader(); while (reader.Read()) distribution.Add((int) reader["NODES\_COUNT"]); return distribution;

} finally

{

\_connection.Close();

}

}

private Dynamic GetDynamicByExperimentId(int id)

{

var getDynamic = new SqlCommand($"SELECT \* FROM DYNAMIC WHERE EXPERIMENT\_ID = {id}", \_connection)

{

CommandType = CommandType.Text

};

var connectionPossibility = new List<float>(); var degreeExpectation = new List<float>(); var arcsExpectation = new List<float>();

try

{

\_connection.Open();

var reader = getDynamic.ExecuteReader(); while (reader.Read())

{

connectionPossibility.Add((float)(double)reader["CONNECTION\_POSSIBILITY"]); degreeExpectation.Add((float)(double)reader["DEGREE\_EXPECTATION"]); arcsExpectation.Add((float)(double)reader["ARCS\_EXPECTATION"]);

}

var dynamic = new Dynamic

{

ConnectionPossibility = connectionPossibility,

DegreeExpectation = degreeExpectation,

ArcsExpectation = arcsExpectation

};

return dynamic;

} finally

{

\_connection.Close();

}

}

public string GetTemplateById(int id)

{

var getTemplate = new SqlCommand($"SELECT \* FROM TEMPLATES WHERE ID = {id}", \_connection)

{

CommandType = CommandType.Text

};

var stringId = 0;

try

{

\_connection.Open();

var reader = getTemplate.ExecuteReader(); while (reader.Read())

{

stringId = (int) reader["BODY\_STRING\_ID"];

} } finally

{

\_connection.Close();

}

var mainTemplate = GetStringById(stringId);

if (GetTemplateChildrenById(id) != " ") mainTemplate =

GetTemplateChildrenById(id).Split(',').Select(int.Parse).Aggregate(mainTemplate,

(current, childId) => current.Replace(GetMarkByTemplateId(childId), GetStringById(childId)));

return mainTemplate;

}

private string GetTemplateChildrenById(int id)

{

var getChildrens = new SqlCommand($"SELECT \* FROM TEMPLATES WHERE ID = {id}", \_connection)

{

CommandType = CommandType.Text

};

var childrens = "";

try

{

\_connection.Open();

var reader = getChildrens.ExecuteReader(); while (reader.Read())

{

childrens = (string)reader["CHILDREN"];

}

return childrens;

} finally {

\_connection.Close();

}

}

private string GetMarkByTemplateId(int id)

{

var getMarkStringId = new SqlCommand($"SELECT \* FROM TEMPLATES WHERE ID = {id}", \_connection)

{

CommandType = CommandType.Text

};

var markStringId = 0;

try

{

\_connection.Open();

var reader = getMarkStringId.ExecuteReader(); while (reader.Read())

{

markStringId = (int)reader["MARK\_STRING\_ID"];

} } finally

{

\_connection.Close();

}

return GetStringById(markStringId);

}

public string GetStringById(int id)

{

var getString =

new SqlCommand($"SELECT \* FROM STRINGS WHERE ID = {id}", \_connection)

{

CommandType = CommandType.Text

};

var stringValue = "";

try

{

\_connection.Open();

var reader = getString.ExecuteReader(); while (reader.Read())

{

stringValue = (string)reader["VALUE"];

}

return stringValue;

} finally

{

\_connection.Close();

}

}

public void CleanTable(string tableName)

{

var command = new SqlCommand($"TRUNCATE TABLE {tableName}", \_connection)

{

CommandType = CommandType.Text

};

try

{

\_connection.Open(); command.ExecuteNonQuery();

} finally

{

\_connection.Close();

}

}

}

}

using System.Collections.Generic;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public class Dynamic

{

public List<float> ConnectionPossibility { get; set; } public List<float> DegreeExpectation { get; set; } public List<float> ArcsExpectation { get; set; }

public Dynamic()

{

ConnectionPossibility = new List<float>();

DegreeExpectation = new List<float>();

ArcsExpectation = new List<float>();

}

public void GeneratePlots()

{

Painter.PaintPlot(ConnectionPossibility, App.DataBaseControl.GetStringById(5));

Painter.PaintPlot(DegreeExpectation, App.DataBaseControl.GetStringById(6)); Painter.PaintPlot(ArcsExpectation, App.DataBaseControl.GetStringById(7));

}

}

}

using System;

using System.Collections.Generic; using System.Globalization; using System.Linq;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public class Experiment

{ public int Id { get; } public int AreaShape { get; } public float AreaHeight { get; } public float AreaWidth { get; } public float NodeRadius { get; } public int GraphsCount { get; } public int NodesCount { get; } public int ArcsCount { get; } public int ConnectedGraphsCount { get; } public int DegreesSum { get; }

public float ConnectedPossibility { get; } public float DegreeExpectation { get; } public float ArcsExpectation { get; }

public Dynamic Dynamic { get; set; } public List<int> PossibilityDistribution { get; set; }

public Experiment(int id, int areaShape, float areaHeight, float areaWidth, float nodeRadius, int graphsCount, int nodesCount,

int arcsCount, int connectedGraphsCount, int degreeSum, List<int> possibilityDistribution, Dynamic dynamic)

{

Id = id;

AreaShape = areaShape;

AreaHeight = areaHeight;

AreaWidth = areaWidth;

NodeRadius = nodeRadius;

GraphsCount = graphsCount;

NodesCount = nodesCount;

ArcsCount = arcsCount;

ConnectedGraphsCount = connectedGraphsCount;

DegreesSum = degreeSum;

PossibilityDistribution = possibilityDistribution;

Dynamic = dynamic;

ConnectedPossibility = (float)connectedGraphsCount / graphsCount;

DegreeExpectation = (float)degreeSum / nodesCount;

ArcsExpectation = (float)arcsCount / graphsCount;

}

public static Experiment MakeExperiment(int areaShape, float areaHeight, float areaWidth, float radius, int nodesCount, int graphsCount, Random random)

{

// Переменные для ведения статистики и отражения этой статистики в свойствах отдельного эксперимента

var connectedGraphsCount = 0; var nodesCountInAllGraphs = 0; var degreeSumInAllGraphs = 0; var arcsCountInAllGraphs = 0.0; //Переменная имеет тип double, поскольку в процессе вычисления для одной вершины используется только половина

//степени(граф неориентированный)

var dynamic = new Dynamic(); var degrees = new List<int>(Enumerable.Repeat(0, nodesCount));

// Цикл сбора статистики

for (var i = 0; i < graphsCount; i++)

{

var graph = new Graph(areaShape, areaHeight, areaWidth, radius, nodesCount, random);

if (graph.IsConnected()) connectedGraphsCount++; dynamic.ConnectionPossibility.Add((float)connectedGraphsCount / (i + 1));

graph.NodeDegreeExpectation(ref degreeSumInAllGraphs, ref nodesCountInAllGraphs); dynamic.DegreeExpectation.Add((float)degreeSumInAllGraphs / nodesCountInAllGraphs);

graph.NumberOfArcExpectation(ref arcsCountInAllGraphs); dynamic.ArcsExpectation.Add((float)arcsCountInAllGraphs / (i + 1));

graph.ProbabilityDistribution(ref degrees);

}

var experiment = new Experiment(App.DataBaseControl.NumberOfNotes("EXPERIMENTS") + 1, areaShape, areaHeight, areaWidth, radius, graphsCount, nodesCountInAllGraphs, (int)arcsCountInAllGraphs, connectedGraphsCount, degreeSumInAllGraphs, degrees, dynamic);

