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. ф.-м. н., профессор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Миков А.И.

(подпись) (инициалы, фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ ДЛЯ ВИРТУАЛЬНОЙ**

**ЛАБОРАТОРИИ ИССЛЕДОВАНИЯ МОБИЛЬНЫХ AD HOC СЕТЕЙ**

Работу выполнил П. А. Беловол

(подпись, дата) (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики Направление 02.03.02 – «Фундаментальные информатика и информационные технологии»

Научный руководитель

к. ф.-м. н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н Лапина

(подпись, дата) (инициалы, фамилия)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.С Фисун

(подпись, дата) (инициалы, фамилия)

Краснодар 2016

РЕФЕРАТ

Выпускная квалификационная работа 101 страниц, 32 рисунков, 11 источников, 2 приложения.

AD HOC, БЕСПРОВОДНЫЕ СЕТИ, ГЕОМЕТРИЧЕСКИЙ ГРАФ, МОДЕЛИРОВАНИЕ

Целью работы является разработка программных модулей для виртуальной лаборатории исследования мобильных ad hoc сетей. С помощью разработанных модулей можно выполнять построение моделей ad hoc сетей и производить анализ изменения их характеристик в зависимости от ряда параметров.

Разработано клиент-серверное приложение, предназначенное для исследования моделей мобильных беспроводных сетей. В серверной части приложения используются разработанные программные модули. Клиентская часть приложения предоставляет удобный пользовательский интерфейс для коммуникации с сервером.

Результат работы – программные модули, с помощью которых можно выполнять построение и исследование моделей ad hoc сетей, т.е. моделировать ad hoc сети, производить анализ параметров связности, анализ расположения компонент ad hoc сетей, выполнять построение графиков изменения характеристик сети от ряда параметров. В разработанном клиент-серверном приложении клиенты имеют доступ ко всем вышеперечисленным функциям.

Таким образом, с помощью разработанных программных модулей можно создавать, обрабатывать и анализировать модели ad hoc сетей, концептуально представимых в виде геометрического графа.

В работе использовался язык программирования С# в среде разработки Microsoft Visual Studio 2015.

СОДЕРЖАНИЕ

Введение ...................................................................................................................... 4

1. Беспроводные сети ................................................................................................ 5
   1. Основные сведения и определения .............................................................. 5
   2. Представление сети с помощью геометрического графа .......................... 7
2. Программные модули ......................................................................................... 10
   1. Основные определения ............................................................................... 10
   2. Модуль Graph .............................................................................................. 11
   3. Модуль Algorithms ...................................................................................... 12
   4. Модуль Test .................................................................................................. 16
3. Клиент-сервер ...................................................................................................... 19
   1. Общие сведения ........................................................................................... 19
   2. Сервер TCP .................................................................................................. 22
   3. Клиент TCP .................................................................................................. 25
   4. Интерфейс клиента ..................................................................................... 27
   5. Интерфейс сервера ...................................................................................... 43

Заключение ................................................................................................................ 45

Список использованных источников ...................................................................... 46

Приложение А Графики зависимости математических

ожиданий величин.......................................................................... 47

Приложение Б Исходный код программы .............................................................. 55

ВВЕДЕНИЕ

Целью дипломной работы является разработка программных модулей для построения и анализа моделей ad hoc сетей.

Разработанные программные модули должны реализовывать следующие задачи: моделировать ad hoc сети, производить анализ параметров связности, анализ расположения компонент ad hoc сетей, выполнять построение графиков изменения характеристик сети от ряда параметров.

Клиент-серверное приложение предоставит возможность клиентам удаленно выполнять операции для построения и исследования моделей мобильных беспроводных сетей. Решит задачу получения вычислений на маломощных ПК, где вычисления при объемном количестве итераций требуют значительных временных затрат. Все вычисления будут выполняться на серверной части с использованием разработанных модулей. Клиенты имеют удобный интерфейс для доступа к функциям сервера.

Моделирование будет проводиться согласно имитационным методам, то есть необходимо строить модели, описывающие процессы так, как они проходили бы в действительности. Такой анализ поможет существенно сэкономить время развертывания данных сетей на физическом уровне, определить, в каком расположении сеть связна и имеет наилучшие характеристики.

# 1 Беспроводные сети

## 1.1 Основные сведения и определения

Ad hoc - это одноранговые беспроводные сети передачи данных с переменной топологией и отсутствием четкой инфраструктуры, где каждый узел может выполнять функции маршрутизатора и принимать участие в ретрансляции пакетов данных. Подобные сети могут применяться во время военных действий, в структурах МЧС, в системах транспорта и различных силовых структурах. Пример структуры Ad hoc-сети изображен на рисунке 1.

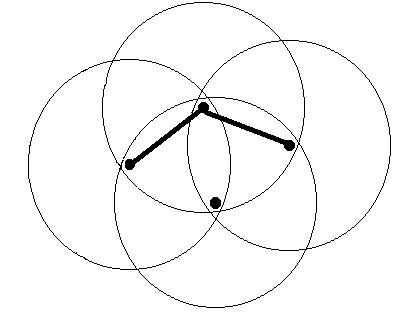


Рисунок 1 – Пример структуры ad hoc сети

Специфика сетей ad hoc состоит в том, что их топология постоянно изменяется из-за перемещения узлов сети в пространстве или изменения условий распространения радиосигнала. Помимо этого, для ad hoc сетей, как и для любых беспроводных систем, характерны ограниченные полоса пропускания и зона радиовидимости. В результате протоколы и технические решения, используемые в классических проводных сетях передачи данных, например, централизованная маршрутизация с иерархией заранее назначенных маршрутизаторов, в сетях ad hoc оказываются неэффективными и не обеспечивают нужную производительность.

Для успешного применения в ad hoc сетях протоколы маршрутизации должны обладать следующими качествами:

* Быть распределенными. Все узлы в сети должны быть способны осуществлять маршрутизацию и не иметь жестко закрепленных за собой функций.
* Обеспечивать надежную доставку пакетов в условиях постоянно изменяющейся топологии сети, когда использование классических механизмов гарантированной доставки, как, например, на транспортном уровне в протоколе ТСР, затруднено.
* Обеспечивать малое время построения маршрута в условиях постоянно изменяющейся топологии сети.
* Обладать механизмами оперативного обнаружения разрыва маршрута и его восстановления.
* Не допускать образования петель в маршрутах.
* Рассылать при функционировании как можно меньший объем служебной информации.
* Обладать высокой масштабируемостью, т.е. обеспечивать высокую производительность сети при различных ее размерах.
* Поддерживать QoS.

Можно выделить следующие основные преимущества беспроводных сетей:

* Низкая стоимость. Массовое производство электронных устройств значительно дешевле производства, установки и настройки многокомпонентного комплекса, решающего аналогичную задачу.
* Быстрота развертывания. Беспроводная сеть готова к работе практически сразу же после размещения и включения всех устройств.
* Высокая эффективность. Благодаря возможности одновременного применения большого количества устройств.

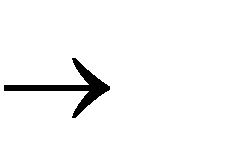
[Винокуров В.М., Пуговкин А.В. Маршрутизация в беспроводных

мобильных Ad hoc сетях. с. 207]

## 1.2 Представление сети с помощью геометрического графа

Компьютерные ad hoc сети – это беспроводные децентрализованные сети с *n* узлами, расположенными в некоторой ограниченной области *S*. Каждый узел характеризуется радиусом действия приемопередатчика сигналов *r*. Два узла могут обмениваться информацией, если расстояние между ними меньше *r*. Математической моделью такой сети является так называемый *геометрический граф*. Работоспособность и качественные характеристики ad hoc сети могут быть описаны через такие характеристикигеометрического графа как связность, количество компонент, длины маршрутов и прочее.

Граф, или неориентированный граф *G* — это упорядоченная пара *G = (V, E),* где *V* — это непустое множество вершин или узлов, а *E* — множество пар (в случае неориентированного графа — неупорядоченных) вершин, называемых рёбрами.

Представим компьютерную сеть соответствующим ей геометрическим *графом*  – набор *GG* = {*V*, *K*, *r*, *S*}, где *V* – множество вершин, *K* – отображение *K*: *V* *S*, *r* – радиус действия приемо-передатчиков, *S* – ориентированная ограниченная поверхность, на которой располагаются вершины.

Узлы сети – вершины графа в качестве атрибутов имеют координаты узла на заданной ограниченной поверхности S.Другим атрибутом является величина радиуса действия приемопередатчика, обеспечивающего прием сигналов от других узлов и передачу сигналов другим узлам. Таким образом, вершина находится в центре окружности некоторого радиуса. Если внутрь этой окружности попадает другая вершина, то связь между вершинами имеется. Такие вершины считаются смежными в геометрическом графе и соединяются ребром.

[Миков А.И. Графы и грамматики. c. 68]

Пример такого геометрического графа приведен на рисунке 2.

Рисунок 2 – Пример геометрического графа

Поверхность (область) *S* определяется рельефом местности, на которой располагается компьютерная сеть. В общем случае она не является выпуклой и отображает также и препятствия распространению радиосигналов микроволнового и/или оптического диапазона.

Геометрический граф *GG* = {*V*, *K*, *r*, *S*} порождает обыкновенный граф *G* = {*V*, *E*} следующим образом: вершины обыкновенного графа соответствуют вершинам геометрического графа; вершины обыкновенного графа *vi* и *v j* смежны, т.е.(*vi* ,*vj* )*E* если длина отрезка между точками *P*(*vi* ) и *P*(*v j* ) строго меньше величины *r* , и никакая часть этого отрезка не проходит «под» ориентированной поверхностью *S* (прямая видимость).

На рисунке 3 изображен несвязный геометрический граф, состоящий из двух компонент связности.

Рисунок 3 - Несвязный геометрический граф

Граф можно представить в виде списочной структуры, состоящей из списков двух типов: списка вершин и списков ребер. Элемент списка вершин содержит поле данных и два указателя. Один указатель связывает данный элемент с элементом другой вершины. Другой указатель связывает элемент списка вершин со списком ребер, связанных с данной вершиной. Для ориентированного графа используют список дуг, исходящих из вершины. Элемент списка дуг состоит только из двух указателей. Первый указатель используется для того, чтобы показать в какую вершину дуга входит, а второй для связи элементов в списке дуг вершины. На рисунке 4 изображено списковое представление графа. [Свами М., Тхуласираман К. Графы, сети и алгоритмы. c. 125]

graph

1

2

3

4

5

2

1

5

4

3

Рисунок 4 – Списковое представление графа

Числами 1..5 на рисунке 4 помечены дескрипторы вершин, также пронумерованы вершины соответствующего графа, нарисованного справа от представления. На рисунке графа пары взаимно обратных дуг изображены линиями без стрелок. В представлении дугам графа соответствуют дескрипторы дуг и толстые стрелки – ссылки на дескрипторы вершин.

2 Программные модули

## 2.1 Основные определения

Модуль — функционально законченный фрагмент [программы.](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) Во многих языках программирования оформляется в виде отдельного [файла](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB) с [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) или поименованной непрерывной её части. Некоторые языки поддерживают объединение модулей в [пакеты.](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))

Принцип модульности является средством упрощения задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков. При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

Программный код часто разбивается на несколько файлов, каждый из которых [компилируется](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) отдельно от остальных. Такая модульность программного кода позволяет значительно уменьшить время перекомпиляции при изменениях, вносимых лишь в небольшое количество исходных файлов, и упрощает групповую [разработку.](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) Также это возможность замены отдельных компонентов (таких как *jar*, *so* или *dll* библиотеки) конечного программного

продукта, без необходимости пересборки всего проекта (например, разработка [плагинов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD) к уже готовой программе).

Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и др. программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней.

В данной работе программные модули определены как классы. Имеется несколько файлов: *Graph.cs*, *Algorithms.cs*, *Test.cs*.

## 2.2 Модуль Graph

Модуль *Graph* представляет целевую структуру ad hoc сети в виде геометрического графа, и содержит вспомогательные методы для построения и анализа.

Методы, содержащиеся в модуле:

* Добавление узла в граф.
* Поиск узла в графе.
* Проверка узлов на смежность
* Получить узел по координатам x и y.
* Поиск компонент связности
* Сортировка компонент связности
* Поиск главной компоненты связности
* Количество узлов в главной компоненте связности
* Количество ребер в главной компоненте связности
* Расстояние ближайшей точки главной компоненты до границы области
* Печать графа
* Печать компонент связности
* Равномерное распределение узлов в ограниченной области
* Максимальная степень вершины главной компоненты
* Количество компонент связности

Graph содержит в себе вложенные классы *Node* – узел, *Arc* – ребро, данные классы имеют вспомогательный характер, содержат дополнительные поля и методы для доступа к этим полям. Для того чтобы использовать вышеперечисленные методы, содержащиеся в *Graph* необходимо создать новый экземпляр Graph, и через созданный экземпляр вызвать нужный метод.

## 2.3 Модуль Algorithms

Модуль *Algorithms* содержит алгоритмы для построения и анализа

выпуклой оболочки.

Методы, содержащиеся в модуле:

* Алгоритм для построения выпуклой оболочки *QuickHull*
* Площадь выпуклой оболочки
* Центр масс выпуклой оболочки
* Количество углов выпуклой оболочки
* Расстояние от центра тяжести многоугольника до центра области

Algorithms содержит вложенный класс *Shape* – фигура, данный класс имеет вспомогательный характер для определения выпуклой оболочки.

*QuickHull* - рекурсивный алгоритм для построения выпуклой оболочки.

Время работы:

* Лучший случай, все точки внутри оболочки: *O(n)*
* Худший случай, все точки на оболочке: *O(n2)*
* Среднее время: *O(nlogn)*

На первом шаге алгоритм выполняет создание четырехугольника, соединяющего крайние точки, как показано на рисунке 5. Только точки, лежащие вне его, могут лежать на оболочке и будут рассматриваться в дальнейшем. В Каждый лежащий вне четырехугольника участок будет рекурсивно обработан функцией *Quickhull*. На диаграмме ниже изображена обработка верхнего-правого угла.

На шаге *А* также находится точка *c* - наиболее удаленная от линии *(a,b).* Следующим шагом определяем два множества точек: справа или на линии *(a,c)* и справа или на *(c ,b),* как показано на рисунке 6. Для них вновь выполняется вызов функции *Quickhull*.

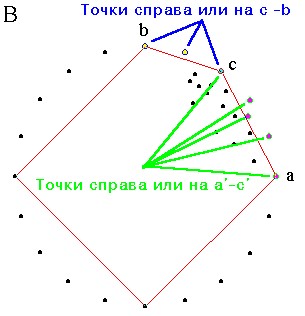
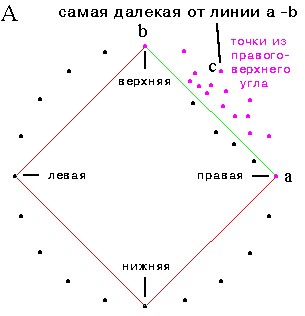


Рисунок 5 – 1 шаг алгоритма Рисунок 6 – 2 шаг алгоритма

Ниже изображена аналогичная обработка первого множества. Находится точка *c’, как* показано на рисунке 7, наиболее удаленная от линии *(a',b'),* определяются два новых множества точек: справа или на *(a',c')* и *(c',b').*

Получается *D* на рисунке 8, которое затем обрабатывается дальше.

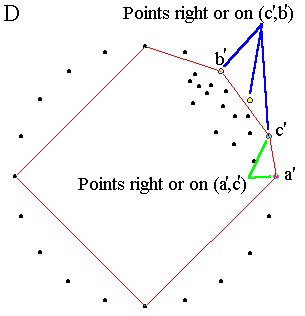
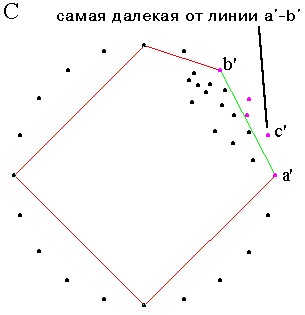


Рисунок 7 – 3 шаг алгоритма

Рисунок 8 – 4 шаг алгоритма

Алгоритм продолжает выполнять вызов *Quickhull* для этих двух множеств. На рисунках 9 и 10 показан результат вызова для первого множества. Если множество состоит только из двух точек, рекурсия останавливается, возвращая эти две точки как сторону выпуклой оболочки (рисунок 10, сторону изображает черная линия).

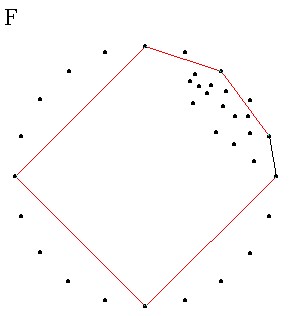
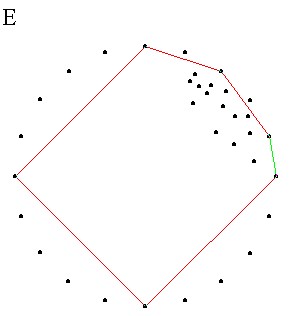


Рисунок 9 – 5 шаг алгоритма Рисунок 10 – 6 шаг алгоритма

Аналогичным способом обрабатываются верхний-левый, нижний-левый и нижний-правый углы, пока не получим полную выпуклую оболочку, как показано на рисунке 11. [Алгоритмы построения выпуклой оболочки. http://algolist.manual.ru/maths/geom/convhull/]

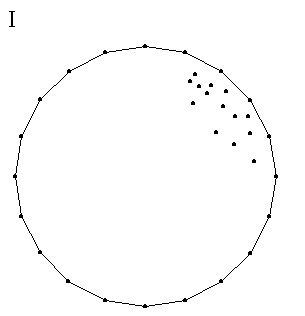


Рисунок 11 – Полная выпуклая оболочка

На случайном наборе точек этот алгоритм работает быстрее других алгоритмов. Наихудшее поведение алгоритм демонстрирует в случае, если заданные точки уже образуют выпуклый многоугольник, так как никакие точки в этом случае не отбрасываются. Тогда он выполняет *O(n2)* операций.