// Вывод в базу данных

App.DataBaseControl.AddExperiment(experiment);

return experiment;

}

public Dictionary<string, string> GetReplaceItemsForReport()

{

var areaShape = AreaShape == 0 ? "Овальная" : "Квадратная";

return new Dictionary<string, string>()

{

{"[experimentId]", Id.ToString()},

{"[areaShape]", areaShape},

{"[graphsCount]", GraphsCount.ToString()},

{"[areaHeight]", AreaHeight.ToString(CultureInfo.InvariantCulture)},

{"[areaWidth]", AreaWidth.ToString(CultureInfo.InvariantCulture)},

{"[nodesCount]", NodesCount.ToString()},

{"[radius]", NodeRadius.ToString(CultureInfo.InvariantCulture)},

{"[connectedGraphsCount]", ConnectedGraphsCount.ToString()},

{"[connectedPossibility]", ConnectedPossibility.ToString(CultureInfo.InvariantCulture)},

{"[degreeSum]", DegreesSum.ToString()},

{"[degreeExpectation]", DegreeExpectation.ToString(CultureInfo.InvariantCulture)},

{"[arcsCount]", ArcsCount.ToString()},

{"[arcsExpectation]", ArcsExpectation.ToString(CultureInfo.InvariantCulture)},

{$"[{App.DataBaseControl.GetStringById(5)}]", $"{App.DataBaseControl.GetStringById(5)}.png"},

{$"[{App.DataBaseControl.GetStringById(6)}]", $"{App.DataBaseControl.GetStringById(6)}.png"},

{$"[{App.DataBaseControl.GetStringById(7)}]", $"{App.DataBaseControl.GetStringById(7)}.png"},

{$"[{App.DataBaseControl.GetStringById(8)}]", $"{App.DataBaseControl.GetStringById(8)}.png"}

};

}

private void MakePlots()

{

Painter.PaintPlot(PossibilityDistribution.Select<int, float>(i => i).ToList(), App.DataBaseControl.GetStringById(8));

Dynamic.GeneratePlots();

}

public void MakeReport()

{

MakePlots();

var template = App.DataBaseControl.GetTemplateById(5).Replace("[reportDate]", DateTime.Today.ToShortDateString());

template = GetReplaceItemsForReport().Aggregate(template, (current, replaceItem) => current.Replace(replaceItem.Key, replaceItem.Value));

Report.GeneratePdf(template, $"{App.DataBaseControl.GetStringById(17)}{Id}");

}

}

}

using System;

using System.Collections.Generic; using System.Linq;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public class Graph

{

private readonly int \_areaShape; public readonly float AreaWidth; public readonly float AreaHeight; private readonly float \_radius; public List<Node> Nodes; private readonly Random \_random;

public Graph(int areaShape, float areaHeight, float areaWidth, float radius, int nodesCount, Random random)

{

\_areaShape = areaShape;

AreaHeight = areaHeight;

AreaWidth = areaWidth;

\_radius = radius;

\_random = random;

Nodes = new List<Node>();

AddNodes(nodesCount);

}

public bool IsConnected()

{

var list = new List<bool>();

for (var i = 0; i < Nodes.Count; i++) list.Add(false); Colorize(list, 0);

return list.All(item => item); /\*foreach (var item in list) if (!item) return false;\*/ }

void Colorize(IList<bool> list, int i)

{ if (list[i]) return; else list[i] = true;

foreach (var item in Nodes[i].Arclist)

Colorize(list, Nodes.IndexOf(item));

}

void AddNodes(int count)

{

for (var i = 0; i < count; i++)

Nodes.Add(new Node(\_areaShape, AreaHeight, AreaWidth, \_radius, ref Nodes, \_random));

}

public void NodeDegreeExpectation(ref int degreeSum, ref int numberOfNodes)

{

degreeSum += Nodes.Sum(item => item.Arclist.Count); numberOfNodes += Nodes.Count;

}

public void NumberOfArcExpectation(ref double numberOfArcs)

{

numberOfArcs += Nodes.Sum(item => item.Arclist.Count\*0.5);

}

public void ProbabilityDistribution(ref List<int> degrees)

{

foreach (var item in Nodes) degrees[item.Arclist.Count]++;

}

}

} using System;

using System.Globalization; using System.Windows; using System.Windows.Controls; using System.Diagnostics;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow

{

private readonly Random \_random; private string \_operationCode; private Experiment \_lastExperiment; private Selection \_lastSelection;

// Переменные для хранения параметров проведения эксперимента private int \_areaShape; private float \_areaHeight; private float \_areaWidth; private float \_radius; private int \_nodesCount; private int \_graphsCount;

// Переменные для хранения параметров поиска эксперимента

// По области private float \_areaHeightFrom; private float \_areaHeightTo; private float \_areaWidthFrom; private float \_areaWidthTo;

// По параметрам эксперимента private float \_radiusFrom; private float \_radiusTo; private int \_nodesCountFrom; private int \_nodesCountTo; private int \_graphsCountFrom; private int \_graphsCountTo;

// По результатам эксперимента private float \_connectionPossibilityFrom; private float \_connectionPossibilityTo; private float \_degreeExpectationFrom; private float \_degreeExpectationTo; private float \_arcsExpectationFrom; private float \_arcsExpectationTo;

public MainWindow()

{

InitializeComponent();

\_random = new Random();

SetProperties();

}

private bool ReadAndValidateDataFromMakeForm()

{

\_areaShape = RadioButtonRoundMake.IsChecked == true ? 1 : 0;

if (!float.TryParse(TextBoxAreaHeight.Text, out \_areaHeight) ||

(float.TryParse(TextBoxAreaHeight.Text, out \_areaHeight) && \_areaHeight <= 0.0))

{

MessageBox.Show("Высота области должна представлять собой положительное число"); return false;

}

if (!float.TryParse(TextBoxAreaWidth.Text, out \_areaWidth) || (float.TryParse(TextBoxAreaWidth.Text, out \_areaWidth) && \_areaWidth <= 0.0))

{

MessageBox.Show("Ширина области должна представлять собой положительное число"); return false;

}

if (!float.TryParse(TextBoxRadius.Text, out \_radius) || (float.TryParse(TextBoxRadius.Text, out \_radius) && \_radius <= 0.0))

{

MessageBox.Show("Радиус вершины должен представлять собой положительное число"); return false;

}

if (!int.TryParse(TextBoxNodesCount.Text, out \_nodesCount) ||

(int.TryParse(TextBoxNodesCount.Text, out \_nodesCount) && \_nodesCount < 2))

{

MessageBox.Show("Количество вершин должно быть положительным целым числом больше 2");

return false;

}

if (!int.TryParse(TextBoxGraphsCount.Text, out \_graphsCount) ||

(int.TryParse(TextBoxGraphsCount.Text, out \_graphsCount) && \_graphsCount <= 1))

{

MessageBox.Show("Количество графов должно быть положительным целым числом больше 1");

return false;

}

return true;

}

private bool ReadAndValidateDataFromSearchForm()

{

var fieldParametrs = "0"; var experimentParametrs = "0"; var experimentResultsParametrs = "0"; if (CheckBoxFieldParametrs.IsChecked == true)

{

fieldParametrs = "1";

if (!float.TryParse(TextBoxAreaHeightFrom.Text, out \_areaHeightFrom) ||

(float.TryParse(TextBoxAreaHeightFrom.Text, out \_areaHeightFrom) && \_areaHeightFrom < 0.0))