2.4 Модуль Test

*Test* – Модуль тестирования. Содержит методы, выполняющие вычисления для построения графиков зависимости. Использует вышеописанные модули Graph и Algorithms.

Графики зависимостей, которые можно построить с помощью данного модуля:

* Максимальная степень вершины главной компоненты от радиуса приема узла
* Расстояние от центра тяжести многоугольника до центра области от радиуса приема узла
* Количество углов в многоугольнике от радиуса приема узла
* Количество компонент связности от радиуса приема узла
* Количество узлов в главной компоненте от радиуса приема узла
* Количество ребер в главной компоненте от радиуса приема узла
* Расстояние ближайшей точки главной компоненты до границы области от радиуса приема узла
* Площадь многоугольника от радиуса приема узла
* Максимальная степень вершины главной компоненты от количества узлов в графе
* Расстояние от центра тяжести многоугольника до центра области от количества узлов в графе
* Количество углов в многоугольнике от количества узлов в графе
* Количество компонент связности от количества узлов в графе
* Количество узлов в главной компоненте от количества узлов в графе
* Количество ребер в главной компоненте от количества узлов в графе
* Расстояние ближайшей точки главной компоненты до границы области от количества узлов в графе
* Площадь многоугольника от количества узлов в графе

Графики формируются при определенном количестве итераций на каждую точку. Модуль содержит метод *InitGraphTestMode*, где одним из параметров целочисленного типа является *countTestCycle* – количество итераций.

Для статистического анализа важный аспект – это количество тестов, которые необходимо выполнить, чтобы получить наиболее точные результирующие данные. На данном этапе формулируется задача об определении минимального количества итераций для более точных вычислений. Построены соответствующие графики - рисунок 12 и рисунок 13 при различном количестве итераций, а именно 4000, 2000, 200.

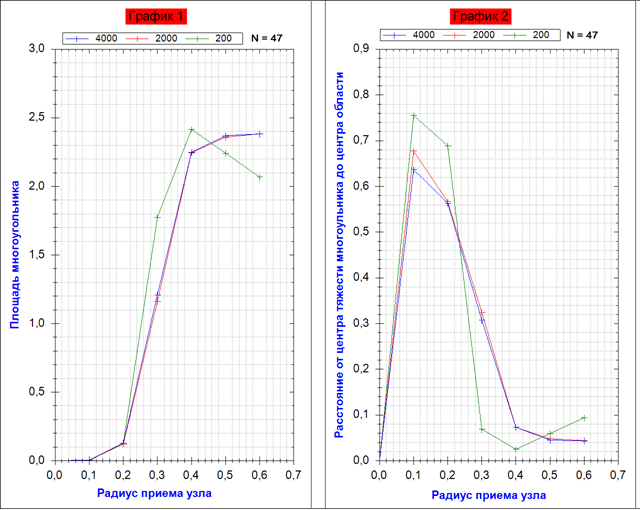


Рисунок 12 – График 1 Рисунок 13 – График 2

На рисунке 12 построен график зависимости площади многоугольника от радиуса приема узла, при итерациях: 4000 – соответствует прямой синего цвета, 2000 – соответствует прямой красного цвета, 200 – соответствует прямой зеленого цвета. Наибольшее отклонение можно заметить у линии зеленого цвета. Тем не менее у построенных прямых при итерациях 4000 и 2000 график функции выглядит практически одинаково и имеет относительно малую погрешность. Следовательно, можно заключить, что результаты, полученные при 4000 и 2000 итерациях практически эквиваленты. Из двух величин выбираем меньшее, так как, чем меньше выбранная величина, тем меньше потребуется времени для получения статистических результатов.

К рисунку 13 также применимы вышеперечисленные рассуждения.

1. Клиент-сервер

3.1 Общие сведения

Клиент-сервер — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Физически клиент и сервер — это [программное обеспечение.](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) Обычно они взаимодействуют через компьютерную сеть посредством [сетевых протоколов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) и находятся на разных вычислительных машинах, но могут выполняться также и на одной машине. Программы — сервера, ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных. Пример коммуникации клиент-серверного приложения изображен на рисунке 14.

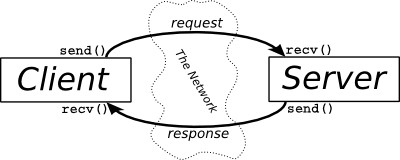


Рисунок 14 – Коммуникация сервера с клиентом

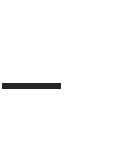
Рассмотрим преимущества и недостатки использования клиент-серверной сетевой архитектуры.

Преимущества:

* Клиентам не нужно дублировать код серверной части программы
* Низкие требования к программам клиентам, так как все основные вычисления выполняются на сервере
* Все данные хранятся на сервере
* Организация контроля полномочий, только клиенты с

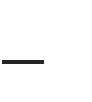
соотвествующими правами доступа могут осуществлять назначенные им соотвествующие операции Недостатки:

* Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п
* Поддержка работы данной системы требует отдельного специалиста

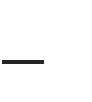
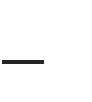
 Высокая стоимость оборудования

Платформа .*NET Framework* — это управляемая среда выполнения, предоставляющая широкий спектр служб для создания всех типов приложений. Она состоит из двух основных компонентов: общеязыковой выолняемой среды *CLR* и библиотеки классов *.NET Framework*. Рассмотрим службы, которые предоставляет *.NET Framework* выполняемым в ней приложениям:

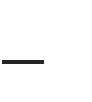
Управление памятью. Во многих языках программисты должны предусматривать выделение и освобождение памяти, а также управлять временем жизни объектов. В приложениях *.NET Framework* эти функции выполняет среда *CLR*



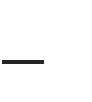
Общая система типов. В традиционных языках программирования базовые типы определяются компилятором, что усложняет взаимодействие между языками. В *.NET Framework* базовые типы определяются системой типов *.NET Framework*, при этом для всех языков *.NET Framework* используются одни и те же базовые типы Обширная библиотека классов. Разработчикам не требуется писать код для выполнения стандартных низкоуровневых операций программирования, поскольку они могут использовать удобную библиотеку входящую в библиотеку классов *.NET Framework*



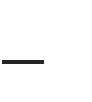
Платформы и технологии разработки. Платформа *.NET Framework* включает библиотеки для конкретных областей разработки приложений, например *ASP.NET* для веб-приложений, *ADO.NET* для доступа к данным и *Windows Communication Foundation* для приложений, ориентированных на службы



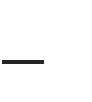
Взаимодействие языков. Языковые компиляторы, ориентированные на *.NET Framework*, выдают промежуточный код, называемый языком *CIL* (*Common Intermediate Language*), который в свою очередь компилируется во время выполнения средой *CLR*. С помощью этой функции подпрограммы, написанные на одном языке, доступны в других языках, поэтому разработчики могут создавать приложения на одном или нескольких предпочитаемых языках



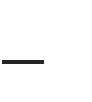
Совместимость версий. Как правило приложения, разработанные на основе конкретной версии платформы *.NET Framework*, могут



выполняться без доработок и на более поздних версиях платформы Параллельное выполнение. Платформа *.NET Framework* помогает разрешать конфликты версий, поскольку на компьютере могут быть установлены несколько версий среды *CLR*. Это означает, что несколько версий приложений также могут существовать, и что приложение может выполняться на версии платформы *.NET Framework*, для которой оно было создано



Настройка для различных версий. Используя переносимую библиотеку классов платформы *.NET Framework*, разработчики могут создавать сборки, которые работают на нескольких платформах *.NET Framework*, например *Windows 7, Windows 8, Windows 8.1, Windows 10, Windows*



*Phone и Xbox 360*

[Абрамян М. Э. Visual C# на примерах. с. 256]

Платформа *Microsoft .NET Framework* предоставляет многоуровневую, расширяемую и управляемую реализацию служб сети, которые можно быстро и легко интегрировать в приложения.

Классы доступа к сети в  [*System.Net*](https://msdn.microsoft.com/ru-ru/library/system.net(v=vs.110).aspx)  и пространства имен [*System.Net.*](https://msdn.microsoft.com/ru-ru/library/system.net(v=vs.110).aspx)*Sockets* можно использовать для реализации клиент-серверных приложений.

Класс *Socket* поддерживает 2 основных режима, синхронный и асинхронный. В синхронном режиме, вызовы функций, которые выполняют сетевые операции (например *Send и* [*Receive*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket.receive(v=vs.110).aspx) ) ожидают, пока операция не завершится до возвращения элемента управления в вызывающей программе. В асинхронном режиме, эти вызовы возвращают немедленно.

Для приложений, которым необходимо выполнить простые запросы, класс [*WebClient*](https://msdn.microsoft.com/ru-ru/library/system.net.webclient(v=vs.110).aspx) предоставляет общие методы для передачи данных или загрузки данных с сервера через интернет. [*WebClient*](https://msdn.microsoft.com/ru-ru/library/system.net.webclient(v=vs.110).aspx) основан на классе *WebRequest* для обеспечения доступа к интернет-ресурсам. Поэтому класс [*WebClient*](https://msdn.microsoft.com/ru-ru/library/system.net.webclient(v=vs.110).aspx) может использовать любой зарегистрированный подключаемый протокол.

Для приложений, которые не могут использовать модель запроса - ответа или для приложений, которые должны прослушиваться в сети так же, как отправлять запросы, *System.Net.Sockets* - пространство имен предоставляет классы [*TCPClient,*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.tcpclient(v=vs.110).aspx) [*TCPListener*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.tcplistener(v=vs.110).aspx) и [*UDPClient.*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.udpclient(v=vs.110).aspx) Эти классы отвечают за установление подключений с использованием различных транспортных протоколов и предоставляют сетевые подключения другим приложениям в виде потоков. [Троелсен Э. Язык программирования C# и платформа .NET с. 478]

3.2 Сервер TCP

TCP, или *Transmission Control Protocol*, используется как надежный протокол, обеспечивающий взаимодействие через взаимосвязанную сеть компьютеров. TCP проверяет, что данные доставляются по назначению и правильно.

TCP — это ориентированный на соединения протокол, предназначенный для обеспечения надежной передачи данных между процессами, выполняемыми или на одном и том же компьютере или на разных компьютерах. Термин “ориентированный на соединения" означает, что два процесса или приложения прежде чем обмениваться какими-либо данными должны установить TCPсоединение. В этом TCP отличается от протокола UDP, являющегося протоколом “без организации соединения”, позволяющим выполнять широковещательную передачу данных неопределенному числу клиентов.

В данной работе в качестве протокола коммуникации между клиентом и сервером используется TCP, чтобы обеспечить упорядоченные, надежные двусторонние потоки байтов. Рассмотрим структуру работы сервера, изображенную на рисунке 15.

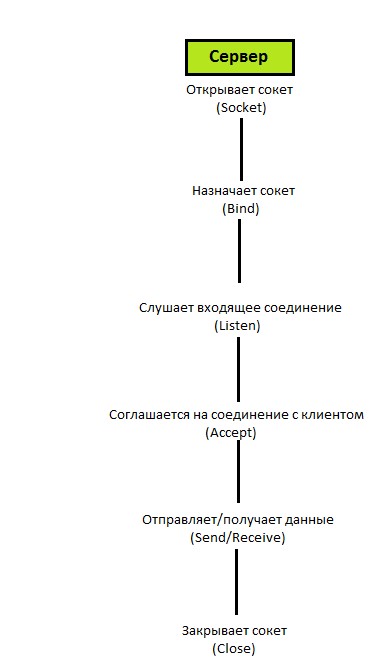


Рисунок 15 – Функциональная диаграмма структуры сервера

На первом шаге необходимо выполнить установление локальной конечной точки для сокета. Прежде чем открывать сокет для ожидания соединений, нужно подготовить для него адрес локальной конечной точки. Уникальный адрес для обслуживания TCP/IP определяется комбинацией IPадреса хоста с номером порта обслуживания, которая создает конечную точку для обслуживания. После того как адрес локальной конечной точки будет подготовлен, можно выполнять открытие сокета. Следующим шагом должно быть назначение сокета с помощью метода *Bind().*Для сокета открытого с помощью конструктора резервируется дескриптор, но не назначается имя.

Чтобы сокет клиента мог идентифицировать потоковый сокет TCP, серверная программа должна назначить имя своему сокету. Метод *Bind()* связывает сокет с локальной конечной точкой. Вызов метода *Bind()* должен быть выполнен строго до любых вызовов методов *Listen()* и *Accept().*

После создания сокета и назначение ему имени, можно выполнять прослушивание входящих соединений используя метод *Listen().*В состоянии прослушивания сокет будет ожидать входящие попытки соединения. Метод *Accept()* извлекает из очереди ожидающих запросов первый запрос на соединение и создает для его обработки новый сокет. Новый сокет создан,но первоначальный сокет продолжает слушать и может использоваться с многопоточной обработкой для приема нескольких запросов на соединение от клиентов.Согласно клиент-серверной архитектуре, сокет выполняющий прослушивание соединений должен продолжать работу наряду с сокетами, созданными методом *Accept().*

Как только клиент и сервер установили между собой соединение, можно отправлять и получать сообщения, используя методы *Send()* и *Receive()* класса Socket. Метод *Send()* записывает исходящие данные сокету, с которым установлено соединение. Метод *Receive()* считывает входящие данные в потоковый сокет. При использовании системы, основанной на TCP, перед выполнением методов *Send()* и *Receive()* между сокетами должно быть установлено соединение.

Точный протокол между двумя взаимодействующими сущностями должен быть определен заблаговременно, чтобы клиентское и серверное приложения не блокировали друг друга, не зная, кто должен отправить свои данные первым.

Вызов метода Close() должен быть выполнен после того как обмен данными между сервером и клиентом будет завершен. [Кровчик Э. Net. Сетевое программирование для профессионалов с.176]

3.3 Клиент TCP

Приложение клиент имеет схожие функции, как и для приложения сервера.Так как используются те же самые методы для определения конечной точки, создания экземпляра сокета, отправки и получения данных и закрытия сокета: На рисунке 16 изображена функциональная диаграмма структуры клиента.



Рисунок 16 - Функциональная диаграмма структуры клиента

Метод *Connect()* – осуществляет установку связи с удаленным сервером.Если применяется протокол, ориентированный на установление соединения, такой как протокол TCP, метод *Connect()* синхронно устанавливает сетевое соединение между [*LocalEndPoint*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket.localendpoint(v=vs.110).aspx) и указанной удаленной конечной точкой.Если используется протокол без установления соединения, метод Connect устанавливает удаленный узел по умолчанию.После вызова метода Connect можно послать данные на удаленное устройство с помощью метода [*Send(*](https://msdn.microsoft.com/ru-ru/library/f3kd48xs(v=vs.110).aspx)*)* или получить данные с удаленного устройства, применив метод [*Receive(*](https://msdn.microsoft.com/ru-ru/library/26f591ax(v=vs.110).aspx)*).*

Метод *Connect()* будет выполнять блокирование, если только не произведена установка для свойства [*Blocking*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket.blocking(v=vs.110).aspx) значения *false* перед вызовом метода *Connect().* Если используется протокол, ориентированный на

установление соединения, такой как протокол TCP, и отключено блокирование, метод *Connect()* создаст исключение [*SocketException*,](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socketexception(v=vs.110).aspx) так как требуется время на установление подключения.Для протоколов без установления соединения исключение не создается, так как они просто работают с удаленным узлом, заданным по умолчанию.Для получения определенного кода ошибки можно воспользоваться свойством [*SocketException.ErrorCode*.](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socketexception.errorcode(v=vs.110).aspx)Получив этот код, можно обратиться за подробным описанием ошибки к документации по кодам ошибок *API Windows Socket* версии 2 в библиотеке MSDN.Если ошибка возвращает значение *WSAEWOULDBLOCK*, подключение к удаленному узлу было инициировано объектом [*Socket*,](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx) работающим в режиме с установлением соединения, но это подключение пока еще не завершено.Необходимо воспользоваться методом [Poll(](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket.poll(v=vs.110).aspx)), чтобы определить, когда объект [*Socket*](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx) завершает установление соединения. [Кровчик Э. Net. Сетевое программирование для профессионалов с.187]

3.4 Интерфейс клиента

Интерфейс клиента состоит из элементов управления, взаимодействуя с которыми, пользователь получает искомые вычисления. На рисунке 17 изображен главный интерфейс клиента.