{

MessageBox.Show("Высота области должна представлять собой положительное число"); return false;

}

if (!float.TryParse(TextBoxAreaHeightTo.Text, out \_areaHeightTo) ||

(float.TryParse(TextBoxAreaHeightTo.Text, out \_areaHeightTo) && (\_areaHeightTo < 0.0 || \_areaHeightTo < \_areaHeightFrom)))

{

MessageBox.Show("Высота области должна представлять собой положительное число, значение От должно быть меньше значения До"); return false;

}

if (!float.TryParse(TextBoxAreaWidthFrom.Text, out \_areaWidthFrom) ||

(float.TryParse(TextBoxAreaWidthFrom.Text, out \_areaWidthFrom) && \_areaWidthFrom < 0.0))

{

MessageBox.Show("Ширина области должна представлять собой положительное число"); return false;

}

if (!float.TryParse(TextBoxAreaWidthTo.Text, out \_areaWidthTo) ||

(float.TryParse(TextBoxAreaWidthTo.Text, out \_areaWidthTo) && (\_areaWidthTo < 0.0 || \_areaWidthTo < \_areaWidthFrom)))

{

MessageBox.Show("Ширина области должна представлять собой положительное число, значение От должно быть меньше значения До"); return false;

}

\_areaShape = RadioButtonRoundSearch.IsChecked == true ? 1 : 0;

}

if (CheckBoxExperimentParametrs.IsChecked == true)

{

experimentParametrs = "1";

if (!float.TryParse(TextBoxRadiusFrom.Text, out \_radiusFrom) ||

(float.TryParse(TextBoxRadiusFrom.Text, out \_radiusFrom) && \_radiusFrom < 0.0))

{

MessageBox.Show("Радиус вершины должен представлять собой положительное число"); return false;

}

if (!float.TryParse(TextBoxRadiusTo.Text, out \_radiusTo) || (float.TryParse(TextBoxRadiusTo.Text, out \_radiusTo) && (\_radiusTo < 0.0 ||

\_radiusTo < \_radiusFrom)))

{

MessageBox.Show("Радиус вершины должен представлять собой положительное число, значение От должно быть меньше значения До");

return false;

}

if (!int.TryParse(TextBoxNodesCountFrom.Text, out \_nodesCountFrom) ||

(int.TryParse(TextBoxNodesCountFrom.Text, out \_nodesCountFrom) && \_nodesCountFrom < 0))

{

MessageBox.Show("Количество вершин должно быть положительным целым числом"); return false;

}

if (!int.TryParse(TextBoxNodesCountTo.Text, out \_nodesCountTo) ||

(int.TryParse(TextBoxNodesCountTo.Text, out \_nodesCountTo) && (\_nodesCountTo < 0 || \_nodesCountTo < \_nodesCountFrom)))

{

MessageBox.Show("Количество вершин должно быть положительным целым числом, значение От должно быть меньше значения До"); return false;

}

if (!int.TryParse(TextBoxGraphsCountFrom.Text, out \_graphsCountFrom) ||

(int.TryParse(TextBoxGraphsCountFrom.Text, out \_graphsCountFrom) && \_graphsCountFrom < 0))

{

MessageBox.Show("Количество графов должно быть положительным целым числом"); return false;

}

if (!int.TryParse(TextBoxGraphsCountTo.Text, out \_graphsCountTo) ||

(int.TryParse(TextBoxGraphsCountTo.Text, out \_graphsCountTo) && (\_graphsCountTo < 0 || \_graphsCountTo < \_graphsCountFrom)))

{

MessageBox.Show("Количество графов должно быть положительным целым числом, значение От должно быть меньше значения До"); return false;

}

}

if (CheckBoxExperimentResultsParametrs.IsChecked == true)

{

experimentResultsParametrs = "1";

if (!float.TryParse(TextBoxConnectionPossibilityFrom.Text, out \_connectionPossibilityFrom) ||

(float.TryParse(TextBoxConnectionPossibilityFrom.Text, out \_connectionPossibilityFrom) && \_connectionPossibilityFrom < 0.0))

{

MessageBox.Show("Вероятность связности должна быть положительным числом"); return false;

}

if (!float.TryParse(TextBoxConnectionPossibilityTo.Text, out \_connectionPossibilityTo) ||

(float.TryParse(TextBoxConnectionPossibilityTo.Text, out \_connectionPossibilityTo) &&

(\_connectionPossibilityTo < 0.0 ||

\_connectionPossibilityTo < \_connectionPossibilityFrom)))

{

MessageBox.Show("Вероятность связности должна быть положительным числом, значение От должно быть меньше значения До"); return false;

}

if (!float.TryParse(TextBoxDegreeExpectationFrom.Text, out \_degreeExpectationFrom) ||

(float.TryParse(TextBoxDegreeExpectationFrom.Text, out \_degreeExpectationFrom) && \_degreeExpectationFrom < 0.0))

{

MessageBox.Show("Математическое ожидание степени вершины должно быть положительным числом");

return false;

}

if (!float.TryParse(TextBoxDegreeExpectationTo.Text, out \_degreeExpectationTo) ||

(float.TryParse(TextBoxDegreeExpectationTo.Text, out \_degreeExpectationTo) && (\_degreeExpectationTo < 0.0 || \_degreeExpectationTo < \_degreeExpectationFrom)))

{

MessageBox.Show("Математическое ожидание степени вершины должно быть положительным числом, значение От должно быть меньше значения До"); return false;

}

if (!float.TryParse(TextBoxArcsExpectationFrom.Text, out \_arcsExpectationFrom) ||

(float.TryParse(TextBoxArcsExpectationFrom.Text, out \_arcsExpectationFrom) && \_arcsExpectationFrom < 0.0))

{

MessageBox.Show("Математическое ожидание количества рёбер должно быть положительным числом");

return false;

}

if (!float.TryParse(TextBoxArcsExpectationTo.Text, out \_arcsExpectationTo) ||

(float.TryParse(TextBoxArcsExpectationTo.Text, out \_arcsExpectationTo) && (\_arcsExpectationTo < 0.0 || \_arcsExpectationTo < \_arcsExpectationFrom)))

{

MessageBox.Show("Математическое ожидание количества рёбер должно быть положительным числом, значение От должно быть меньше значения До"); return false;

}

}

if (CheckBoxFieldParametrs.IsChecked != true && CheckBoxExperimentParametrs.IsChecked != true && CheckBoxExperimentResultsParametrs.IsChecked != true)

{

MessageBox.Show("Должны быть выбраны параметры поиска"); return false;

}

\_operationCode = fieldParametrs + experimentParametrs + experimentResultsParametrs;

return true;

}

private void MakeExperiment\_Click(object sender, RoutedEventArgs e)

{

// Валидация if (!ReadAndValidateDataFromMakeForm()) return; // Генерация эксперимента var newExperiment = Experiment.MakeExperiment

(\_areaShape, \_areaHeight, \_areaWidth, \_radius, \_nodesCount, \_graphsCount, \_random);

//Заполнение результирующих полей TextBlockConnectionPossibility.Text =

newExperiment.ConnectedPossibility.ToString(CultureInfo.InvariantCulture); TextBlockDegreeExpectation.Text =

newExperiment.DegreeExpectation.ToString(CultureInfo.InvariantCulture); TextBlockArcsExpectation.Text = newExperiment.ArcsExpectation.ToString(CultureInfo.InvariantCulture);