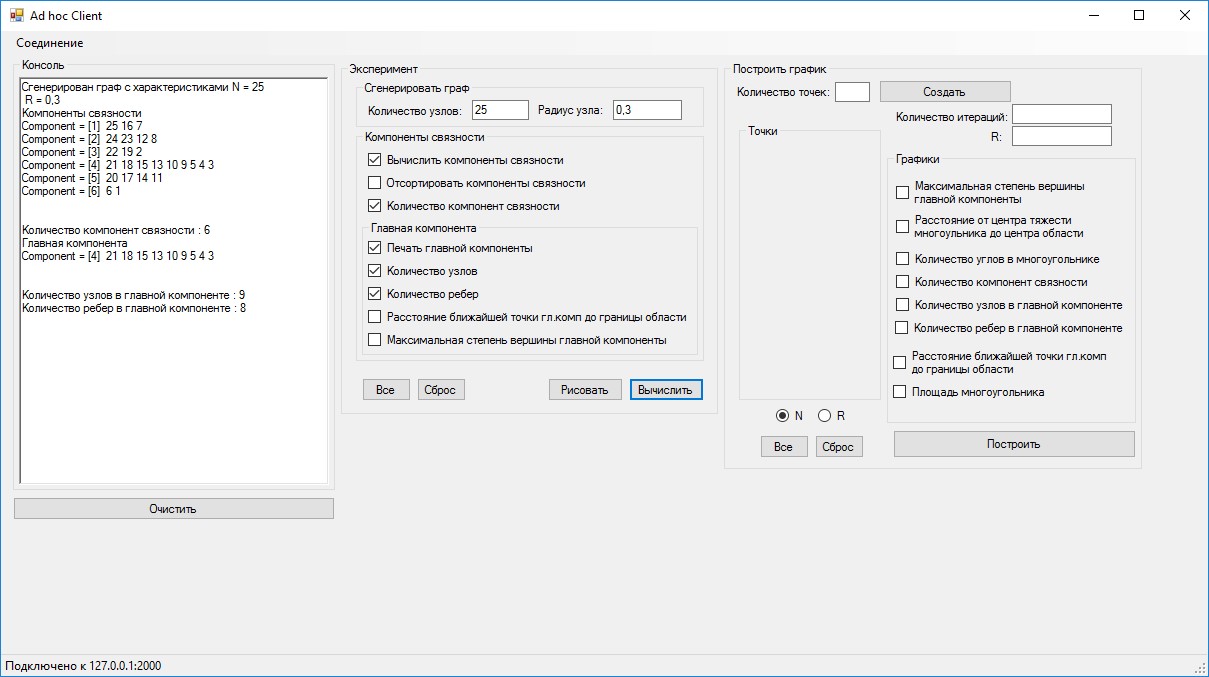


Рисунок 17 – Главный интерфейс клиента

Прежде, чем начинать выполнение сетевых операций, необходимо установить соединение с целевым сервером. Для этого необходимо перейти в меню – соединение – параметры подключения и указать ip адрес и порт для связи с целевым сервером. Определить статус подключения можно с помощью статус строки, расположенной в нижней части окна приложения.

Не подключен – соединение с сервером не установлено.

Подключено к ip : port – соединение с сервером установлено, где ip и port, это соответственно ip адрес и порт сервера.

Если сервер недоступен приложение выведет соответствующее сообщение.

Валидация формы осуществляется на клиенте, в случае ввода некорректных данных пользователь получит соответствующее сообщение об ошибке, данные не будут отправлены на сервер.

Рассмотрим функции для исследования ad hoc сетей. Одиночный эксперимент – пользователь указывает количество узлов, которое необходимо сгенерировать и радиус приема узла. Вычисления, которые доступны:

* Количество компонент связности.
* Поиск главной компоненты.
* Количество узлов в главной компоненте.
* Количество ребер в главной компоненте.
* Расстояние ближайшей точки главной компоненты до границы области.
* Максимальная степень вершины главной компоненты

После выбора всех необходимых параметров для вычисления, пользователь нажимает на кнопку “Вычислить” – запрос отправляется к серверу, после обработки запроса сервером результаты вычислений будут отображены в консоли клиента. На рисунке 18 изображен детальный интерфейс одиночного эксперимента.

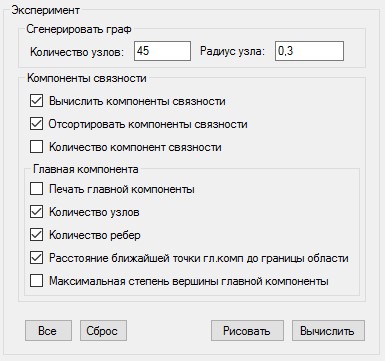


Рисунок 18 – Интерфейс одиночного эксперимента

Также можно наглядно представить, как расположены и связанны между собой узлы в ограниченной области. Кнопка “Рисовать” создает новое окно, содержимое которого будет сформировано из сгенерированного графа. На рисунке 19 изображено расположение узлов сгенерированного графа с параметрами *n =* 20 и *r* *=* 0,3.

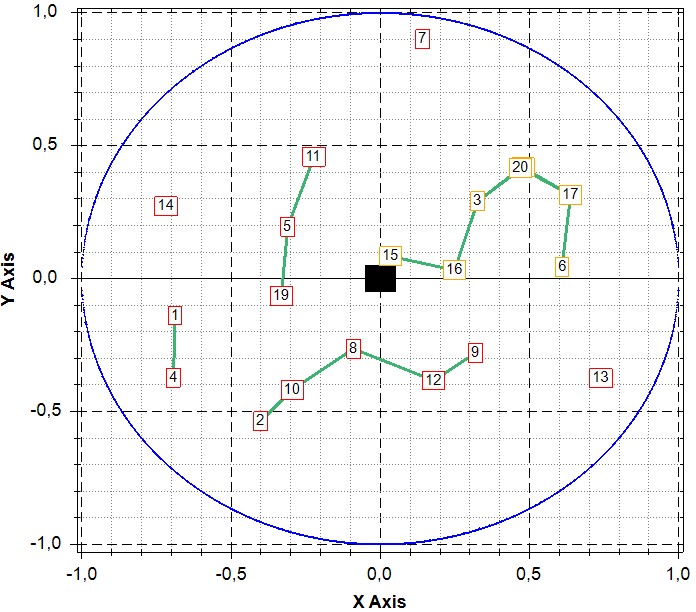


Рисунок 19 – Расположение узлов сгенерированного графа

Дескрипторы узлов, помеченные желтым цветом, являются узлами главной компоненты. Дескрипторы остальных узлов помечены красным. Если между узлами есть связь, то между ними ставится ребро, окрашенное в зеленый цвет.

В качестве библиотеки для рисования используется *ZedGraph*. Данная библиотека обеспечивает высокую степень гибкости - почти каждый аспект графа может быть установлен пользователем и изменен.

Пользователь может также построить выпуклую оболочку. К узлам, составляющим главную компоненту можно применить алгоритм *QuickHull* реализован в отдельном модуле в *Algorithms.cs*.Для этого необходимо в том же окне выполнить нажатие кнопки “*Shape*”. На рисунках 20,21,22 показана последовательность действий необходимая для построения фигуры.

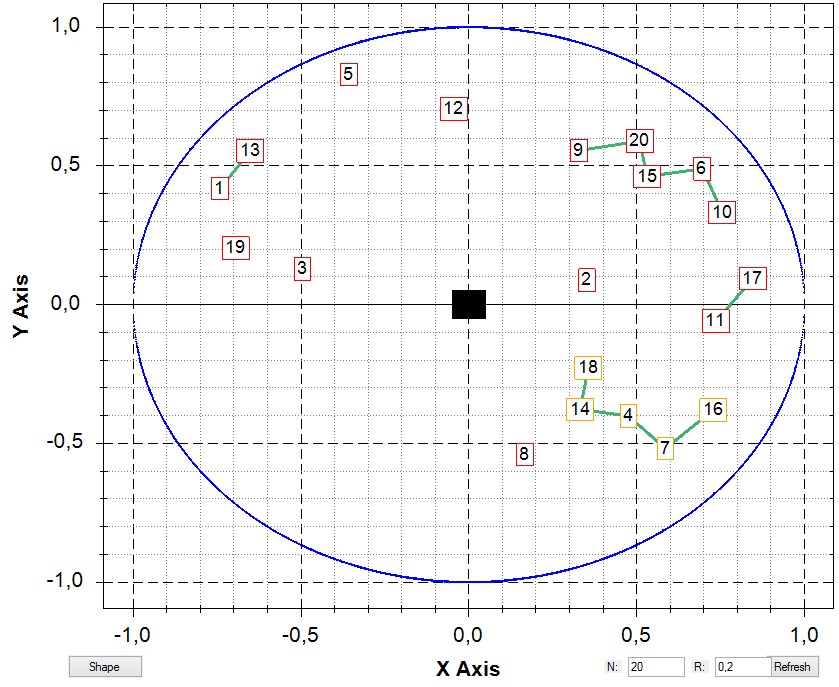


Рисунок 20 – Сгенерированный граф *n=*20, *r=*0.2

После генерации заданного количества узлов n=20 с определенным радиусом приема каждого из них *r=*0,2, сформируются компоненты связности, главная компонента содержит наибольшее количество узлов.

Входными данными *QuickHull* является набор точек *(х,y)* главной компоненты из которых и будет в дальнейшем построен наименьший выпуклый многоугольник. Для этих узлов выполним алгоритм *QuickHull* для построения выпуклой оболочки.

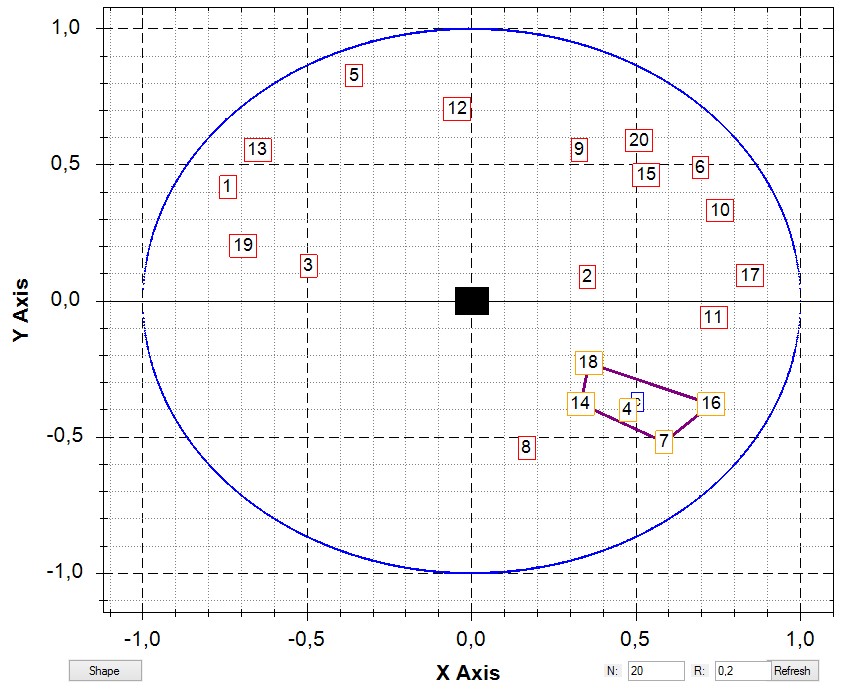


Рисунок 21 – Построенный выпуклый многоугольник

Построенная фигура содержит узлы с дескрипторами 18,16,7, 14.Данная выпуклая оболочка является четырехугольником, на рисунке 21 стороны фигуры окрашены в фиолетовый цвет.

Для отрисовки получившейся фигуры используется обьект *Arrow* библиотеки *ZedGraph*, у объекта имеется конструктор, который принимает обязательные параметры при инициализации в виде *(x1,y1,x2,y2).*Свойство *Line.Width* задает ширину линии отрисованной фигуры, *Line.Color* – задает цвет используемой линии.

После всех операций инициализации нового обьекта *Arrow* необходимо добавить его в *GraphObjList* с помощью метода *Add*. Все операции, связанные с перерисовкой, выполняются на клиенте.

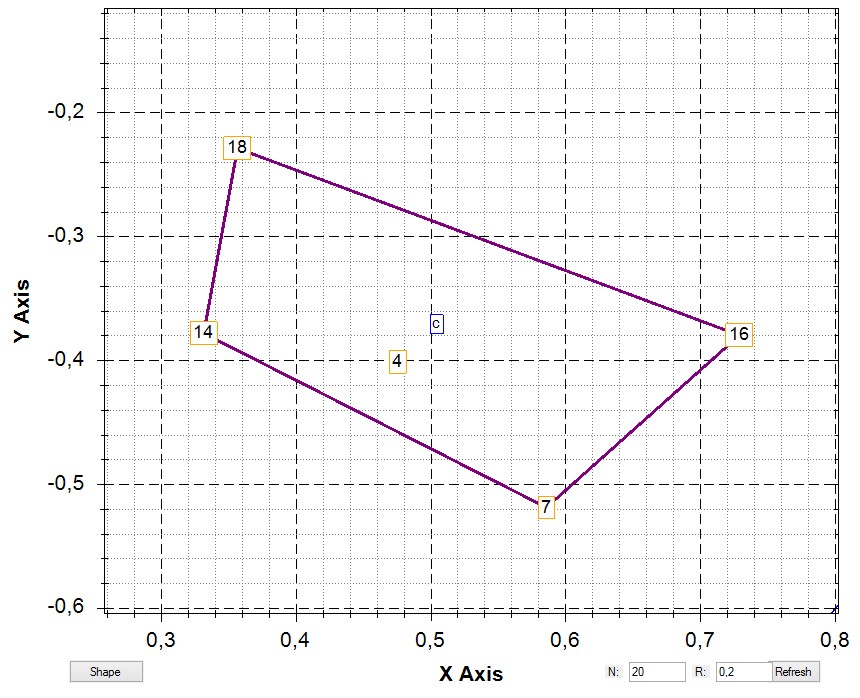


Рисунок 22 – Построенный выпуклый многоугольник

На рисунке 22 практически в центре фигуры обозначена точка. *С* – центр тяжести многоугольника. Центром тяжести треугольника принято называть точку *M* пересечения его медиан.

Рассмотрим интерфейс, с помощью которого клиент выполняет построение графиков зависимостей для исследования мобильных беспроводных сетей. В программе клиента - пользователь указывает количество точек, из вычисленных значений которых будет формироваться искомый график. Тип точек – радиус узла или количество узлов, соответственно *N* или *R*. Далее указывает количество итераций вычислений на одну точку.

Доступные графики:

* Максимальная степень вершины главной компоненты
* Расстояние от центра тяжести многоугольника до центра области
* Количество углов в многоугольнике
* Количество компонент связности
* Количество узлов в главной компоненте
* Количество ребер в главной компоненте
* Расстояние ближайшей точки главной компоненты до границы области  Площадь многоугольника

На рисунке 23 изображен детальный интерфейс, с помощью которого можно выбрать какие именно графики необходимо построить.

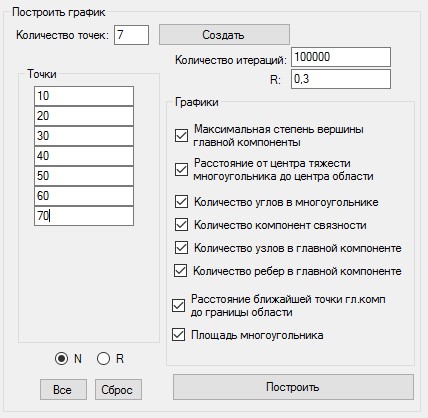


Рисунок 23 – Интерфейс для построения графиков

Данные графики формируются с зависимостью от изменения параметра N или R, т.е. от количества узлов или радиуса приема узла соответственно.

По нажатию кнопки “Построить” клиент, отправляет запрос серверу, сервер производит вычисления согласно запросу, сформированному клиентом.

По окончанию вычислений сервер отправляет данные клиенту, клиент в свою очередь из полученных данных выполняет построение графиков.

Для исследования математических ожиданий, примем во внимание, что случайная величина имеет [дискретное равномерное распределение](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D0%B2%D0%BD%D0%BE%D0%BC%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) как на формуле *(1).*

*(1)*

Тогда её математическое ожидание *(2):*

*(2)*

равно среднему арифметическому всех принимаемых значений. Руководствуясь формулой *(2)* вычислим математические ожидания для следующих величин:

* Максимальная степень вершины главной компоненты.
* Расстояние от центра тяжести многоугольника до центра области.
* Количество углов выпуклого многоугольника.
* Количество компонент связности.
* Количество узлов в главной компоненте.
* Количество ребер в главной компоненте.
* Расстояние от границы многоугольника до границы области.
* Площадь выпуклого многоугольника.

Затем смоделируем графики зависимостей, изображающие характеристики сети при указанных параметрах.

Как уже рассматривалось в пункте 2.4, достаточно 2000 итераций для вычисления каждой точки графика с минимальной погрешностью. Таким образом все вышеперечисленные графики будут построены с количеством экспериментов равным 2000 на каждую точку. На рисунке 24 изображен график зависимости максимальной степени вершины главной компоненты от количества узлов в графе при *R =* 0,25 каждого узла.

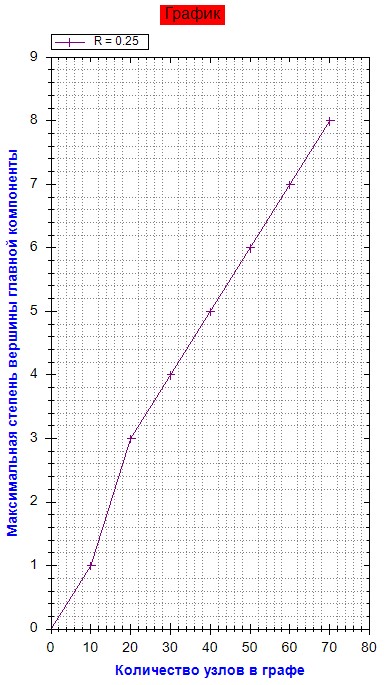


Рисунок 24 – График зависимости максимальной степени вершины главной

компоненты от количества узлов в графе

На рисунке 25 изображен график зависимости расстояния от центра тяжести многоугольника до центра области от количества узлов в графе при *R =*0,25.

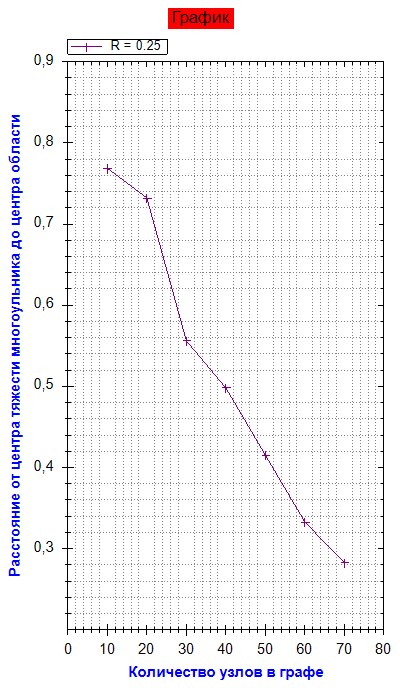


Рисунок 25 – График зависимости расстояния от центра тяжести многоугольника до центра области от количества узлов в графе На рисунке 26 изображен график зависимости количества углов в многоугольнике от количества узлов в графе, при *R =*0,25.

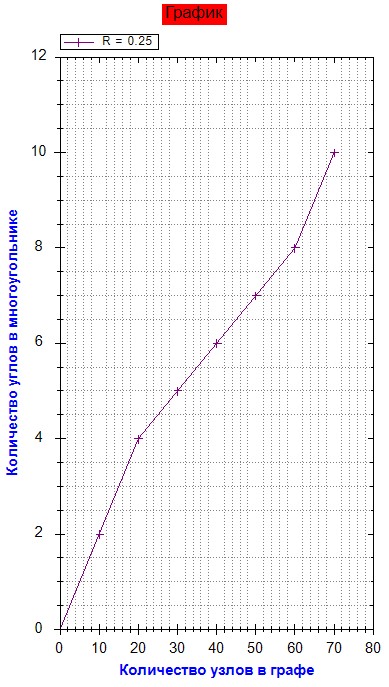


Рисунок 26 – График зависимости количества углов в многоугольнике от количества узлов в графе

На рисунке 27 изображен график зависимости количества компонент связности от количества узлов в графе, при *R =*0,25.

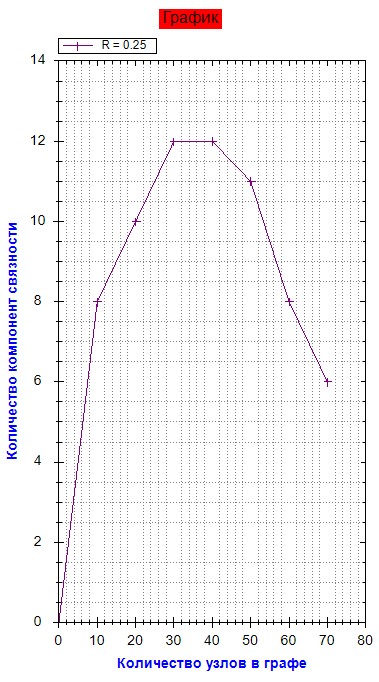


Рисунок 27 – График зависимости количества компонент связности от количества узлов в графе

На рисунке 28 изображен график зависимости количества узлов в главной компоненте от радиуса приема узла, при *N =* 40.

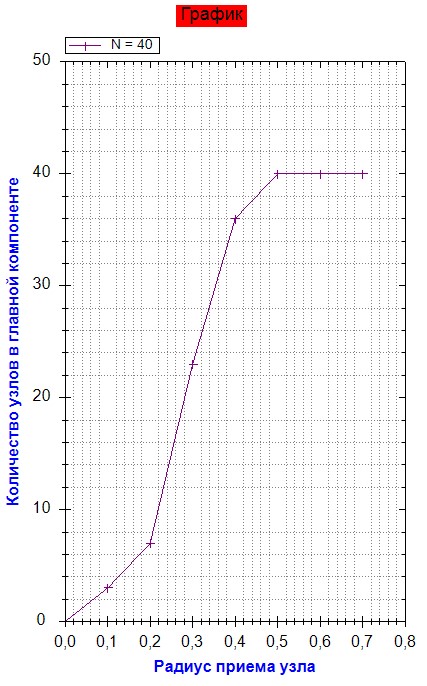


Рисунок 28 – График зависимости количества узлов в главной компоненте от радиуса приема узла

На рисунке 29 изображен график зависимости количества ребер в главной компоненте от радиуса приема узла, при *N =* 40.

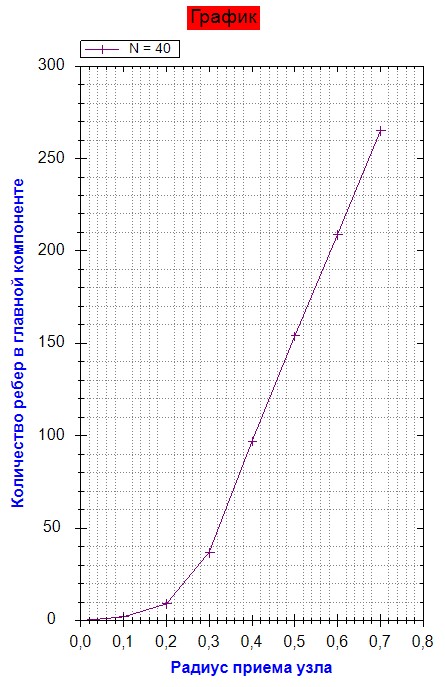


Рисунок 29 – График зависимости количества ребер в главной компоненте от радиуса приема узла

На рисунке 30 изображен график зависимости расстояния ближайшей точки главной компоненты до границы области от радиуса приема узла, при *N =*40.

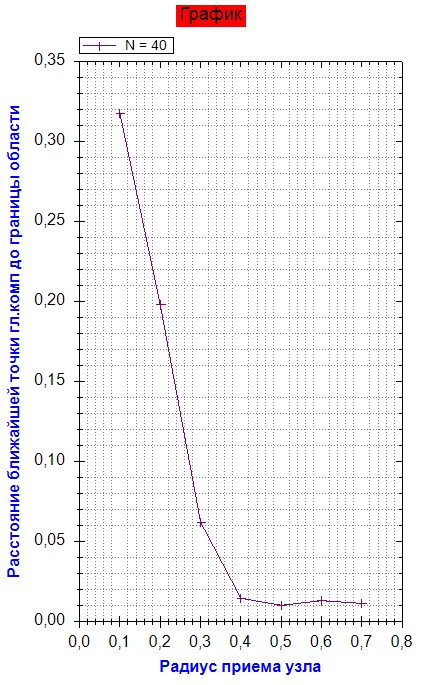


Рисунок 30 – График зависимости расстояния ближайшей точки главной компоненты до границы области от радиуса приема узла

На рисунке 31 изображен график зависимости площади многоугольника от количества узлов в графе, при *R =* 0,25.

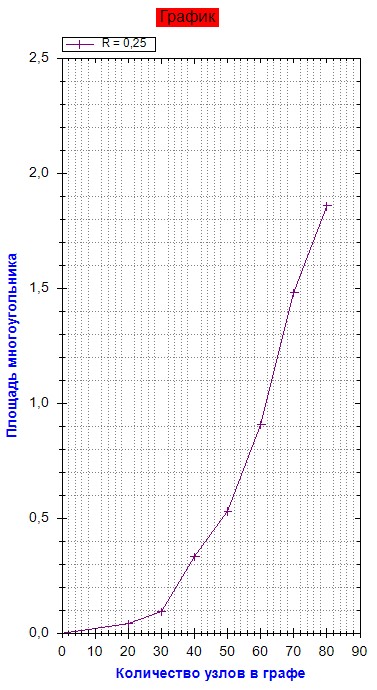


Рисунок 31 – График зависимости площади многоугольника от количества узлов в графе

3.5 Интерфейс сервера

Интерфейс сервера представляет собой консольное приложение. В консоли отображаются строки вида: “ip адрес – метод”, т.е. можно посмотреть какие методы вызывал определенный пользователь. На рисунке 32 изображено консольное приложение – сервер.

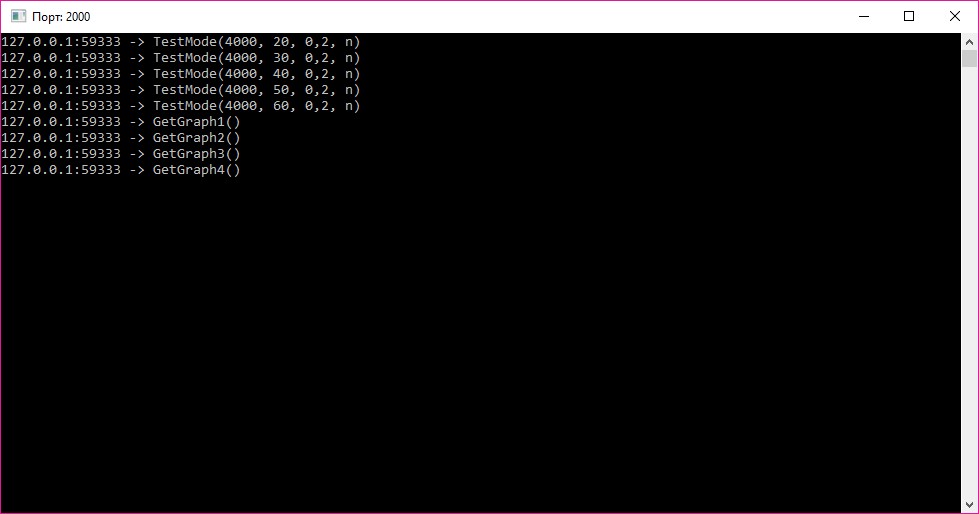


Рисунок 32 – Консольное приложение – сервер

Рассмотрим подробно строки, изображенные в окне консольного приложения на рисунке 32.

* 127.0.0.1:59333 - *TestMode(4000, 20, 0,2, n)*
* 127.0.0.1:59333 - *TestMode(4000, 30, 0,2, n)*
* 127.0.0.1:59333 - *TestMode(4000, 40, 0,2, n)*
* 127.0.0.1:59333 - *TestMode(4000, 50, 0,2, n)*
* 127.0.0.1:59333 - *TestMode(4000, 60, 0,2, n)*

*TestMode* *(4000, 60, 0,2, n)* – метод с параметрами, вызов которого выполнил клиент с ip адресом 127.0.0.1:59333.

В данном случае вызов вышеперечисленных методов означает вычисление точек для построения графиков при параметрах: количество итераций цикла на точку = 4000, количество узлов = [20,30,40,50,60], радиус приема узла = 0,2, последний параметр n означает, что радиус постоянен, а изменяется только количество узлов.

* 127.0.0.1:59333 - *GetGraph1()*
* 127.0.0.1:59333 - *GetGraph2()*
* 127.0.0.1:59333 - *GetGraph3()*
* 127.0.0.1:59333 - *GetGraph4()*

Методы возвращающие значения для построения соответствующих графиков. В данном случае так как было выполнено 5 вызовов *TestMode* – каждый график будет построен по 5 точкам.

*GetGraph1()* – Максимальная степень вершины главной компоненты от

количества узлов в графе.

*GetGraph2()* - Расстояние от центра тяжести многоугольника до центра области от количества узлов в графе.

*GetGraph3()* -Количество углов в многоугольнике от количества узлов в графе.

*GetGraph4()* -Количество компонент связности от количества узлов в графе.

Изменить порт и другие сетевые настройки сервера можно путем редактирования файла *Settings.conf.,* расположен в корне с программой

сервером. По умолчанию в качестве порта используется 2000.

Приложение работает на платформе *NET*, для обеспечения работоспособности необходимо наличие предустановленного пакета *NET Framework* версии 4 и выше. Также приложение сервер для коммуникации с клиентами не должно быть блокировано приложениями брандмауэрами.Для настройки необходимо разрешить активность на выбраном порту или добавить его в исключение.

ЗАКЛЮЧЕНИЕ

С помощью разработанных программных модулей можно создавать, имитационные модели ad hoc сетей, концептуально представимых в виде графа. Модули содержат в себе необходимый комплекс процедур и функций по редактированию и выполнению различных операций над создаваемыми моделями. Функции по анализу и обработке моделей ad hoc, основанных на графовых структурах, могут работать как с конкретными узлами модели, так и со всей моделью целиком. Так же предусмотрены функции, предоставляющие вывод в удобочитаемом виде созданных ad hoc структур.

С помощью разработанного клиент –серверного приложения, клиенты могут выполнять построение и исследование моделей ad hoc сетей из любой точки, так как все вычисления выполняются на сервере. Данная коммуникация не требует наличия мощного ПК, тем самым не ограничивает пользователя в количестве, итераций которое необходимо для получения среднего вычисления с минимальной погрешностью. Серверная часть приложения использует разработанные программные модули. Клиенты могут выполнить построение различных графиков зависимостей.

Использование данных разработок поможет существенно сэкономить время развертывания данных сетей на физическом уровне, определить, в каком расположении сеть связна и имеет наилучшие характеристики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы построения выпуклой оболочки. [Электронный ресурс] URL:

http://algolist.manual.ru/maths/geom/convhull/

Дата обращения [15 апреля 2016].

1. Абрамян М. Э. Visual C# на примерах. // СПб.: БХВ-Петербург, 2008. – 496 с
2. Винокуров В.М., Пуговкин А.В., Пшенников А.А., Ушарова Д.Н.

Маршрутизация в беспроводных мобильных Ad hoc сетях. // СПб: Питер,

* 1. – 396 с.

1. Джим Г. Беспроводные сети. Принципы, технологии, протоколы. // СПб:

Питер, 2005. – 189 с.

1. Кровчик Э. Net. Сетевое программирование для профессионалов // Москва
   1. – 417 с.
2. Миков А.И. Графы и грамматики. – Краснодар: Кубанский государственный

университет, 2014. – 160 с.

1. Свами М., Тхуласираман К. Графы, сети и алгоритмы. М.: Мир, 1984, - 454 с.
2. Троелсен Э. Язык программирования C# и платформа .NET // Москва 2011. – 1392 с.

ПРИЛОЖЕНИЕ А

Графики зависимости математических ожиданий величин

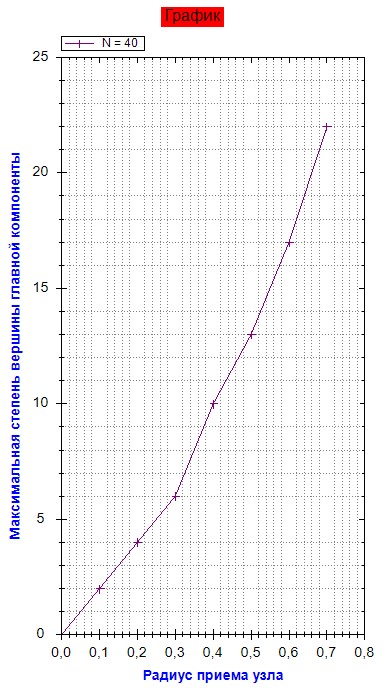


Рисунок А.1 – График 1 – зависимость максимальной степени вершины

главной компоненты от радиуса приема узла

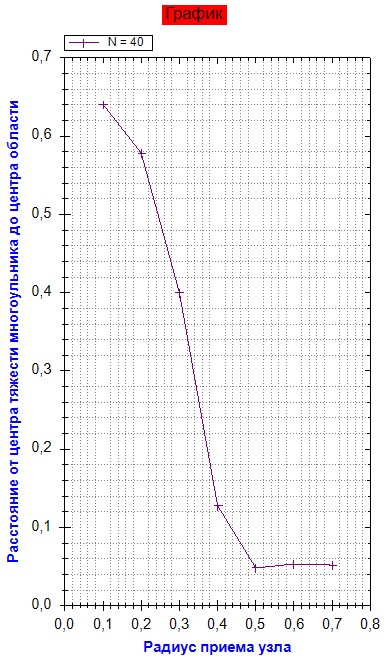


Рисунок А.2 – График 2 – зависимость расстояния от центра тяжести многоугольника до центра области от радиуса приема узла

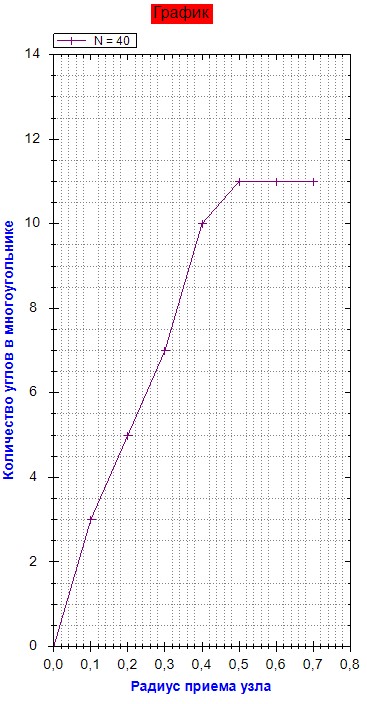


Рисунок А.3 – График 3 – зависимость количества углов в многоугольнике от радиуса приема узла