// Отрисовка изображений

Painter.PaintGraph(new Graph(\_areaShape, \_areaHeight, \_areaWidth, \_radius, \_nodesCount, \_random), CanvasGraphDemo);

\_lastExperiment = newExperiment;

ButtonExperimentMakeReport.IsEnabled = true;

}

private void SeachExperiment\_Click(object sender, RoutedEventArgs e)

{

//Валидация if (!ReadAndValidateDataFromSearchForm()) return;

// Поиск подходящих экспериментов в бд

\_lastSelection = Selection.FindExperiments(\_areaShape, \_areaHeightFrom, \_areaHeightTo,

\_areaWidthFrom, \_areaWidthTo,

\_radiusFrom, \_radiusTo, \_nodesCountFrom, \_nodesCountTo, \_graphsCountFrom,

\_graphsCountTo, \_connectionPossibilityFrom, \_connectionPossibilityTo,

\_degreeExpectationFrom, \_degreeExpectationTo, \_arcsExpectationFrom, \_arcsExpectationTo, \_operationCode);

// Вывод результатов поиска на экран

if (\_lastSelection.ExperimentSelection.Count == 0)

MessageBox.Show("Не найдено ни одного эксперимента с данными параметрами"); else

ShowResults();

ButtonSelectionMakeReport.IsEnabled = true;

}

private void CheckBoxSearchParametrs\_Click(object sender, RoutedEventArgs e)

{

switch (((CheckBox)sender).Name)

{

case "CheckBoxFieldParametrs":

BorderFieldParametrs.Visibility = BorderFieldParametrs.Visibility == Visibility.Collapsed

? Visibility.Visible : Visibility.Collapsed; break;

case "CheckBoxExperimentParametrs":

BorderExperimentParametrs.Visibility = BorderExperimentParametrs.Visibility == Visibility.Collapsed

? Visibility.Visible : Visibility.Collapsed; break; case "CheckBoxExperimentResultsParametrs":

BorderExperimentResultsParametrs.Visibility = BorderExperimentResultsParametrs.Visibility ==

Visibility.Collapsed

? Visibility.Visible : Visibility.Collapsed; break;

}

}

private void SetProperties()

{

//Вкладка проведения эксперимента

TextBoxAreaWidth.Text = Properties.Settings.Default.areaWidth;

TextBoxAreaHeight.Text = Properties.Settings.Default.areaHeight;

TextBoxRadius.Text = Properties.Settings.Default.radius;

TextBoxNodesCount.Text = Properties.Settings.Default.nodesCount;

TextBoxGraphsCount.Text = Properties.Settings.Default.graphsCount;

//Вкладка поиска экспериментов

TextBoxAreaWidthFrom.Text = Properties.Settings.Default.areaWidthFrom;

TextBoxAreaWidthTo.Text = Properties.Settings.Default.areaWidthTo;

TextBoxAreaHeightFrom.Text = Properties.Settings.Default.areaHeightFrom;

TextBoxAreaHeightTo.Text = Properties.Settings.Default.areaHeightTo;

TextBoxRadiusFrom.Text = Properties.Settings.Default.radiusFrom;

TextBoxRadiusTo.Text = Properties.Settings.Default.radiusTo;

TextBoxNodesCountFrom.Text = Properties.Settings.Default.nodesCountFrom;

TextBoxNodesCountTo.Text = Properties.Settings.Default.nodesCountTo;

TextBoxGraphsCountFrom.Text = Properties.Settings.Default.graphsCountFrom;

TextBoxGraphsCountTo.Text = Properties.Settings.Default.graphsCountTo;

TextBoxConnectionPossibilityFrom.Text = Properties.Settings.Default.connectionPossibilityFrom;

TextBoxConnectionPossibilityTo.Text = Properties.Settings.Default.connectionPossibilityTo;

TextBoxDegreeExpectationFrom.Text = Properties.Settings.Default.degreeExpectationFrom;

TextBoxDegreeExpectationTo.Text = Properties.Settings.Default.degreeExpectationTo;

TextBoxArcsExpectationFrom.Text = Properties.Settings.Default.arcsExpectationFrom;

TextBoxArcsExpectationTo.Text = Properties.Settings.Default.arcsExpectationTo;

}

private void RememberProperties()

{

//Вкладка проведения эксперимента

Properties.Settings.Default.areaWidth = TextBoxAreaWidth.Text;

Properties.Settings.Default.areaHeight = TextBoxAreaHeight.Text;

Properties.Settings.Default.radius = TextBoxRadius.Text;

Properties.Settings.Default.nodesCount = TextBoxNodesCount.Text; Properties.Settings.Default.graphsCount = TextBoxGraphsCount.Text;

//Вкладка поиска экспериментов

Properties.Settings.Default.areaWidthFrom = TextBoxAreaWidthFrom.Text;

Properties.Settings.Default.areaWidthTo = TextBoxAreaWidthTo.Text;

Properties.Settings.Default.areaHeightFrom = TextBoxAreaHeightFrom.Text;

Properties.Settings.Default.areaHeightTo = TextBoxAreaHeightTo.Text;

Properties.Settings.Default.radiusFrom = TextBoxRadiusFrom.Text;

Properties.Settings.Default.radiusTo = TextBoxRadiusTo.Text;

Properties.Settings.Default.nodesCountFrom = TextBoxNodesCountFrom.Text;

Properties.Settings.Default.nodesCountTo = TextBoxNodesCountTo.Text;

Properties.Settings.Default.graphsCountFrom = TextBoxGraphsCountFrom.Text;

Properties.Settings.Default.graphsCountTo = TextBoxGraphsCountTo.Text;

Properties.Settings.Default.connectionPossibilityFrom = TextBoxConnectionPossibilityFrom.Text;

Properties.Settings.Default.connectionPossibilityTo = TextBoxConnectionPossibilityTo.Text;

Properties.Settings.Default.degreeExpectationFrom = TextBoxDegreeExpectationFrom.Text;

Properties.Settings.Default.degreeExpectationTo = TextBoxDegreeExpectationTo.Text;

Properties.Settings.Default.arcsExpectationFrom = TextBoxArcsExpectationFrom.Text;

Properties.Settings.Default.arcsExpectationTo = TextBoxArcsExpectationTo.Text;

Properties.Settings.Default.Save();

}

private void ShowResults()

{

DataGridSearchResults.ItemsSource = \_lastSelection.ExperimentSelection;

DataGridSearchResults.Columns[0].Header = "Номер";

DataGridSearchResults.Columns[1].Header = "Форма";

DataGridSearchResults.Columns[2].Header = "Высота";

DataGridSearchResults.Columns[3].Header = "Ширина";

DataGridSearchResults.Columns[4].Header = "Радиус";

DataGridSearchResults.Columns[5].Header = "Кол-во графов";

DataGridSearchResults.Columns[6].Header = "Кол-во вершин";

DataGridSearchResults.Columns[7].Header = "Кол-во рёбер";

DataGridSearchResults.Columns[8].Header = "Кол-во связных графов";

DataGridSearchResults.Columns[9].Header = "Сумма степеней";

DataGridSearchResults.Columns[10].Header = "Вероятность связности";

DataGridSearchResults.Columns[11].Header = "Мат. ожидание степени вершины";

DataGridSearchResults.Columns[12].Header = "Мат. ожидание кол-ва рёбер";

DataGridSearchResults.Columns[13].Visibility = Visibility.Collapsed;

DataGridSearchResults.Columns[14].Visibility = Visibility.Collapsed; DataGridSearchResults.Visibility = Visibility.Visible;

}

private void Window\_Closing(object sender, EventArgs e)

{

RememberProperties();

}

private void MakeReport\_Click(object sender, RoutedEventArgs e)

{

if (((Button) sender).Name == $"{ButtonExperimentMakeReport.Name}")

{

\_lastExperiment.MakeReport();

ButtonExperimentViewReport.IsEnabled = true;

} else

{

\_lastSelection.MakeReport();

ButtonSelectionViewReport.IsEnabled = true;

}

}

private void ViewReport\_Click(object sender, RoutedEventArgs e)

{

Process.Start(((Button) sender).Name == $"{ButtonExperimentViewReport.Name}"

? $"{App.DataBaseControl.GetStringById(17)}{\_lastExperiment.Id}.pdf" : $"{App.DataBaseControl.GetStringById(18)}.pdf");

}

/\*

private static void CleanTables()

{

App.DataBaseControl.CleanTable("EXPERIMENTS");

App.DataBaseControl.CleanTable("POSSIBILITY\_DISTRIBUTION");

App.DataBaseControl.CleanTable("DYNAMIC");

}

\*/

}

}

using System; using System.Collections.Generic; using System.Linq;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public class Node

{

public readonly float Radius; public readonly float X; public readonly float Y; public readonly List<Node> Arclist;

public Node(int areaShape, float areaHeight, float areaWidth, float radius,ref List<Node> nodes, Random random)

{

Radius = radius;

do

{

1. = (float)(-0.5 + random.NextDouble()) \* 2 \* areaWidth;
2. = (float)(-0.5 + random.NextDouble()) \* 2 \* areaHeight;

}

while (areaShape == 1 && X \* X / Math.Pow(areaWidth, 2) + Y \* Y / Math.Pow(areaHeight, 2) > 1);

Arclist = new List<Node>();

foreach (var item in nodes.Where(item => IsUnited(item, this)))

{

item.Arclist.Add(this);

Arclist.Add(item);

}

}

static bool IsUnited(Node node1, Node node2)

{

var distance = Math.Sqrt((node2.X - node1.X) \* (node2.X - node1.X) + (node2.Y - node1.Y) \*

(node2.Y - node1.Y)); return node1.Radius >= distance || node2.Radius >= distance;

}

}

}

using System; using System.Collections.Generic; using System.IO; using System.Linq; using System.Windows; using System.Windows.Controls; using System.Windows.Media; using System.Windows.Media.Imaging; using System.Windows.Shapes;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public static class Painter

{

public static void PaintGraph(Graph graph, Panel canvas)

{

canvas.Children.Clear();

canvas.Width = graph.AreaWidth \* 2; canvas.Height = graph.AreaHeight \* 2;

foreach (var item in graph.Nodes)

{

var point = new Ellipse

{

Width = item.Radius \* 2,

Height = item.Radius \* 2,

Fill = new SolidColorBrush(Color.FromArgb((byte)(50 / Math.Sqrt(graph.Nodes.Count)), 0, 0, 0))

};

Canvas.SetLeft(point, graph.AreaWidth + item.X - item.Radius); Canvas.SetTop(point, graph.AreaHeight + item.Y - item.Radius); canvas.Children.Add(point);

point = new Ellipse { Width = Math.Max(graph.AreaHeight, graph.AreaWidth) / 50 }; point.Height = point.Width; point.Fill = Brushes.Black;

Canvas.SetLeft(point, graph.AreaWidth + item.X - point.Width / 2); Canvas.SetTop(point, graph.AreaHeight + item.Y - point.Width / 2); canvas.Children.Add(point);

foreach (var subItem in item.Arclist)

{

var line = new Line

{

Opacity = 0.2,

Stroke = Brushes.Black,

StrokeThickness = point.Width / 10,

X1 = item.X + graph.AreaWidth,

Y1 = item.Y + graph.AreaHeight,

X2 = subItem.X + graph.AreaWidth,

Y2 = subItem.Y + graph.AreaHeight

};

canvas.Children.Add(line);

}

}

}

private static void LabelingPlot(Panel canvas, float content, float oX, float oY, double fontSize, double width, HorizontalAlignment alignment)

{

var label = new Label

{

Content = content.ToString("0.##"),

HorizontalContentAlignment = alignment,

FontSize = fontSize,

Padding = new Thickness(0),

Margin = new Thickness(0),

FontFamily = new FontFamily("Times New Roman"),

Width = width

};

canvas.Children.Add(label); Canvas.SetLeft(label, oX);

Canvas.SetTop(label, oY);

}

public static void PaintPlot(IReadOnlyList<float> additional, string fileName)

{

var width = additional.Count - 1; var height = width/7.0;

var maxData = additional.Max() != 0.0 ? additional.Max() : 1; var heightScale = height / maxData; var canvas = new Canvas

{

Width = width,

Height = height

};

var pc = new PointCollection(); for (var i = 0; i < additional.Count; i++)

{

pc.Add(new Point(i, height - additional[i] \* heightScale)); if (i % Math.Max(1, (additional.Count / 25)) == 0)

LabelingPlot(canvas, i, i - (float)height / 6, (float)(canvas.Height \* 1.03), canvas.Height / 9, height / 3, HorizontalAlignment.Center);

}

for (var i = 0; i <= 6; i++)

{

canvas.Children.Add(new Line

{

Y1 = height - maxData \* i / 6 \* heightScale,

Y2 = height - maxData \* i / 6 \* heightScale,

X2 = width,

Stroke = Brushes.Gray,

StrokeThickness = height / 200

}); if (i == 0) continue; var q = maxData\*i/10; var q1 = q + q\*0.66666;

LabelingPlot(canvas, (float) q1, (float) (-height\*1.05),

(float) (height - maxData\*i/6\*heightScale - height / 18), height/9, height,

HorizontalAlignment.Right);

}

var line = new Polyline

{

Points = pc,

Stroke = Brushes.Black,

StrokeThickness = height/75

};

canvas.Children.Add(line); canvas.Children.Add(new Line

{

Y2 = height,

Stroke = Brushes.Gray,

StrokeThickness = height / 200

});

canvas.Children.Add(new Line

{

X1 = width,

X2 = width,

Y2 = height,

Stroke = Brushes.Gray,

StrokeThickness = height / 200

});

SaveCanvasAsAnImage(canvas, fileName);

}

private static void SaveCanvasAsAnImage(FrameworkElement canvas, string fileName)

{

var scale = 5000/ canvas.Width; var renderWidth = canvas.Width; var renderHeight = canvas.Height;

canvas.Margin = new Thickness(renderWidth \* 0.07, renderHeight \* 0.08, 0.0, 0.0);

var tempCanvas = new Canvas

{

Children = {canvas},

Background = Brushes.White,

RenderTransform = new ScaleTransform(scale, scale),

RenderTransformOrigin = new Point(0, 0)

};

tempCanvas.Measure(new Size(renderWidth \* 1.1, renderHeight \* 1.3)); tempCanvas.Arrange(new Rect(new Size(renderWidth \* 1.1, renderHeight \* 1.3))); var width = (int)(tempCanvas.RenderSize.Width \* scale); var height = (int)(tempCanvas.RenderSize.Height \* scale);

var rtb = new RenderTargetBitmap(width, height, 96d, 96d, PixelFormats.Default); rtb.Render(tempCanvas);

var crop = new CroppedBitmap(rtb, new Int32Rect(0, 0, width, height));