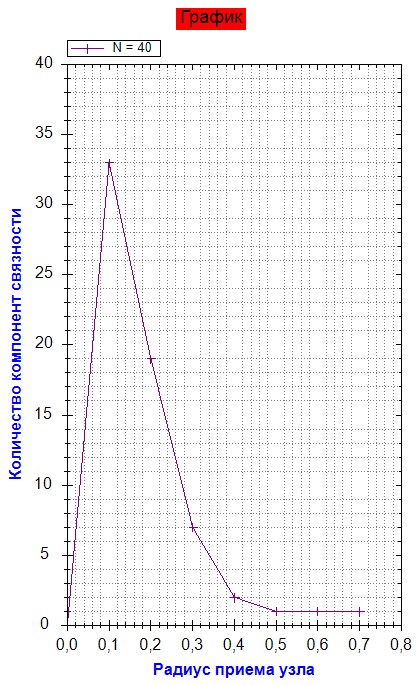


Рисунок А.4 – График 4 – зависимость количества компонент связности от радиуса приема узла

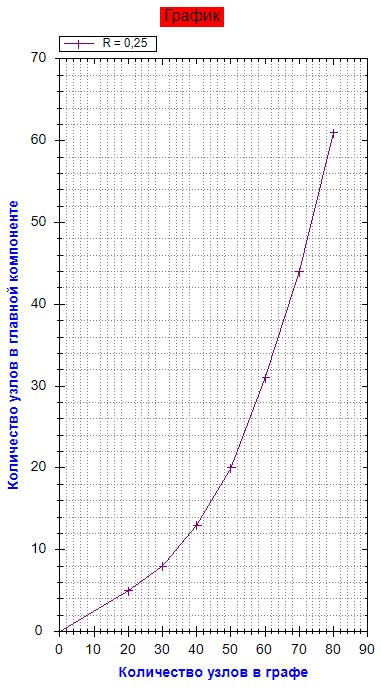


Рисунок А.5 – График 5 – зависимость количества узлов в главной компоненте от количества узлов в графе

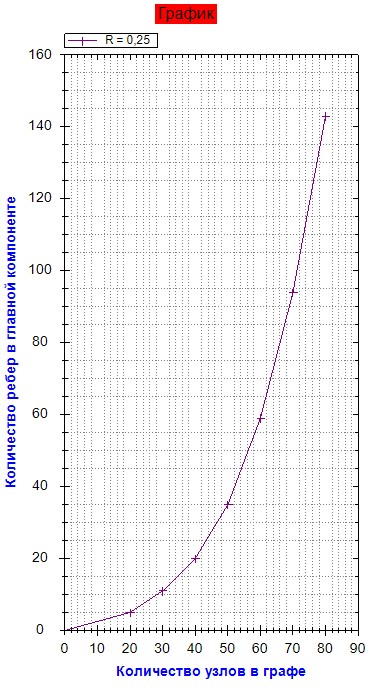


Рисунок А.6 – График 6 – зависимость количества ребер в главной компоненте от количества узлов в графе

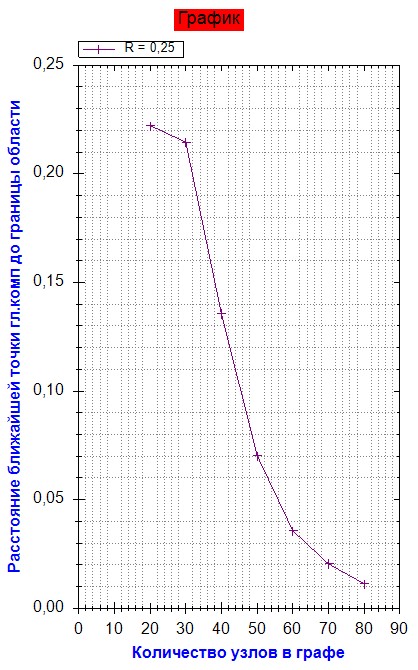


Рисунок А.7 – График 7 – зависимость расстояния ближайшей точки главной

компоненты до границы области от количества узлов в графе

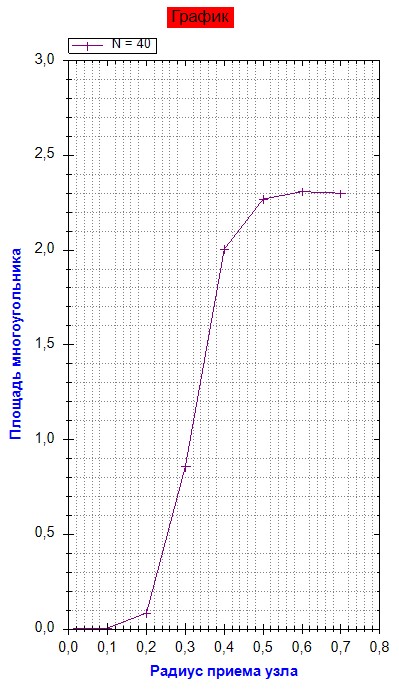


Рисунок А.8 – График 8 – зависимость площади многоугольника от радиуса приема узла

ПРИЛОЖЕНИЕ Б

Исходный код программы

Клиентская часть.

Client.cs

#define USE\_COMPRESSION

using System; using System.IO; using System.IO.Compression; using System.Linq; using System.Net.Sockets; using System.Reflection;

using System.Runtime.Remoting.Messaging; using System.Runtime.Remoting.Proxies;

using System.Runtime.Serialization.Formatters.Binary; using System.Threading; using System.Threading.Tasks; using Interfaces;

namespace Ad\_hoc\_Client

{

public interface IEvents

{

Action OnStartConnect { get; set; }

Action OnConnected { get; set; }

Action OnDisconnect { get; set; }

Action OnPing { get; set; }

Action<Exception> OnError { get; set; }

Action<int> OnBark { get; set; }

}

public class UniservClient : IEvents

{

public const int TCP\_SIZE = 8192; public const int TIMEOUT = 30000; public int Port; public string Host; private TcpClient ServerSocket;

private readonly object tcpSendLock = new object(); private readonly object syncLock = new object(); private Message \_syncResult; private Task ListenerTask;

private CancellationTokenSource ListenerToken;

private readonly ManualResetEventSlim \_OnResponce = new ManualResetEventSlim(false); private readonly object \_IsConnectedLock = new object(); private bool \_IsConnected = false; private bool IsConnected

{ get {

lock (\_IsConnectedLock)

{

return \_IsConnected;

}

}

set {

lock (\_IsConnectedLock)

{

\_IsConnected = value;

}

} }

private readonly Proxy<ICommon> CommonProxy;

public IEvents Events

{

get { return this; }

}

public ICommon Common { get; private set; }

#region Proxy

private class Proxy<T> : RealProxy where T : class

{

UniservClient client;

public Proxy(UniservClient client) : base(typeof(T))

{

this.client = client;

}

public override IMessage Invoke(IMessage msg)

{

IMethodCallMessage call = (IMethodCallMessage)msg; object[] parameters = call.Args; int OutArgsCount = call.MethodBase.GetParameters().Where(x => x.IsOut).Count();

Message result = client.Execute(call.MethodName, parameters); parameters = parameters.Select((x, index) => result.prms[index] ?? x).ToArray();

return new ReturnMessage(result.ReturnValue, parameters, OutArgsCount, call.LogicalCallContext, call);

}

}

#endregion

#region События

Action IEvents.OnStartConnect { get; set; } Action IEvents.OnConnected { get; set; }

Action IEvents.OnDisconnect { get; set; }

Action IEvents.OnPing { get; set; }

Action<Exception> IEvents.OnError { get; set; }

Action<int> IEvents.OnBark { get; set; }

#endregion

public UniservClient()

{

CommonProxy = new Proxy<ICommon>(this);

Common = (ICommon)CommonProxy.GetTransparentProxy();

}

public void StartAsync()

{

ListenerToken = new CancellationTokenSource();

ListenerTask = Task.Factory.StartNew(Listener, TaskCreationOptions.LongRunning);

}

public bool Connect(bool RaiseException)

{ try

{

\_Connect();

}

catch (Exception ex)

{

\_Dicsonnect();

if (RaiseException) throw ex;

}

bool b = IsConnected; StartAsync(); return b;

}

public void ReConnect()

{

if (ListenerTask != null && ListenerTask.Status == TaskStatus.Running) {

ListenerToken.Cancel();

ListenerTask.Wait();

ListenerToken = new CancellationTokenSource();

ListenerTask = Task.Factory.StartNew(Listener,

ListenerTask.CreationOptions);

} else

{

ListenerToken = new CancellationTokenSource();

ListenerTask = Task.Factory.StartNew(Listener, ListenerTask.CreationOptions);

}

}

public void ReConnectAsync()

{

if (ListenerTask != null && ListenerTask.Status == TaskStatus.Running) {

ListenerToken.Cancel();

ListenerTask = ListenerTask.ContinueWith(t =>

{

t.Dispose();

ListenerToken = new CancellationTokenSource();

Listener();

}, (TaskContinuationOptions)ListenerTask.CreationOptions);

} else

{

ListenerToken = new CancellationTokenSource();

ListenerTask = Task.Factory.StartNew(Listener,

ListenerTask.CreationOptions);

}

}

#region Private

private void \_Connect()

{

ServerSocket = new System.Net.Sockets.TcpClient();

ServerSocket.ReceiveBufferSize = TCP\_SIZE;

ServerSocket.SendBufferSize = TCP\_SIZE;

ServerSocket.ReceiveTimeout = TIMEOUT;

ServerSocket.SendTimeout = TIMEOUT;

#region Подключение

if (Events.OnStartConnect != null) Events.OnStartConnect.BeginInvoke(null, null);

ServerSocket.Connect(Host, Port);

if (Events.OnConnected != null) Events.OnConnected.BeginInvoke(null, null);

IsConnected = true;

#endregion

}

private void Listener()

{

while (true)

{ try {

if (ListenerToken.IsCancellationRequested) return;

if (!IsConnected) \_Connect();

while (true)

{

if (ListenerToken.IsCancellationRequested) return;

Message msg = ReceiveData<Message>(); if (msg.Command == "OnPing")

{

// отражаем пинг

SendData(msg); if (Events.OnPing != null) Events.OnPing.BeginInvoke(null, null);

continue;

}

if (msg.IsSync)

{ // получен результат синхронной процедуры

SyncResult(msg);

} else

{

// асинхронный вызов события try

{

// ищем соответствующий Action var pi = typeof(IEvents).GetProperty(msg.Command, BindingFlags.Instance | BindingFlags.Public);

if (pi == null) throw new Exception(string.Concat("Свойство \"", msg.Command, "\" не найдено"));

var delegateRef = pi.GetValue(this, null) as Delegate;

// инициализируем событие if (delegateRef != null) ThreadPool.QueueUserWorkItem(state => delegateRef.DynamicInvoke(msg.prms));

}

catch (Exception ex) { throw new Exception(string.Concat("Не удалось выполнить делегат \"", msg.Command, "\""), ex);

}

}

} }

catch (TaskCanceledException)

{

return;

}

catch (Exception ex)

{

if (Events.OnError != null) Events.OnError.BeginInvoke(ex, null, null);

} finally

{

\_Dicsonnect();

}

Thread.Sleep(2000);

}

}

private void \_Dicsonnect()

{

if (ServerSocket != null) ServerSocket.Close(); IsConnected = false;

if (Events.OnDisconnect != null) Events.OnDisconnect.BeginInvoke(null, null);

\_OnResponce.Set();

GC.Collect(2, GCCollectionMode.Optimized);

}

private Message Execute(string MethodName, object[] parameters)

{

lock (syncLock)

{

\_syncResult = new Message(MethodName, parameters);

\_syncResult.IsSync = true;

\_OnResponce.Reset();

SendData(\_syncResult);

\_OnResponce.Wait(); // ожидаем ответ сервера

if (\_syncResult.IsEmpty)

{// произошел дисконект, результат не получен throw new Exception(string.Concat("Ошибка при получении результата на команду \"", MethodName, "\""));

}

if (\_syncResult.Exception != null) throw \_syncResult.Exception; // исключение переданное сервером

return \_syncResult;

}

}

private void SyncResult(Message msg) { // получен результат выполнения процедуры

\_syncResult = msg;

\_syncResult.IsEmpty = false;

\_OnResponce.Set(); // разблокируем поток

}

private T ReceiveData<T>() where T : class

{

#if USE\_COMPRESSION

byte[] DataLength = BitConverter.GetBytes((int)1);

ServerSocket.GetStream().Read(DataLength, 0, DataLength.Length); int len = BitConverter.ToInt32(DataLength, 0);

byte[] BinaryData = new byte[len];

ServerSocket.GetStream().Read(BinaryData, 0, BinaryData.Length);

using (MemoryStream memory = new MemoryStream(BinaryData))

{ using (var gZipStream = new GZipStream(memory, CompressionMode.Decompress, false))

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); return (T)binaryFormatter.Deserialize(gZipStream);

}

}

#else

BinaryFormatter binaryFormatter = new BinaryFormatter(); return (T)binaryFormatter.Deserialize(ServerSocket.GetStream());

#endif

}

private void SendData(Message msg)

{

#if USE\_COMPRESSION

using (MemoryStream memory = new MemoryStream())

{ using (var gZipStream = new GZipStream(memory, CompressionMode.Compress, false))

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); binaryFormatter.Serialize(gZipStream, msg);

}

byte[] BinaryData = memory.ToArray();

byte[] DataLength = BitConverter.GetBytes(BinaryData.Length); byte[] DataWithHeader = DataLength.Concat(BinaryData).ToArray();

lock (tcpSendLock)

{

ServerSocket.GetStream().Write(DataWithHeader, 0, DataWithHeader.Length);

}

} #else

lock (tcpSendLock)

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); binaryFormatter.Serialize(ServerSocket.GetStream(), msg);

}

#endif

}

#endregion

}

}

Form1.cs

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq;

using System.Runtime.InteropServices; using System.Text; using System.Threading.Tasks; using System.Windows.Forms; using System.Windows;

namespace Ad\_hoc\_Client

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//-----console-----//

[DllImport("kernel32.dll", SetLastError = true)] [return: MarshalAs(UnmanagedType.Bool)] static extern bool AllocConsole();

//----------------//

private static string GetAllNestedMessages(Exception ex)

{

string s = ex.Message; while (ex.InnerException != null)

{

ex = ex.InnerException;

s += string.Concat(Environment.NewLine, ex.Message);

} return s;

} static void OnError(Exception ex)

{

Console.WriteLine(GetAllNestedMessages(ex));

}

private string WhatChecked()

{

if (radioN.Checked)

{

what = "Количество узлов в графе"; return "n";

}

if (radioR.Checked)

{

what = "Радиус приема узла"; return "r";

}

return null;

}

UniservClient client; string what;

private void button1\_Click(object sender, EventArgs e)

{

string a = WhatChecked();

int n = Convert.ToInt32(textBox2.Text);

switch (a) { case "n": {

for(int i=0;i< Convert.ToInt32(textBox1.Text);i++) { client.Common.TestMode(n, Convert.ToInt16(list[i].Text), Convert.ToDouble(textBox3.Text), a);

} break;

} case "r": {

for (int i = 0; i < Convert.ToInt32(textBox1.Text); i++) { client.Common.TestMode(n,

Convert.ToInt16(textBox3.Text),Convert.ToDouble(list[i].Text), a);

} break;

}

}

if(checkBox1.Checked)

{

Form2 a1 = new Form2();

a1.DrawTestPanel(Color.Purple, what, "Максимальная степень вершины главной компоненты", client.Common.GetGraph1());

a1.Show();

}

if(checkBox10.Checked)

{

Form2 a11 = new Form2();

a11.DrawTestPanel(Color.Purple, what, "Расстояние от центра тяжести многоульника до центра области", client.Common.GetGraph2());

a11.Show();

}

if(checkBox11.Checked)

{

Form2 a111 = new Form2(); a111.DrawTestPanel(Color.Purple, what, "Количество углов в многоугольнике", client.Common.GetGraph3());

a111.Show();

}

if(checkBox12.Checked)

{

Form2 a111 = new Form2(); a111.DrawTestPanel(Color.Purple, what, "Количество компонент связности", client.Common.GetGraph4());

a111.Show();

}

if(checkBox13.Checked)

{

Form2 a111 = new Form2();

a111.DrawTestPanel(Color.Purple, what, "Количество узлов в главной компоненте", client.Common.GetGraph5());

a111.Show();

}

if(checkBox14.Checked)

{

Form2 a111 = new Form2();

a111.DrawTestPanel(Color.Purple, what, "Количество ребер в главной компоненте", client.Common.GetGraph6());

a111.Show();

}

if(checkBox15.Checked)

{

Form2 a111 = new Form2();

a111.DrawTestPanel(Color.Purple, what, "Расстояние ближайшей точки гл.комп до границы области", client.Common.GetGraph7());

a111.Show();

}

if(checkBox16.Checked)

{

Form2 a111 = new Form2();

a111.DrawTestPanel(Color.Purple, what, "Площадь многоугольника", client.Common.GetGraph8());

a111.Show();

}

}

public void InitGraph(int power, int countNodes, double r)

{

int temppoint = 0;

client.Common.GenerateGraph(countNodes, r, power);

Console.WriteLine("----------------Components-----------------------------\n");

// Methods.printMap(client.Common.Components());

Console.WriteLine();

Console.WriteLine("--------------Sort Components---------------------------\n");

// Methods.printMap(client.Common.SortedComponents());

Console.WriteLine();

Console.WriteLine("----------------Main Component-------------------------\n");

// Methods.printMap(client.Common.MainComponent());

Console.WriteLine();

Console.WriteLine("------Number of Nodes in main----");

Console.WriteLine(client.Common.NumberNodesInMain());

Console.WriteLine("----------------------------------\n"); Console.WriteLine("------Number of Arcs in main----");

Console.WriteLine(client.Common.GetCountArcsInMain());

Console.WriteLine("----------------------------------\n");

Console.WriteLine("---Distance To the boundary of ----\n");

Console.WriteLine(" distance: " +

client.Common.DistancePointToRoundInMain(out temppoint).ToString() + " \n");

Console.WriteLine(" Point : " + temppoint.ToString() + " \n"); Console.WriteLine("----------------------------------\n");

Console.WriteLine();

Console.WriteLine("---------------Count Comp-------------------------\n");

Console.WriteLine(client.Common.CountComp().ToString());

Console.WriteLine("----------------------------------\n");

Console.WriteLine("MAX Node degree in main------------------------\n");

Console.WriteLine(client.Common.MaxNodeDegree());

}

private void button10\_Click(object sender, EventArgs e)

{

int temppoint = 0;

int N = 0; double R = 0; int power = 1000;

try

{

N = Convert.ToInt32(N\_box.Text);

R = Convert.ToDouble(R\_box.Text);

} catch

{

MessageBox.Show("Error");

}

client.Common.GenerateGraph(N, R, power); console\_rich.AppendText("Cгенерирован граф с характеристиками N = " +

N.ToString() + "\n R = " + R.ToString());

if (checkBox2.Checked)

{

console\_rich.AppendText("\n");

console\_rich.AppendText("Компоненты связности");

Methods.printMap(client.Common.Components(), console\_rich);

}

if (checkBox3.Checked)

{

console\_rich.AppendText("\n");

console\_rich.AppendText("Отсортированные компоненты связности"); Methods.printMap(client.Common.SortedComponents(), console\_rich);

}

if (checkBox8.Checked)

{

console\_rich.AppendText("\n");

console\_rich.AppendText("Количество компонент связности : "); console\_rich.AppendText(client.Common.CountComp().ToString());

}

if (checkBox4.Checked)

{

console\_rich.AppendText("\n");

console\_rich.AppendText("Главная компонента");

Methods.printMap(client.Common.MainComponent(), console\_rich);

}

if (checkBox5.Checked)

{

console\_rich.AppendText("\n");

console\_rich.AppendText("Количество узлов в главной компоненте : "); console\_rich.AppendText(client.Common.NumberNodesInMain().ToString());

}

if (checkBox6.Checked)

{

console\_rich.AppendText("\n");

console\_rich.AppendText("Количество ребер в главной компоненте : "); console\_rich.AppendText(client.Common.GetCountArcsInMain().ToString());

}

if (checkBox7.Checked)

{

console\_rich.AppendText("\n"); console\_rich.AppendText("Расстояние ближайщей точки гл.комп до границы области : ");

console\_rich.AppendText(client.Common.DistancePointToRoundInMain(out temppoint).ToString() + "\n");

console\_rich.AppendText(" Точка : " + temppoint.ToString());

}

if (checkBox9.Checked)

{

console\_rich.AppendText("\n"); console\_rich.AppendText("Максимальная степень вершины главной компоненты:");

console\_rich.AppendText(client.Common.MaxNodeDegree().ToString());

}

}

private void подключитьсяToolStripMenuItem\_Click(object sender, EventArgs e)

{

client = new UniservClient(); //client.Host = "localhost"; client.Host = "127.0.0.1"; client.Port = 2000; client.Events.OnError = OnError; client.Events.OnBark = OnBark;

toolStripStatusLabel1.Text = "Подключение... к " + client.Host + ":" + client.Port;

if (client.Connect(false))

{ toolStripStatusLabel1.Text = "Подключено к " + client.Host + ":" + client.Port;

}

}

private void button12\_Click(object sender, EventArgs e)

{

console\_rich.Clear();

} private void button13\_Click(object sender, EventArgs e)

{

checkBox2.Checked = true; checkBox3.Checked = true; checkBox4.Checked = true; checkBox5.Checked = true; checkBox6.Checked = true; checkBox7.Checked = true; checkBox8.Checked = true; checkBox9.Checked = true;

}

private void button11\_Click(object sender, EventArgs e)

{

N\_box.Clear(); R\_box.Clear(); checkBox2.Checked = false; checkBox3.Checked = false; checkBox4.Checked = false; checkBox5.Checked = false; checkBox6.Checked = false; checkBox7.Checked = false; checkBox8.Checked = false; checkBox9.Checked = false;

}

private void button14\_Click(object sender, EventArgs e)

{

Dictionary<int, Dictionary<double, double>> f = client.Common.GetAllPointsGraphExceptMain(); Dictionary<int, Dictionary<double, double>> main =

client.Common.GetAllPointsMain();

Dictionary<System.Windows.Point, List<System.Windows.Point>> arcs = client.Common.GetAllArcs();

Form2 a111 = new Form2(); a111.DrawGraph(f,main,arcs); a111.Show();

}

List<TextBox> list;

private void button15\_Click(object sender, EventArgs e)

{

int n = Convert.ToInt16(textBox1.Text);

list = new List<TextBox>();

List<Label> labellist = new List<Label>(); for (int i = 0; i < n; i++)

{

list.Add(new TextBox());

list[i].Text = (i+1).ToString();

list[i].Location = new System.Drawing.Point(10, (i \* 20)); this.panel1.Controls.Add(list[i]);

}

} private void radioN\_CheckedChanged(object sender, EventArgs e)

{

label5.Text = "R:";

} private void radioR\_CheckedChanged(object sender, EventArgs e)

{

label5.Text = "N:";

}

}

}

Form2.cs

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Text; using System.Threading.Tasks; using System.Windows.Forms; using ZedGraph;

namespace Ad\_hoc\_Client

{

public partial class Form2 : Form

{

public GraphPane pane;

public int xmin, xmax, ymin, ymax;

public Form2()

{

InitializeComponent();

}

public void DrawTestPanel(Color color, String xA, String Ya, Dictionary<double, double> graph)

{

// Создаем экземпляр класса MasterPane, который представляет собой область,

// на которйо "лежат" все графики (экземпляры класса GraphPane)

ZedGraph.MasterPane masterPane = zedGraph.MasterPane;

// По умолчанию в MasterPane содержится один экземпляр класса GraphPane // (который можно получить из свойства zedGraph.GraphPane)

// Очистим этот список, так как потом мы будем создавать графики вручную

masterPane.PaneList.Clear();

// Создаем экземпляр класса GraphPane, представляющий собой один график

GraphPane pane = new GraphPane();

DrawSingleGraph1(pane, color, xA, Ya, graph);

masterPane.Add(pane);

// Будем размещать добавленные графики в MasterPane using (Graphics g = CreateGraphics())

{

// Закомментарены разные варианты (не все) размещения графиков.

// Графики будут размещены в один столбец друг под другом

//masterPane.SetLayout (g, PaneLayout.SingleColumn);

//Графики будут размещены в одну строку друг за другом

//masterPane.SetLayout (g, PaneLayout.SingleRow);

// Графики будут размещены в две строки,

// в первой будет один столбец, а во второй - две masterPane.SetLayout(g, PaneLayout.SingleRow);

}

// Обновим оси и перерисуем график zedGraph.AxisChange(); zedGraph.Invalidate();

}

private string WhatChecked(string what)

{

if ("Количество узлов в графе" == what)

{

what = "Количество узлов в графе"; return "R";

}

if ("Радиус приема узла" == what)

{

what = "Радиус приема узла"; return "N";

}

return null;

}

private void DrawSingleGraph1(GraphPane pane, Color color, String xA, String Ya,

Dictionary<double, double> graph)

{

pane.CurveList.Clear();

// Включаем отображение сетки напротив мелких рисок по оси X pane.YAxis.MinorGrid.IsVisible = true;

// Задаем вид пунктирной линии для крупных рисок по оси Y:

// Длина штрихов равна одному пикселю, ... pane.YAxis.MinorGrid.DashOn = 1;

// затем 2 пикселя - пропуск pane.YAxis.MinorGrid.DashOff = 2;

// Включаем отображение сетки напротив мелких рисок по оси Y pane.XAxis.MinorGrid.IsVisible = true;

// Аналогично задаем вид пунктирной линии для крупных рисок по оси Y pane.XAxis.MinorGrid.DashOn = 1; pane.XAxis.MinorGrid.DashOff = 2;

//-----end netting----// pane.XAxis.Title.Text = xA;

pane.XAxis.Title.FontSpec.FontColor = Color.Blue;

// Изменим параметры шрифта для оси X

// pane.XAxis.Title.FontSpec.IsUnderline = true; // pane.XAxis.Title.FontSpec.IsBold = false; pane.YAxis.Title.FontSpec.FontColor = Color.Blue;

// Изменим текст по оси Y

// pane.YAxis.Title.Text = "Площадь многоугольника";

pane.YAxis.Title.Text = Ya;

// Изменим текст заголовка графика pane.Title.Text = "График";

// В параметрах шрифта сделаем заливку красным цветом pane.Title.FontSpec.Fill.Brush = new SolidBrush(Color.Red); pane.Title.FontSpec.Fill.IsVisible = true;

// Сделаем шрифт не полужирным

pane.Title.FontSpec.IsBold = false;

PointPairList list = new PointPairList();

// int sizedefault = CountsComps.Count;

// int size = CountsComps.Count;

if (Ya != "Расстояние от центра тяжести многоульника до центра области" & Ya !=

"Расстояние ближайшей точки гл.комп до границы области")

list.Add(0, 0);

// string e="";

foreach (KeyValuePair<double, double> entry in graph)

{

// do something with entry.Value or entry.Key list.Add(entry.Key, entry.Value);

// e = entry.Value.ToString();

}

LineItem myCurve = pane.AddCurve( WhatChecked(xA) + " = " + Form1.textBox3.Text, list, color, SymbolType.Plus);

this.Shape\_button.Hide();

}

public void DrawGraph(Dictionary<int, Dictionary<double, double>> allexceptmain, Dictionary<int, Dictionary<double, double>> main,

Dictionary<System.Windows.Point, List<System.Windows.Point>> arcs) {

DrawPanel();

DrawRound();

DrawNetting();

DrawNodes(allexceptmain, main);

DrawArcs(arcs);

DrawSquare();

}

public void DrawPanel()

{

// Получим панель для рисования this.pane = zedGraph.GraphPane; pane.CurveList.Clear(); zedGraph.AxisChange(); pane.GraphObjList.Clear(); xmin = -3; xmax = 3; ymin = -3; ymax = 3;

// Устанавливаем интересующий нас интервал по оси X

pane.XAxis.Scale.Min = xmin; pane.XAxis.Scale.Max = xmax;

// Устанавливаем интересующий нас интервал по оси Y pane.YAxis.Scale.Min = ymin; pane.YAxis.Scale.Max = ymax;

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

// В противном случае на рисунке будет показана только часть графика, // которая умещается в интервалы по осям, установленные по умолчанию zedGraph.AxisChange(); // Обновляем график zedGraph.Invalidate();

}

public void DrawRound()

{

// Очистим список кривых на тот случай, если до этого сигналы уже были нарисованы

pane.CurveList.Clear();

// Создадим список точек

PointPairList list = new PointPairList();

// Интервал, в котором будут лежать точки

for (double x = xmin; x <= xmax; x += 0.001)

{

list.Add(x, Methods.f(x)); list.Add(x, -Methods.f(x));

}

LineItem myCurve = pane.AddCurve("Scatter", list, Color.Blue, SymbolType.Diamond);

// !!!

// У кривой линия будет невидимой myCurve.Line.IsVisible = false;

// myCurve2.Line.IsVisible = false;

// !!!

// Цвет заполнения отметок (ромбиков) - колубой myCurve.Symbol.Fill.Color = Color.Blue;

//myCurve2.Symbol.Fill.Color = Color.Red;

// !!!

// Тип заполнения - сплошная заливка myCurve.Symbol.Fill.Type = FillType.Solid;

// myCurve2.Symbol.Fill.Type = FillType.Solid;

// !!!

// Размер ромбиков myCurve.Symbol.Size = 1; // myCurve2.Symbol.Size = 7;

}

public void DrawNetting()

{

// Включаем отображение сетки напротив крупных рисок по оси X pane.XAxis.MajorGrid.IsVisible = true;

// Задаем вид пунктирной линии для крупных рисок по оси X:

// Длина штрихов равна 10 пикселям, ...

pane.XAxis.MajorGrid.DashOn = 10;

// затем 5 пикселей - пропуск pane.XAxis.MajorGrid.DashOff = 5;

// Включаем отображение сетки напротив крупных рисок по оси Y pane.YAxis.MajorGrid.IsVisible = true;

// Аналогично задаем вид пунктирной линии для крупных рисок по оси Y pane.YAxis.MajorGrid.DashOn = 10; pane.YAxis.MajorGrid.DashOff = 5;

// Включаем отображение сетки напротив мелких рисок по оси X pane.YAxis.MinorGrid.IsVisible = true;

// Задаем вид пунктирной линии для крупных рисок по оси Y:

// Длина штрихов равна одному пикселю, ... pane.YAxis.MinorGrid.DashOn = 1;

// затем 2 пикселя - пропуск pane.YAxis.MinorGrid.DashOff = 2;

// Включаем отображение сетки напротив мелких рисок по оси Y pane.XAxis.MinorGrid.IsVisible = true;

// Аналогично задаем вид пунктирной линии для крупных рисок по оси Y pane.XAxis.MinorGrid.DashOn = 1; pane.XAxis.MinorGrid.DashOff = 2;

}

public void DrawSquare()

{

BoxObj box = new BoxObj(-0.05, 0.05, 0.1, 0.1, Color.Black, Color.Black); pane.GraphObjList.Add(box);

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

// В противном случае на рисунке будет показана только часть графика, // которая умещается в интервалы по осям, установленные по умолчанию zedGraph.AxisChange();

// Обновляем график zedGraph.Invalidate();

}

public void DrawNodes(Dictionary<int, Dictionary<double, double>> allexceptmain,

Dictionary<int, Dictionary<double, double>> main) {

PointPairList points = new PointPairList();

foreach (KeyValuePair<int, Dictionary<double, double>> kvp in allexceptmain) {

Dictionary<double, double> po = kvp.Value;

points.Add(po.Keys.First(), po.Values.First());

TextObj text2 = new TextObj(kvp.Key.ToString(), po.Keys.First(), po.Values.First());

text2.FontSpec.Border.Color = Color.Red; pane.GraphObjList.Add(text2);

}

foreach (KeyValuePair<int, Dictionary<double, double>> kvp in main) {

Dictionary<double, double> po = kvp.Value;

points.Add(po.Keys.First(), po.Values.First());

TextObj text2 = new TextObj(kvp.Key.ToString(), po.Keys.First(), po.Values.First());

text2.FontSpec.Border.Color = Color.Orange; pane.GraphObjList.Add(text2);

}

}

public void DrawArcs(Dictionary<System.Windows.Point, List<System.Windows.Point>> arcs)

{

foreach (KeyValuePair<System.Windows.Point, List<System.Windows.Point>> kvp in arcs)

{

List<System.Windows.Point> d = kvp.Value;

for (int i = 0; i < d.Count; i++)

{

ArrowObj arrow = new ArrowObj(kvp.Key.X, kvp.Key.Y, d[i].X, d[i].Y); arrow.IsArrowHead = false;

arrow.Line.Color = Color.MediumSeaGreen; arrow.Line.Width = 3;

// Добавим стрелку в список отображаемых объектов pane.GraphObjList.Add(arrow);

}

}

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

// В противном случае на рисунке будет показана только часть графика, // которая умещается в интервалы по осям, установленные по умолчанию zedGraph.AxisChange();

// Обновляем график zedGraph.Invalidate();

}

} }

Серверная часть.