BitmapEncoder pngEncoder = new PngBitmapEncoder(); pngEncoder.Frames.Add(BitmapFrame.Create(crop));

canvas.RenderTransform = null;

using (var fs = File.OpenWrite($"{fileName}.png"))

{

pngEncoder.Save(fs);

}

}

}

}

using System.Text; using System.IO; using iTextSharp.text; using iTextSharp.text.pdf; using iTextSharp.tool.xml; using iTextSharp.tool.xml.html; using iTextSharp.tool.xml.parser; using iTextSharp.tool.xml.pipeline.css; using iTextSharp.tool.xml.pipeline.end; using iTextSharp.tool.xml.pipeline.html;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public static class Report

{

public static void GeneratePdf(string template, string fileName)

{

var document = new Document(); var writer = PdfWriter.GetInstance(document, new FileStream($"{fileName}.pdf", FileMode.Create)); document.Open(); var htmlContext = new HtmlPipelineContext(null); htmlContext.SetTagFactory(Tags.GetHtmlTagProcessorFactory());

var cssResolver = XMLWorkerHelper.GetInstance().GetDefaultCssResolver(true); IPipeline pipeline = new CssResolverPipeline(cssResolver, new HtmlPipeline(htmlContext, new PdfWriterPipeline(document, writer))); var worker = new XMLWorker(pipeline, true); var parser = new XMLParser(true, worker, Encoding.Unicode); var byteArray = Encoding.UTF8.GetBytes(template); var stream = new MemoryStream(byteArray); parser.Parse(stream); document.Close();

}

}

} using System; using System.Collections.Generic; using System.Globalization; using System.Linq;

namespace Ad\_Hoc\_v.\_4.\_0.\_1

{

public class Selection

{

// По области private readonly int \_areaShapeFrom; private readonly int \_areaShapeTo; private readonly float \_areaHeightFrom; private readonly float \_areaHeightTo; private readonly float \_areaWidthFrom; private readonly float \_areaWidthTo;

// По параметрам эксперимента private readonly float \_radiusFrom; private readonly float \_radiusTo; private readonly int \_nodesCountFrom; private readonly int \_nodesCountTo; private readonly int \_graphsCountFrom; private readonly int \_graphsCountTo;

// По результатам эксперимента private readonly float \_connectionPossibilityFrom; private readonly float \_connectionPossibilityTo; private readonly float \_degreeExpectationFrom; private readonly float \_degreeExpectationTo; private readonly float \_arcsExpectationFrom; private readonly float \_arcsExpectationTo;

public List<Experiment> ExperimentSelection { get; }

// Средние значения private float \_connectedPossibilityAverage; private float \_degreeExpectationAverage; private float \_arcsExpectationAverage; private readonly List<float> \_possibilityDistributionAverage;

private Selection(int areaShapeFrom, int areaShapeTo, float areaHeightFrom, float areaHeightTo, float areaWidthFrom, float areaWidthTo, float radiusFrom, float radiusTo, int nodesCountFrom, int nodesCountTo, int graphsCountFrom, int graphsCountTo, float connectionPossibilityFrom, float connectionPossibilityTo, float degreeExpectationFrom, float degreeExpectationTo, float arcsExpectationFrom, float arcsExpectationTo, List<Experiment> experimentSelection)

{

\_areaShapeFrom = areaShapeFrom;

\_areaShapeTo = areaShapeTo;

// По области

\_areaHeightFrom = areaHeightFrom;

\_areaHeightTo = areaHeightTo;

\_areaWidthFrom = areaWidthFrom;

\_areaWidthTo = areaWidthTo;

// По параметрам эксперимента

\_radiusFrom = radiusFrom;

\_radiusTo = radiusTo;

\_nodesCountFrom = nodesCountFrom;

\_nodesCountTo = nodesCountTo;

\_graphsCountFrom = graphsCountFrom;

\_graphsCountTo = graphsCountTo;

// По результатам эксперимента

\_connectionPossibilityFrom = connectionPossibilityFrom;

\_connectionPossibilityTo = connectionPossibilityTo;

\_degreeExpectationFrom = degreeExpectationFrom;

\_degreeExpectationTo = degreeExpectationTo;

\_arcsExpectationFrom = arcsExpectationFrom;

\_arcsExpectationTo = arcsExpectationTo;

ExperimentSelection = experimentSelection;

\_possibilityDistributionAverage = new List<float>();

СonsiderAverages();

}

public static Selection FindExperiments(int areaShape, float areaHeightFrom, float areaHeightTo, float areaWidthFrom, float areaWidthTo, float radiusFrom, float radiusTo, int nodesCountFrom, int nodesCountTo, int graphsCountFrom, int graphsCountTo, float connectionPossibilityFrom,

float connectionPossibilityTo, float degreeExpectationFrom, float degreeExpectationTo, float arcsExpectationFrom, float arcsExpectationTo, string operationCode)

{

List<Experiment> resultOfSearch;

Selection result;

switch (operationCode)

{ case "100":

resultOfSearch = App.DataBaseControl.FindExperiments(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, 0, float.MaxValue,

0, int.MaxValue, 0, int.MaxValue, 0, float.MaxValue, 0, float.MaxValue, 0, float.MaxValue);

result = new Selection(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, 0, float.MaxValue,

0, int.MaxValue, 0, int.MaxValue, 0, float.MaxValue, 0, float.MaxValue, 0, float.MaxValue, resultOfSearch);

break; case "010":

resultOfSearch = App.DataBaseControl.FindExperiments(0, 1, 0, float.MaxValue, 0, float.MaxValue, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, 0, float.MaxValue, 0, float.MaxValue, 0, float.MaxValue);

result = new Selection(0, 1, 0, float.MaxValue, 0, float.MaxValue, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, 0, float.MaxValue,

0, float.MaxValue, 0, float.MaxValue, resultOfSearch); break; case "001":

resultOfSearch = App.DataBaseControl.FindExperiments(0, 1, 0, float.MaxValue, 0, float.MaxValue, 0, float.MaxValue,

0, int.MaxValue, graphsCountFrom, graphsCountTo, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo);

result = new Selection(0, 1, 0, float.MaxValue, 0, float.MaxValue, 0, float.MaxValue, 0, int.MaxValue, graphsCountFrom, graphsCountTo, connectionPossibilityFrom, connectionPossibilityTo,

degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo, resultOfSearch);

break; case "110":

resultOfSearch = App.DataBaseControl.FindExperiments(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, radiusFrom, radiusTo,

nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, 0, float.MaxValue, 0, float.MaxValue, 0, float.MaxValue);

result = new Selection(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, 0, float.MaxValue,

0, float.MaxValue, 0, float.MaxValue, resultOfSearch); break; case "101":

resultOfSearch = App.DataBaseControl.FindExperiments(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, 0, float.MaxValue,

0, int.MaxValue, 0, int.MaxValue, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo);

result = new Selection(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, 0, float.MaxValue,

0, int.MaxValue, 0, int.MaxValue, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo, resultOfSearch);

break; case "011":

resultOfSearch = App.DataBaseControl.FindExperiments(0, 1, 0, float.MaxValue, 0, float.MaxValue, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo);

result = new Selection(0, 1, 0, float.MaxValue, 0, float.MaxValue, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo, resultOfSearch);

break; case "111":

resultOfSearch = App.DataBaseControl.FindExperiments(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo);

result = new Selection(areaShape, areaShape, areaHeightFrom, areaHeightTo, areaWidthFrom, areaWidthTo, radiusFrom, radiusTo, nodesCountFrom, nodesCountTo, graphsCountFrom, graphsCountTo, connectionPossibilityFrom, connectionPossibilityTo, degreeExpectationFrom, degreeExpectationTo, arcsExpectationFrom, arcsExpectationTo, resultOfSearch);

break; default:

return null;

}

return result;

}

private void СonsiderAverages()

{

var connectedPossibilityAverage = 0.0; var degreeExpectationAverage = 0.0; var arcsExpectationAverage = 0.0; var possibilityDistributionAverage = new List<int>();

var experimentCount = ExperimentSelection.Count;

foreach (var experiment in ExperimentSelection)

{

connectedPossibilityAverage += experiment.ConnectedPossibility; degreeExpectationAverage += experiment.DegreeExpectation; arcsExpectationAverage += experiment.ArcsExpectation;

possibilityDistributionAverage = FoldLists(possibilityDistributionAverage, experiment.PossibilityDistribution);

}

\_connectedPossibilityAverage = (float) connectedPossibilityAverage/experimentCount;

\_degreeExpectationAverage = (float) degreeExpectationAverage/experimentCount;

\_arcsExpectationAverage = (float) arcsExpectationAverage/experimentCount;

foreach (var node in possibilityDistributionAverage)

{

\_possibilityDistributionAverage.Add((float) node / experimentCount);

}

}

private static List<int> FoldLists(List<int> list1, List<int> list2)

{

var firstMax = (list1.Count > list2.Count);

var minLength = firstMax ? list2.Count : list1.Count;

var result = (firstMax ? list1 : list2);

for (var i = 0; i < result.Count; i++)

{

if (i >= minLength) break; result[i] = list1[i] + list2[i];

}

return result;

}

private Dictionary<string, string> GetReplaceItemForReport()

{

var areaShapeFrom = \_areaShapeFrom == 0 ? "Овальная" : "Квадратная"; var areaShapeTo = \_areaShapeTo == 0 ? "Овальная" : "Квадратная";

return new Dictionary<string, string>()

{

{"[experimentsCount]", ExperimentSelection.Count.ToString()},

{"[areaShapeFrom]", areaShapeFrom},

{"[areaShapeTo]", areaShapeTo},

{"[graphsCountFrom]", \_graphsCountFrom.ToString()},

{"[graphsCountTo]", \_graphsCountTo.ToString()},

{"[areaHeightFrom]", \_areaHeightFrom.ToString(CultureInfo.InvariantCulture)},

{"[areaHeightTo]", \_areaHeightTo.ToString(CultureInfo.InvariantCulture)},

{"[areaWidthFrom]", \_areaWidthFrom.ToString(CultureInfo.InvariantCulture)},

{"[areaWidthTo]", \_areaWidthTo.ToString(CultureInfo.InvariantCulture)},

{"[nodesCountFrom]", \_nodesCountFrom.ToString()},

{"[nodesCountTo]", \_nodesCountTo.ToString()},

{"[radiusFrom]", \_radiusFrom.ToString(CultureInfo.InvariantCulture)},

{"[radiusTo]", \_radiusTo.ToString(CultureInfo.InvariantCulture)},

{"[connectedPossibilityFrom]",

\_connectionPossibilityFrom.ToString(CultureInfo.InvariantCulture)},

{"[connectedPossibilityTo]", \_connectionPossibilityTo.ToString(CultureInfo.InvariantCulture)},

{"[degreeExpectationFrom]", \_degreeExpectationFrom.ToString(CultureInfo.InvariantCulture)},

{"[degreeExpectationTo]", \_degreeExpectationTo.ToString(CultureInfo.InvariantCulture)},

{"[arcsExpectationFrom]", \_arcsExpectationFrom.ToString(CultureInfo.InvariantCulture)},

{"[arcsExpectationTo]", \_arcsExpectationTo.ToString(CultureInfo.InvariantCulture)},

{"[connectionPossibilityAverage]",

\_connectedPossibilityAverage.ToString(CultureInfo.InvariantCulture)},

{"[arcsExpectationAverage]", \_arcsExpectationAverage.ToString(CultureInfo.InvariantCulture)}, {"[degreeExpectationAverage]",

\_degreeExpectationAverage.ToString(CultureInfo.InvariantCulture)},

{$"[{App.DataBaseControl.GetStringById(16)}]", $"{App.DataBaseControl.GetStringById(16)}.png"}

};

}

private void MakePlots()

{

Painter.PaintPlot(\_possibilityDistributionAverage, App.DataBaseControl.GetStringById(16));

}

public void MakeReport()

{

// Отрисовка графиков

MakePlots();

// Тянем шаблон из базы данных

var template = App.DataBaseControl.GetTemplateById(6).Replace("[reportDate]", DateTime.Today.ToShortDateString());

var insertString = "";

// Генерация строки, содержащей метки для вставки необходимого кол-ва записей в таблицу for (var i = 0; i < ExperimentSelection.Count; i++)

{

insertString += $"[selectionRow{i}]";

}

// Вставляем эту строку в нужное место шаблона template = template.Replace("[selectionRow]", insertString);

// Заменим каждую метку на заполненный шаблон строки for (var i = 0; i < ExperimentSelection.Count; i++)

{

// Тянем шаблон строки из базы var rowTemplate = App.DataBaseControl.GetTemplateById(7);

// Заменяем метки на значения

rowTemplate = ExperimentSelection[i].GetReplaceItemsForReport().Aggregate(rowTemplate,

(current, replaceItem) => current.Replace(replaceItem.Key, replaceItem.Value));

// Заменяем метку в основном шаблоне на нашу готовую строку

template = template.Replace($"[selectionRow{i}]", rowTemplate);

}

// Меняем оставшиеся метки в шаблоне template = GetReplaceItemForReport().Aggregate(template, (current, replaceItem) => current.Replace(replaceItem.Key, replaceItem.Value));

Report.GeneratePdf(template, $"{App.DataBaseControl.GetStringById(18)}");

}

}

}

ПРИЛОЖЕНИЕ В

Разметка окна

<Window x:Class="Ad\_Hoc\_v.\_4.\_0.\_1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d"

Title="MainWindow" Height="600" Width="1225" Closing="Window\_Closing"

WindowStartupLocation="CenterScreen">

<Grid>

<TabControl>

<TabItem Header="Поиск эксперимента">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="1.5\*"/>

<RowDefinition Height="Auto"/> <RowDefinition Height="3\*"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="1.2\*"/>

</Grid.ColumnDefinitions>

<CheckBox x:Name="CheckBoxFieldParametrs" IsChecked="True"

Click="CheckBoxSearchParametrs\_Click">Параметры расположения сети</CheckBox>

<Grid Grid.Row="1" Grid.Column="0">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Grid.Column="1">От</Label>

<Label Grid.Row="0" Grid.Column="2">До</Label>

<Label Grid.Row="1" Grid.Column="0">Высота области</Label>

<TextBox x:Name="TextBoxAreaHeightFrom" Grid.Row="1" Grid.Column="1"/> <TextBox x:Name="TextBoxAreaHeightTo" Grid.Row="1" Grid.Column="2"/> <Label Grid.Row="2" Grid.Column="0">Ширина области</Label>

<TextBox x:Name="TextBoxAreaWidthFrom" Grid.Row="2" Grid.Column="1"/> <TextBox x:Name="TextBoxAreaWidthTo" Grid.Row="2" Grid.Column="2"/>

<Label Grid.Row="3" Grid.Column="0">Форма области</Label>

<StackPanel Orientation="Vertical" Grid.Column="1" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.ColumnSpan="2" Grid.Row="3">