Server.cs

#define USE\_COMPRESSION

using Interfaces; using System;

using System.Collections.Concurrent; using System.Collections.Generic; using System.IO; using System.IO.Compression; using System.Linq; using System.Net.Sockets; using System.Reflection;

using System.Runtime.Serialization.Formatters.Binary; using System.Threading; using System.Windows;

namespace ConsoleApplication1

{

public enum UserType

{

Unautorized,

}

public class UniservServer

{

public const int PING\_TIME = 7000; public System.Net.Sockets.TcpListener SERV;

static readonly SyncAccess ConnectedUsers = new SyncAccess();

#region Синхронный доступ к юзерам

private class SyncAccess

{

private List<User> userList = new List<User>(); private readonly object listLock = new object();

public void Add(User item)

{

lock (listLock)

{

userList.Add(item);

}

}

public bool Remove(User up)

{

lock (listLock)

{ up.Dispose();

return userList.Remove(up);

}

}

public User[] ToArray()

{

lock (listLock)

{

return userList.ToArray();

}

}

}

#endregion

#region Юзер

#region FIFO

public class ConqurentNetworkStream : IDisposable

{ private readonly BlockingCollection<byte[]> \_fifo = new

BlockingCollection<byte[]>();

private readonly NetworkStream \_nstream;

private readonly CancellationTokenSource \_token = new CancellationTokenSource(); private readonly ManualResetEventSlim \_disposeEvent = new

ManualResetEventSlim(false);

public ConqurentNetworkStream(NetworkStream nstream)

{

this.\_nstream = nstream;

ThreadPool.QueueUserWorkItem(\_thread);

}

private void \_thread(object state)

{ try {

while (true)

{ try {

byte[] data = \_fifo.Take(\_token.Token);

\_nstream.BeginWrite(data, 0, data.Length, null, null);

}

catch (InvalidOperationException)

{

continue;

}

} }

catch (Exception)

{

\_disposeEvent.Set(); return;

}

}

public void Add(byte[] data)

{

\_fifo.Add(data);

}

public int Read(byte[] data)

{

return \_nstream.Read(data, 0, data.Length);

}

public int EndRead(IAsyncResult asyncResult)

{

return \_nstream.EndRead(asyncResult);

}

public IAsyncResult BeginRead(byte[] data, AsyncCallback callback, object state)

{

return \_nstream.BeginRead(data, 0, data.Length, callback, state);

}

private readonly object \_disposeLock = new object(); private bool \_IsDisposed; public void Dispose()

{

lock (\_disposeLock)

{

if (!\_IsDisposed)

{

\_IsDisposed = true;

\_token.Cancel();

\_disposeEvent.Wait();

\_token.Dispose();

\_disposeEvent.Dispose();

\_nstream.Dispose();

}

}

}

}

#endregion

public class User : IDisposable

{

public User(TcpClient Socket)

{

this.\_socket = Socket;

Socket.ReceiveTimeout = PING\_TIME \* 4; Socket.SendTimeout = PING\_TIME \* 4;

nStream = new ConqurentNetworkStream(Socket.GetStream()); \_pingTimer = new Timer(OnPing, null, PING\_TIME, PING\_TIME);

ClassInstance = new Ring2(this);

}

private void OnPing(object state)

{

SendMessage(nStream, new Message("OnPing", null));

}

private readonly Timer \_pingTimer; public Type RingType { get; private set; } private Ring \_ClassInstance; public Ring ClassInstance

{

get { return \_ClassInstance; } set

{

\_ClassInstance = value;

RingType = \_ClassInstance.GetType(); } }

public UserType UserType = UserType.Unautorized;

public byte[] HeaderLength = BitConverter.GetBytes((int)0);

public readonly TcpClient \_socket;

public readonly ConqurentNetworkStream nStream; private readonly object \_disposeLock = new object(); private bool \_IsDisposed = false;

public void Dispose()

{

lock (\_disposeLock)

{

if (!\_IsDisposed)

{

\_IsDisposed = true; nStream.Dispose(); \_pingTimer.Dispose();

\_socket.Close();

}

}

}

}

#endregion

public UniservServer(int Port)

{

SERV = new System.Net.Sockets.TcpListener(System.Net.IPAddress.Any, Port);

Console.Title = string.Concat("Порт: ", Port);

}

public void Start()

{

SERV.Start();

SERV.BeginAcceptTcpClient(OnAcceptClient, null);

}

private void OnAcceptClient(IAsyncResult asyncResult)

{

var client = SERV.EndAcceptTcpClient(asyncResult);

SERV.BeginAcceptTcpClient(OnAcceptClient, null);

User up = new User(client);

ConnectedUsers.Add(up);

try {

up.nStream.BeginRead(up.HeaderLength, OnDataReadCallback, up);

}

catch (IOException)

{

ConnectedUsers.Remove(up);

}

}

private void OnDataReadCallback(IAsyncResult asyncResult) {

User up = (User)asyncResult.AsyncState; byte[] data;

try {

up.nStream.EndRead(asyncResult);

int dataLength = BitConverter.ToInt32(up.HeaderLength, 0); data = new byte[dataLength]; up.nStream.Read(data);

Message msg = MessageFromBinary<Message>(data);

ProcessMessage(msg, up);

up.nStream.BeginRead(up.HeaderLength, OnDataReadCallback, up); }

catch (Exception)

{

ConnectedUsers.Remove(up);

GC.Collect(2, GCCollectionMode.Optimized); return;

}

}

private void ProcessMessage(Message msg, User u)

{

string MethodName = msg.Command; if (MethodName == "OnPing") return;

// ищем запрошенный метод в кольце текущего уровня

MethodInfo method = u.RingType.GetMethod(MethodName,

BindingFlags.Instance | BindingFlags.Public | BindingFlags.FlattenHierarchy);

try {

if (method == null)

{

Console.WriteLine(string.Concat(u.UserType.ToString(), " -> ",

MethodName, "(", string.Join(", ", msg.prms), ")"));

// Console.WriteLine(string.Concat(.ToString(), " -> ", MethodName, "(", string.Join(", ", msg.prms), ")")); throw new Exception(string.Concat("Метод \"", MethodName, "\" недоступен"));

}

try

{

// выполняем метод интерфейса

msg.ReturnValue = method.Invoke(u.ClassInstance, msg.prms);

}

catch (Exception ex)

{

throw ex.InnerException;

}

// Console.WriteLine(up.\_socket.Client.RemoteEndPoint.ToString());

Console.WriteLine(string.Concat(u.\_socket.Client.RemoteEndPoint.ToString(),

" -> ", MethodName, "(", string.Join(", ", msg.prms), ")"));

// возвращаем ref и out параметры msg.prms = method.GetParameters().Select(x =>

x.ParameterType.IsByRef ? msg.prms[x.Position] : null).ToArray(); }

catch (Exception ex)

{

msg.Exception = ex;

} finally

{

// возвращаем результат выполнения запроса

SendMessage(u.nStream, msg);

}

}

public class Ring2 : Ring, ICommon

{

Graph gr; Test d; int temppoint = 0;

public Ring2(User u) : base(u)

{

}

public void GenerateGraph(int count,double r,int power)

{

gr = new Graph(); gr.Radius = r;

gr.UniformDistribution(power, count); gr.CheckArc();

}

public int MaxNodeDegree()

{

return gr.MaxNodeDegree();

}

public int CountComp()

{

return gr.CheckCountComp();

}

public Dictionary<int, int[]> Components()

{

gr.FindComps(); gr.SortComp(); return gr.Map;

}

public Dictionary<int, int[]> SortedComponents()

{

return gr.SortMap;

}

public Dictionary<int,int[]> MainComponent()

{

return gr.MainComp;

}

public int NumberNodesInMain()

{

return gr.GetCountNodesInMain();

}

public int GetCountArcsInMain()

{

return gr.GetCountArcsInMain();

}

public double DistancePointToRoundInMain(out int temppoint)

{

return gr.DistancePointToRoundInMain(out temppoint); }

public int GetPointToRoundInMain()

{

return temppoint;

}

public void TestMode(int CountTestCycle,int n,double r,string c)

{

if(d == null)

{

d = new Test();

}

d.InitGraphTestMode(CountTestCycle, n, r,c);

}

public Dictionary<double, double> GetGraph1()

{ return d.firstGraph;

}

public Dictionary<double, double> GetGraph2()

{

return d.twoGraph;

}

public Dictionary<double, double> GetGraph3()

{

return d.threeGraph;

}

public Dictionary<double, double> GetGraph4()

{

return d.fourGraph;

}

public Dictionary<double, double> GetGraph5()

{

return d.fiveGraph;

}

public Dictionary<double, double> GetGraph6()

{

return d.sixGraph;

}

public Dictionary<double, double> GetGraph7()

{

return d.seven;

}

public Dictionary<double, double> GetGraph8()

{

return d.eight;

}

public Dictionary<int, Dictionary<double, double>> GetAllPointsMain()

{

return gr.GetAllPointsMain();

}

public Dictionary<int, Dictionary<double, double>> GetAllPointsGraphExceptMain()

{

return gr.GetAllPointsGraphExceptMain(); }

public Dictionary<Point, List<Point>> GetAllArcs()

{

return gr.GetAllArcs();

}

}

public abstract class Ring

{

public readonly User up;

public Ring(User up)

{

this.up = up;

}

}

#region Send/Receive

private T MessageFromBinary<T>(byte[] BinaryData) where T : class

{

#if USE\_COMPRESSION

using (MemoryStream memory = new MemoryStream(BinaryData))

{ using (var gZipStream = new GZipStream(memory, CompressionMode.Decompress, false))

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); return (T)binaryFormatter.Deserialize(gZipStream);

}

} #else

using (MemoryStream memory = new MemoryStream(BinaryData))

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); return (T)binaryFormatter.Deserialize(memory);

}

#endif

}

private static void SendMessage(ConqurentNetworkStream nStream, Message msg) {

#if USE\_COMPRESSION

using (MemoryStream memory = new MemoryStream())

{ using (var gZipStream = new GZipStream(memory, CompressionMode.Compress, false))

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); binaryFormatter.Serialize(gZipStream, msg);

}

byte[] BinaryData = memory.ToArray();

byte[] DataLength = BitConverter.GetBytes(BinaryData.Length); byte[] DataWithHeader = DataLength.Concat(BinaryData).ToArray();

nStream.Add(DataWithHeader);

}

#else

using (MemoryStream memory = new MemoryStream())

{

BinaryFormatter binaryFormatter = new BinaryFormatter(); binaryFormatter.Serialize(memory, msg); nStream.Add(memory.ToArray());

}

#endif

}

#endregion

} }

Test.cs

class Test {

public Stack<double> r1, r2, r3, CountDistance; public Stack<double> Time1, Time2, TIME11, TIME22;

public Dictionary<double, double> firstGraph = new Dictionary<double, double>(); public Dictionary<double, double> twoGraph = new Dictionary<double, double>(); public Dictionary<double, double> threeGraph = new Dictionary<double, double>(); public Dictionary<double, double> fourGraph = new Dictionary<double, double>(); public Dictionary<double, double> fiveGraph = new Dictionary<double, double>(); public Dictionary<double, double> sixGraph = new Dictionary<double, double>(); public Dictionary<double, double> seven = new Dictionary<double, double>(); public Dictionary<double, double> eight = new Dictionary<double, double>(); public Dictionary<int, List<int>> threePoints; public Dictionary<int, int> twoPoints; public Dictionary<int, int> Pointlist; private Graph GraphTest;

public void InitGraphTestMode(int countTestCycle, int n, double r,string c)

{

int power = 1000;

double memorizeMaxDegree = 0; double memorizeCenter\_mass = 0; double memorizeCountArcsMain = 0; double middleCountComp = 0; double middleCountCompMain = 0; double middleCountArcsMain = 0; double memorizeDistance = 0; double middleAreaPolygon = 0;

int i = 1;

while (i <= countTestCycle)

{

// File.AppendAllText("stats.txt", i.ToString() + ". ");

Testing(power, n, r, ref memorizeMaxDegree, ref memorizeCenter\_mass, ref memorizeCountArcsMain,

ref middleCountComp,ref middleCountCompMain,ref middleCountArcsMain, ref memorizeDistance,ref middleAreaPolygon); i++;

}

memorizeMaxDegree /= countTestCycle; memorizeCenter\_mass /= countTestCycle; memorizeCountArcsMain /= countTestCycle; middleCountComp /= countTestCycle;

middleCountCompMain /= countTestCycle;

middleCountArcsMain /= countTestCycle; memorizeDistance /= countTestCycle; middleAreaPolygon /= countTestCycle;

if (c == "n")

{

this.firstGraph.Add(n, Math.Round(memorizeMaxDegree)); this.twoGraph.Add(n, memorizeCenter\_mass);

this.threeGraph.Add(n, Math.Round(memorizeCountArcsMain)); this.fourGraph.Add(n, Math.Round(middleCountComp)); this.fiveGraph.Add(n, Math.Round(middleCountCompMain)); this.sixGraph.Add(n, Math.Round(middleCountArcsMain)); this.seven.Add(n, memorizeDistance); this.eight.Add(n, middleAreaPolygon);

}

if (c == "r")

{

this.firstGraph.Add(r, Math.Round(memorizeMaxDegree)); this.twoGraph.Add(r, memorizeCenter\_mass);

this.threeGraph.Add(r, Math.Round(memorizeCountArcsMain)); this.fourGraph.Add(r, Math.Round(middleCountComp)); this.fiveGraph.Add(r, Math.Round(middleCountCompMain)); this.sixGraph.Add(r, Math.Round(middleCountArcsMain)); this.seven.Add(r, memorizeDistance); this.eight.Add(r, middleAreaPolygon);

}

} public void Testing(int power, int countNodes, double r, ref double middleMaxDegree, ref double middleCentreMass,

ref double middleCountAngles,ref double middleCountComp,ref double

middleCountCompMain,ref double middleCountArcsMain,

ref double middleDistance,ref double middleAreaPolygon)

{

GraphTest = new Graph(); int temppoint = 0; GraphTest.Radius = r;

GraphTest.UniformDistribution(power, countNodes);

GraphTest.CheckArc();

GraphTest.FindComps();

GraphTest.SortComp();

middleMaxDegree += GraphTest.MaxNodeDegree(); middleCentreMass += Algorithms.center\_mass(this.GraphTest); middleCountAngles += GraphTest.CountAngles; middleCountComp += GraphTest.CheckCountComp(); middleCountCompMain += GraphTest.GetCountNodesInMain(); middleCountArcsMain += GraphTest.GetCountArcsInMain();

middleDistance += GraphTest.DistancePointToRoundInMain(out temppoint); middleAreaPolygon += Algorithms.area\_of\_polygon(this.GraphTest);

}

}

Graph.cs

class Graph {

public Graph()

{

nodes = null; Roundradius = 1;

map = new Dictionary<int, int[]>(); sortmap = new Dictionary<int, int[]>(); maincomp = new Dictionary<int, int[]>();

}

public Node Nodes

{ get {

return this.nodes;

}

}

#region

public int MaxNodeDegree()

{

Node p = this.nodes; int max = 0;

while (p != null)

{

if (p.Componenta == this.maincomp.Keys.FirstOrDefault())

{

Arc a = p.Arclist; int pre\_max = 0; while (a != null)

{ pre\_max++; a = a.Next;

}

if (pre\_max > max) { max = pre\_max; }

}

p = p.Next;

}

return max;

}

public double D(double x1, double y1, double x2, double y2)

{

return Math.Sqrt(Math.Pow((x2 - x1), 2) + Math.Pow((y2 - y1), 2));

}

public Dictionary<int, Dictionary<double, double>> GetAllPointsMain() {

Dictionary<int, Dictionary<double, double>> r = new Dictionary<int,

Dictionary<double, double>>();

int p = 0;

while (p != CheckCountComp())

{ p++;

Node temp = this.nodes;

while (temp != null)

{

if (p == mainNumberComp)

if ((p != 0) & (temp.Componenta != 0))

{

Dictionary<double, double> f = new Dictionary<double, double>();

f.Add(temp.X, temp.Y);

r.Add(temp.Id, f);

}

temp = temp.Next;

}

}

return r;

}

public Dictionary<int, Dictionary<double, double>> GetAllPointsGraphExceptMain() {

Dictionary<int, Dictionary<double, double>> r = new Dictionary<int,

Dictionary<double, double>>();

int p = 0;

while (p != CheckCountComp())

{ p++;

Node temp = this.nodes;

while (temp != null)

{

if ((p == temp.Componenta) & !((p == mainNumberComp))) if ((p != 0) & (temp.Componenta != 0))

{

Dictionary<double, double> f = new Dictionary<double, double>();

f.Add(temp.X, temp.Y);

r.Add(temp.Id, f);

temp = temp.Next;

}

}

return r;

}

Dictionary<Point, List<Point>> p;

public void additemlist(Point a,Point b)

{

if(p.ContainsKey(a))

{

p[a].Add(b);

}else {

p[a] = new List<Point>(); p[a].Add(b);

}

}

public Dictionary<Point, List<Point>> GetAllArcs()

{

Node p1 = this.nodes;

p = new Dictionary<Point, List<Point>>();

while (p1 != null)

{

Arc a1 = p1.Arclist;

while (a1 != null)

{

Point a = new Point();

a.X = p1.X;

a.Y = p1.Y;

Point b = new Point();

b.X = a1.Adj.X;

b.Y = a1.Adj.Y;

additemlist(a, b);

a1 = a1.Next;

}

p1 = p1.Next;

}

return p;

}

public void UniformDistribution(int N, int count) {

Random rd1 = new Random();

double x, y, r;

//int count1 = 0, count2 = 0, count3 = 0; int count3 = 0;

for (int i = 1; i <= N; i++)

{

r = this.Roundradius \* rd1.NextDouble();

// x = this.Roundradius \* Math.Sqrt(rd1.NextDouble()); x = Math.Sqrt(r);

y = 2 \* Math.PI \* rd1.NextDouble();

if ((rd1.Next() / rd1.Next() > 0) & (count3 <= count))

{ count3++;

this.AddNode(count3, 0, (x \* Math.Sin(y)), (x \* Math.Cos(y)));

}

if (count == count3) break;

// if ((x \* Math.Sin(y) > 0.99)) count1++; //points in round

// if ((x \* Math.Cos(y)) > 0.99) count2++;

} }

public void CheckArc()

{

Node p = this.nodes;

Node p1;

while (p != null)

{

p1 = this.nodes.Next;

if (p1.Id == p.Id) p1 = p1.Next;

while (p1 != null)

{ if ((D(p.X, p.Y, p1.X, p1.Y) < this.radiusNode) & (!FindArc(p, p1)) &

(!FindArc(p1, p)))

{

if (!(p1.Id == p.Id)) {

AddArc(p, p1, 0);

AddArc(p1, p, 0);

}

}

p1 = p1.Next;

}

p = p.Next;

} }

public int CheckCountNodesInComp(int i)

{

Node p = this.nodes; int count = 0;

while (p != null)

{

if (p.Componenta == i)

{

count++;

}

p = p.Next;

}

return count;

}

public int WhatComp(int value, bool[] pos) {

int position = 0;

while (position <= CheckCountComp())

{

if ((CheckCountNodesInComp(position) == value) & (!pos[position]))

{

pos[position] = true; return position;

}

position++;

}

return position;

}

public int CheckCountComp()

{

int kol = 0;

for (Node p = nodes; p != null; p = p.Next) if (!p.Mark)

{ kol++; glubina(p, kol);

}

for (Node p = nodes; p != null; p = p.Next) if (p.Mark)

{

p.Mark = false; } return kol;

}

public void glubina(Node nod, int kol)

{

nod.Mark = true; nod.Componenta = kol;

Arc a = nod.Arclist; while (a != null) if (a.Adj.Mark == false) glubina(a.Adj, kol); else

a = a.Next;

}

public void SortComp()

{

int[] forSortingValues = new int[CheckCountComp() + 1];

for (int i = 1; i <= CheckCountComp(); i++)

{

forSortingValues[i] = this.CheckCountNodesInComp(i);

}

QuickSort(forSortingValues, forSortingValues.Length); int[] klo; int loc;

bool[] pos = new bool[CheckCountComp() + 1];

//Основная сортировка компонент связности for (int i = 1; i <= CheckCountComp(); i++)

{

// Console.WriteLine(forSortingValues[i].ToString());

loc = WhatComp(forSortingValues[i], pos); map.TryGetValue(loc, out klo);

sortmap.Add(loc, klo);

if (i == CheckCountComp())

{

maincomp.Add(loc, klo); mainNumberComp = loc;

}

}

}

public void FindComps()

{

Node p = this.nodes;

for (int i = 1; i <= CheckCountComp(); i++)

{

int[] mas = new int[this.CheckCountNodesInComp(i)];

int v = 0; p = this.nodes;

while (p != null)

{

if (p.Componenta == i)

{ mas[v] = p.Id; v++;

}

p = p.Next;

}

map.Add(i, mas);

}

}

public void AddNode(int id, int inf, double x, double y)

{

Node TempNode = new Node(id, inf, x, y);

TempNode.Next = nodes; nodes = TempNode;

}

public Node FindNode(int id) //поиск вершины графа

{

Node s = nodes;

//Node v = null;

while (s != null)

{

if (s.Id == id) //v = s; return s;

// Console.WriteLine(s.Id.ToString());

s = s.Next; //s = s->next;

} return null;

}

public void AddArc(Node u, Node v, int x)

{

if ((u == null) || (v == null))

Console.WriteLine("There are no node!"); else

{

Arc nod = new Arc();

nod.Adj = v; nod.Infarc = x; nod.Next = u.Arclist;

u.Arclist = nod;

} }

public bool FindArc(Node u, Node v) //поиск дуги графа

{

if ((u == null) || (v == null)) return false;

Arc nod = FindNode(u.Id).Arclist; while (nod != null)

{

if (nod.Adj.Id == v.Id) return true; nod = nod.Next;

}

return false;

}

public void printMap(Dictionary<int, int[]> tempmap)

{

foreach (KeyValuePair<int, int[]> kvp in tempmap)

{

Console.Write("Сomponent = {0} :", kvp.Key);

for (int i = 0; i < kvp.Value.Count(); i++)

{

Console.Write(" " + kvp.Value[i].ToString());

}

Console.WriteLine();

} }

public void print()

{

Node p = this.nodes;

Console.WriteLine(" vershiny : \n"); while (p != null)

{

Console.WriteLine(p.Id.ToString()); p = p.Next;

}

Console.WriteLine(" rebra : \n"); p = this.nodes; while (p != null)

{

Arc a = p.Arclist; while (a != null)

{

Console.WriteLine(p.Id.ToString() + " " + a.Adj.Id.ToString() + " \n"); a = a.Next;

}

p = p.Next;

}

}

public void QuickSort(int[] array, int a, int b) // фукнция сортировки

{ int A = a; int B = b; double mid;

if (b > a)

{

// Находим разделительный элемент в середине массива mid = array[(a + b) / 2];

// Обходим массив while (A <= B)

{

/\* Находим элемент, который больше или равен \* разделительному элементу от левого индекса.

\*/

while ((A < b) && (array[A] < mid)) ++A;

/\* Находим элемент, который меньше или равен \* разделительному элементу от правого индекса.

\*/

while ((B > a) && (array[B] > mid)) --B;

// Если индексы не пересекаются, меняем if (A <= B)

{ int T; T = array[A]; array[A] = array[B]; array[B] = T;

++A;

--B;

}

}

/\* Если правый индекс не достиг левой границы массива,

* нужно повторить сортировку левой части.

\*/

if (a < B) QuickSort(array, a, B);

/\* Если левый индекс не достиг правой границы массива,

* нужно повторить сортировку правой части.

\*/

if (A < b) QuickSort(array, A, b);

} }

public void QuickSort(int[] array, int n) {

QuickSort(array, 0, n - 1);

}

public double DistancePointToRoundInMain(out int point)

{

int[] array = new int[GetCountNodesInMain() + 1]; array = maincomp.FirstOrDefault().Value;

double min = this.Roundradius - D(FindNode(array[0]).X,

FindNode(array[0]).Y, 0, 0);

point = array[0];

// Console.WriteLine(array[0].ToString() + " distance: " + min.ToString()); for (int i = 1; i < GetCountNodesInMain(); i++)

{ if (min > (this.Roundradius - D(FindNode(array[i]).X, FindNode(array[i]).Y, 0, 0)))

{ min = this.Roundradius - D(FindNode(array[i]).X,

FindNode(array[i]).Y, 0, 0);

point = array[i];

}

// Console.WriteLine(array[i].ToString() + " distance: " +

(this.Roundradius - D(FindNode(array[i]).X, FindNode(array[i]).Y, 0,

0)).ToString());

}

return min;

}

public int GetCountNodesInMain()

{

return maincomp.FirstOrDefault().Value.Length;

}

public int GetCountArcsInMain()

{

Node p1 = this.nodes; int counter = 0;

while (p1 != null)

{

Arc a1 = p1.Arclist; while (a1 != null)

{

if (p1.Componenta == this.mainNumberComp) counter++;

a1 = a1.Next;

}

p1 = p1.Next;

}

return counter / 2;

}

public int Get\_node\_id\_by\_XY(double x, double y) {

Node p = this.nodes;

while (p != null)

{

if ((p.X == x) && (p.Y == y)) return p.Id;

p = p.Next;

}

return -1;

}

public double Radius

{ get {

return this.radiusNode;

} set {

this.radiusNode = value;

} }

public Dictionary<int, int[]> Map

{ get {

return this.map;

} }

public Dictionary<int, int[]> SortMap

{ get {

return this.sortmap;

} }

public Dictionary<int, int[]> MainComp

{ get {

return this.maincomp;

}

}

#endregion

private Node nodes; private double radiusNode; private double Roundradius; public int mainNumberComp; public int CountAngles; private Dictionary<int, int[]> map; private Dictionary<int, int[]> sortmap; private Dictionary<int, int[]> maincomp; public List<int> Pointlist;

} class Node {

public Node(int i, int infnode, double x, double y)

{ this.id = i; this.arclist = null; this.mark = false; this.componenta = 0; this.x = x; this.y = y;

}

#region

public int Componenta { get {

return this.componenta;

} set {

this.componenta = value;

} } public bool Mark { get { return this.mark; } set { this.mark = value;

} } public int Infnode { get {

return this.infnode;

} set {

this.infnode = value;

}

}

public Node Next { get { return this.next; } set { this.next = value;

}

} public int Id { get { return this.id; } set { this.id = value;

}

}

public Arc Arclist { get {

return this.arclist; }

set {

this.arclist = value;

} } public double X { get { return this.x;

}

}

public double Y { get { return this.y;

}

}

#endregion private Node next; private Arc arclist; private int infnode; private int id; private bool mark; private int componenta; private double x, y;

} class Arc { public Arc() { adj = null; } #region public Node Adj { set { this.adj = value; } get { return this.adj;

}

}

public int Infarc { set {

this.infarc = value;

}

}

public Arc Next { set { this.next = value;

} get {

return this.next;

}

} #endregion private Arc next; private Node adj; private int infarc;

}

}

Algorithms.cs

class Algorithms

{ public static void ShapePerform(Graph gr,ref Dictionary<int, List<int>> theePoints, ref Dictionary<int, int> twoPoints, ref Dictionary<int, int> Pointlist)

{

int[] array = gr.MainComp.First().Value;

Shape q = new Shape(array.Length);

int count = array.Length;

for (int i = 0; i < array.Length; i++)

{

q.X[i] = gr.FindNode(array[i]).X;

q.Y[i] = gr.FindNode(array[i]).Y;

q.PT[i] = i;

}

q.Start();

theePoints = q.TreePoints; twoPoints = q.TwoPoints;

gr.CountAngles = 0;

gr.Pointlist = new List<int>();

// Console.WriteLine("Points of the convex hull:\n"); for (int j = 0; j < count; j++) if (q.Belongs\_to\_hull[j] == 1)

{

Pointlist.Add(j, gr.Get\_node\_id\_by\_XY(q.X[j], q.Y[j])); gr.CountAngles++;

}

}

class Shape { int N = 0; public Shape(int n)

{

this.x = new double[n]; this.y = new double[n]; this.pt = new int[n]; this.belongs\_to\_hull = new int[n]; N = n;

threePoints = new Dictionary<int, List<int>>(); twoPoints = new Dictionary<int, int>();

}

Dictionary<int, List<int>> threePoints; Dictionary<int, int> twoPoints; double[] x;

double[] y; /\* x and y point coordinates \*/

int[] pt; int[] belongs\_to\_hull;

public Dictionary<int, List<int>> TreePoints

{ get {

return threePoints;

}

}

public Dictionary<int, int> TwoPoints

{ get {

return twoPoints;

} }

public double[] X

{ get { return x;

}

}

public double[] Y

{ get { return y;

}

}

public int[] Belongs\_to\_hull

{ get {

return belongs\_to\_hull;

}

}

public int[] PT

{ get {

return pt;

}

}

/\* output routines: \*/ public void Start()

{

int uppercnt, lowercnt; int j;

int minxpt = 0, maxxpt = 0; double xj;

int[] upper = new int[N]; int[] lower = new int[N];

for (j = 0; j < N; j++) belongs\_to\_hull[j] = 0;

inithull(ref pt, N, ref minxpt, ref maxxpt);

uppercnt = lowercnt = N;

qh(delete\_right(pt, ref uppercnt, ref minxpt, ref maxxpt), uppercnt); qh(delete\_right(pt, ref lowercnt, ref maxxpt, ref minxpt), lowercnt);

}

public void print\_array(int[] pt, int n)

{

long j;

Console.WriteLine("Array with %d points:\n" + n.ToString()); for (j = 0; j < n; j++)

Console.WriteLine("point %d = (%d,%d)\n" + pt[j].ToString() +

" " + x[pt[j]].ToString() + " " + y[pt[j]].ToString());

Console.WriteLine("\n");

}

public void print\_hull()

{ int j;

Console.WriteLine("Points of the convex hull:\n"); for (j = 0; j < N; j++) if (belongs\_to\_hull[j] == 1)

Console.WriteLine(j.ToString() + " " + x[j].ToString() +

" " + y[j].ToString());

}

public double cross(int p, int a, int b)

{

return ((x[a] - x[p]) \* (y[b] - y[p]) - (y[a] - y[p]) \* (x[b] - x[p]));

}

public bool leftturn(int a, int b, int c)

{

return (cross(c, a, b) > 0.0);

}

public int[] delete\_right(int[] pt, ref int num, ref int p1, ref int p2)

{ int j; int leftcnt; int[] left = new int[N]; int n = num;

/\* delete all points in pt located right from line p1p2: \*/

left[0] = p1; left[1] = p2; leftcnt = 2; for (j = 0; j < num; j++)

if (!(pt[j] == p1 || pt[j] == p2)) /\*p1 and p2 already in left[]\*/ if (leftturn(p1, p2, pt[j])) /\* point j is lefthand to vector p1p2 left[leftcnt++] = pt[j];

num = leftcnt;

return left;

}

public void inithull(ref int[] pt, int n, ref int minx, ref int maxx)

{

int p1, p2; int i;

/\* determine points p1,p2 with minimal and maximal x coordinate: \*/ p1 = p2 = pt[0]; /\* init. p1,p2 to first point \*/ for (i = 1; i < n; i++)

{ /\* seq. search for minimum / maximum \*/ if (x[pt[i]] < x[p1]) p1 = i; if (x[pt[i]] > x[p2]) p2 = i;

}

belongs\_to\_hull[p1] = 1; belongs\_to\_hull[p2] = 1; minx = p1; maxx = p2;

}

public int pivotize(ref int[] pt, int n)

{

int i, p1 = pt[0], p2 = pt[1]; int pivotpos = 2;

double maxcross = cross(pt[2], p1, p2); for (i = 3; i < n; i++)

{ /\* sequential maximization \*/ double newcross = cross(pt[i], p1, p2);

if (newcross > maxcross)

{

maxcross = newcross; pivotpos = i;

}

}

return pt[pivotpos];

}

public void qh(int[] pt, int n)

{

/\* DC step: select pivot point from pt \*/ int pivotpos, pivot; int p1 = pt[0], p2 = pt[1]; int[] left1, left2; int leftcnt1, leftcnt2;

/\* DC step: select any pivot point from pt. We have p1==pt[0],p2==pt[1] \*/ if (n == 2)

{

this.twoPoints.Add(pt[0], pt[1]); return; }

if (n == 3)

{

List<int> temp = new List<int>();

temp.Add(pt[0]); temp.Add(pt[1]);

// this.threePoints.Add(pt[2]+1, temp);

this.threePoints.Add(pt[2], temp);

/\* one point (beyond old p1,p2) must belong to hull \*/ belongs\_to\_hull[pt[2]] = 1; /\* saves a recursive call \*/ return;

}

pivot = pivotize(ref pt, n);

belongs\_to\_hull[pivot] = 1; leftcnt1 = n;

left1 = delete\_right(pt, ref leftcnt1, ref p1, ref pivot); qh(left1, leftcnt1); leftcnt2 = n;

left2 = delete\_right(pt, ref leftcnt2, ref pivot, ref p2); qh(left2, leftcnt2);

} }

public class Center\_mass

{

private double[] x; private double[] y; private int n;

public Center\_mass(int N)

{ this.n = N; this.x = new double[N]; this.y = new double[N];

}

// возвращает длину отрезка с координатами (x1,y1)-(x2,y2) double length(double x1, double y1, double x2, double y2)

{

return Math.Sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

}

public Point start()

{

// Console.WriteLine("Incoming points : "); for (int j = 0; j < n; j++)

{ }

double xc = 0, yc = 0; double P = 0;

for (int i = 0; i < n; i++)

{

// применяем формулы (\*)

double l = length(x[i], y[i], x[(i + 1) % n], y[(i + 1) % n]); xc += l \* (x[i] + x[(i + 1) % n]) / 2; yc += l \* (y[i] + y[(i + 1) % n]) / 2;

P += l; } xc /= P; yc /= P;

//printf("%lf %lf\n",xc,yc);

// Console.Write("Center mass (point) : ");

// Console.Write((xc).ToString() + " " + (yc).ToString() + "\n");

//}

Point k = new Point();

k.X = Convert.ToSingle(xc);

k.Y = Convert.ToSingle(yc);

return k;

}

public double[] X

{ get {

return this.x;

}

}

public double[] Y

{ get {

return this.y;

}

} }

public static double area\_of\_polygon(Graph GraphTest)

{

Dictionary<int, List<int>> threePoints = new Dictionary<int, List<int>>();

Dictionary<int, int> twoPoints = new Dictionary<int, int>(); Dictionary<int, int> Pointlist = new Dictionary<int, int>();

Algorithms.ShapePerform(GraphTest, ref threePoints, ref twoPoints, ref Pointlist);

Dictionary<int, double> PointlistTan = new Dictionary<int, double>();

foreach (KeyValuePair<int, int> pair in Pointlist) {

Node a1 = GraphTest.FindNode(pair.Value);

PointlistTan.Add(pair.Value, Math.Atan2(a1.X, a1.Y));

}

return S(PointlistTan,GraphTest);

}

public static double S(Dictionary<int, double> sortedDict,Graph GraphTest)

{

double Xp, Yp, S;

double[] X = new double[100]; double[] Y = new double[100]; double[] L = new double[100]; double[] C = new double[100];

int N = sortedDict.Count; int j = 0;

foreach (KeyValuePair<int, double> pair in sortedDict.OrderBy(pair =>

pair.Value))

{

Node a1 = GraphTest.FindNode(pair.Key);

X[j] = a1.X; Y[j] = a1.Y; j++;

}

for (int i = 0; i < N - 1; i++)

{

L[i] = Math.Sqrt(Math.Pow((X[i] - X[i + 1]), 2) +

Math.Pow((Y[i] - Y[i + 1]), 2));

}

L[N - 1] = Math.Sqrt(Math.Pow((X[N - 1] - X[0]), 2) + Math.Pow((Y[N - 1] - Y[0]), 2));

if (N == 3)

{ S = Geron(L[0], L[1], L[2]);

} else

{

Xp = (X[0] + X[2]) / 2;

Yp = (Y[0] + Y[2]) / 2; S = 0;

for (int i = 0; i < N; i++)

C[i] = Math.Sqrt(Math.Pow((X[i] - Xp), 2) + Math.Pow((Y[i] - Yp), 2)); for (int i = 0; i < N - 1; i++) S = S + Geron(C[i], L[i], C[i + 1]);

S = S + Geron(C[N - 1], L[N - 1], C[0]);

}

//Console.WriteLine("S= " + S.ToString()); return S;

}

public static double Geron(double a, double b, double c)

{

double p;

p = (a + b + c) / 2;

return Math.Sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

} }