<RadioButton x:Name="RadioButtonRoundSearch" IsChecked="True">Овальная</RadioButton>

<RadioButton>Прямоугольная</RadioButton>

</StackPanel>

</Grid>

<Border x:Name="BorderFieldParametrs" Visibility="Collapsed" Grid.Row="1" Grid.Column="0"

Background="#7FBFBFBF"/>

<CheckBox x:Name="CheckBoxExperimentParametrs" Grid.Row="0" Grid.Column="1" IsChecked="True" Click="CheckBoxSearchParametrs\_Click">Основные свойства эксперимента</CheckBox>

<Grid Grid.Column="1" Grid.Row="1">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Grid.Column="1">От</Label>

<Label Grid.Row="0" Grid.Column="2">До</Label>

<Label Grid.Row="1" Grid.Column="0">Количество графов</Label>

<TextBox x:Name="TextBoxGraphsCountFrom" Grid.Row="1" Grid.Column="1"/> <TextBox x:Name="TextBoxGraphsCountTo" Grid.Row="1" Grid.Column="2"/>

<Label Grid.Row="2" Grid.Column="0">Количество вершин</Label>

<TextBox x:Name="TextBoxNodesCountFrom" Grid.Row="2" Grid.Column="1"/> <TextBox x:Name="TextBoxNodesCountTo" Grid.Row="2" Grid.Column="2"/>

<Label Grid.Row="3" Grid.Column="0">Радиус вершины</Label>

<TextBox x:Name="TextBoxRadiusFrom" Grid.Row="3" Grid.Column="1"/> <TextBox x:Name="TextBoxRadiusTo" Grid.Row="3" Grid.Column="2"/>

</Grid>

<Border x:Name="BorderExperimentParametrs" Visibility="Collapsed" Grid.Row="1"

Grid.Column="1" Background="#7FBFBFBF"/>

<CheckBox x:Name="CheckBoxExperimentResultsParametrs" Grid.Row="0" Grid.Column="2" IsChecked="True" Click="CheckBoxSearchParametrs\_Click">Результаты проведения эксперимента</CheckBox>

<Grid Grid.Column="2" Grid.Row="1">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Grid.Column="1">От</Label>

<Label Grid.Row="0" Grid.Column="2">До</Label>

<Label Grid.Row="1" Grid.Column="0">Вероятность связности сети</Label>

<TextBox x:Name="TextBoxConnectionPossibilityFrom" Grid.Row="1" Grid.Column="1"/> <TextBox x:Name="TextBoxConnectionPossibilityTo" Grid.Row="1" Grid.Column="2"/>

<Label Grid.Row="2" Grid.Column="0">Мат. ожидание степени узла сети</Label>

<TextBox x:Name="TextBoxDegreeExpectationFrom" Grid.Row="2" Grid.Column="1"/> <TextBox x:Name="TextBoxDegreeExpectationTo" Grid.Row="2" Grid.Column="2"/>

<Label Grid.Row="3" Grid.Column="0">Мат. ожидание количества связей в сети</Label>

<TextBox x:Name="TextBoxArcsExpectationFrom" Grid.Row="3" Grid.Column="1"/> <TextBox x:Name="TextBoxArcsExpectationTo" Grid.Row="3" Grid.Column="2"/>

</Grid>

<Border x:Name="BorderExperimentResultsParametrs" Visibility="Collapsed" Grid.Row="1"

Grid.Column="2" Background="#7FBFBFBF"/>

<Button Grid.Row="2" Grid.Column="1" Click="SeachExperiment\_Click">Найти эксперименты</Button>

<DataGrid x:Name="DataGridSearchResults" Grid.Row="3" Grid.ColumnSpan="3"

Grid.Column="0"

HorizontalAlignment ="Center" Visibility="Hidden" FontSize="11"/>

<Button x:Name="ButtonSelectionMakeReport" IsEnabled="False" Grid.Row="4" Grid.Column="1"

Click="MakeReport\_Click">

Сформировать отчёт</Button>

<Button x:Name="ButtonSelectionViewReport" IsEnabled="False" Grid.Row="4"

Grid.Column="2"

Click="ViewReport\_Click">Просмотреть отчёт</Button>

</Grid>

</TabItem>

<TabItem Header="Проведение эксперимента">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.45\*"/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid Margin="20" Background="#4CFFFFFF">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label>Форма области</Label>

<StackPanel Orientation="Vertical" Grid.Row="0" Grid.Column="1">

<RadioButton x:Name="RadioButtonRoundMake" IsChecked="True">Овальная</RadioButton>

<RadioButton>Прямоугольная</RadioButton>

</StackPanel>

<Label Grid.Row="1" Grid.Column="0">Высота области</Label>

<TextBox x:Name="TextBoxAreaHeight" Grid.Column="1" TextWrapping="Wrap" Grid.Row="1"/>

<Label Grid.Row="2" Grid.Column="0">Ширина области</Label>

<TextBox x:Name="TextBoxAreaWidth" Grid.Column="1" TextWrapping="Wrap" Grid.Row="2"/>

<Label Grid.Row="3" Grid.Column="0">Радиус вершины</Label>

<TextBox x:Name="TextBoxRadius" Grid.Column="1" TextWrapping="Wrap" Grid.Row="3"/>

<Label Grid.Row="4" Grid.Column="0">Количество вершин</Label>

<TextBox x:Name="TextBoxNodesCount" Grid.Column="1" TextWrapping="Wrap" Grid.Row="4"/>

<Label Grid.Row="5" Grid.Column="0">Количество графов</Label>

<TextBox x:Name="TextBoxGraphsCount" Grid.Column="1" TextWrapping="Wrap" Grid.Row="5"/>

<Button Grid.ColumnSpan="2" Grid.Column="0" Grid.Row="6" Click="MakeExperiment\_Click">Провести эксперимент</Button>

<Label Grid.ColumnSpan="2" Grid.Column="0" Grid.Row="7">Вероятность связности:</Label>

<TextBlock x:Name="TextBlockConnectionPossibility" Grid.Row="8" Grid.ColumnSpan="2"

Grid.Column="0" />

<Label Grid.ColumnSpan="2" Grid.Column="0" Grid.Row="9">Математическое ожидание степени вершны:</Label>

<TextBlock x:Name="TextBlockDegreeExpectation" Grid.ColumnSpan="2"

Grid.Column="0" Grid.Row="10"/>

<Label Grid.ColumnSpan="2" Grid.Column="0" Grid.Row="11">Математическое ожидание количества рёбер:</Label>

<TextBlock x:Name="TextBlockArcsExpectation" Grid.ColumnSpan="2" Grid.Column="0" Grid.Row="12"/>

<Button IsEnabled="False" x:Name="ButtonExperimentMakeReport" Grid.Row="13" Grid.ColumnSpan="2" Grid.Column="0"

Margin="25, 10, 25, 10" Click="MakeReport\_Click">Сформировать отчёт</Button>

<Button IsEnabled="False" x:Name="ButtonExperimentViewReport" Grid.Row="14" Grid.ColumnSpan="2" Grid.Column="0"

Margin="25, 10, 25, 10" Click="ViewReport\_Click">Просмотреть отчёт</Button>

</Grid>

<Viewbox Panel.ZIndex="-1" RenderTransformOrigin="0.5,0.5" Grid.Column="1" Grid.Row="0">

<Canvas x:Name="CanvasGraphDemo" />

</Viewbox>

</Grid>

</TabItem>

</TabControl>

</Grid>

</Window>