ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

на тему Планування топології локальних мереж на основі розв'язання задачі Штейнера
Local network topology planning based on the solution of the Steiner problem

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Зінгер В'ячеслав Геннадійович
(прізвище, ім'я, по-батькові)
Керівник к.ф.-м.н., доц. Шпінарева І.М.
(науковий ступінь, вчене звання, прізвище та ініціали, підпис)
Рецензент Розновець О.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№ 12 від «9» червня 2023 р.

Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___
протокол № __ від «__» ___ 2023р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ Алла КОБОЗЄВА
(підпис) (ім'я, прізвище)

Одеса – 2023

АНОТАЦІЯ

Метою роботи є розробка застосунку для побудови топології мережі мінімальної протяжності на основі розв'язання задачі Штейнера для покращення надійності працездатності мережі.

Для планування топології локальних мереж обрано задачу Штейнера в евклідовому просторі. Запропоновано її розв’язання на основі алгоритму Мелзака.

Розроблено програмний застосунок дозволяє знаходити розв'язання задачі Штейнера на площині для невеликої кількості точок. Як мову програмування обрано мову C#, що дозволяє створити зручний інтерфейс, а також візуалізувати дерево Штейнера, яке є результатом побудови. Довжина дерева Штейнера порівнюється з довжиною остовного дерева, яке обчислюється алгоритмом Прима.

Варто визнати, що алгоритм Мелзака є далеко не оптимальним, про що свідчать результати роботи програми для різної кількості точок. Алгоритм чудово справляється з кількістю точок $n \leq 9$. На побудову дерева Штейнера 6 точок, алгоритму необхідно менше 1 секунди, а для 8 точок час роботи алгоритму вже перевищує 9 хвилин.

Розроблений програмний застосунок дозволяє проектувати реальні мережі з невеликою кількістю вузлів з найкоротшим шляхом, що покращує їх працездатність і збільшує функціональні можливості протоколів маршрутизації.

ABSTRACT

The purpose of the work is to develop an application for building a minimum length network topology based on the solution of the Steiner problem to improve the reliability of the network.

The Steiner problem in Euclidean space was chosen for planning the topology of local networks. It is proposed to solve it based on Melzack's algorithm.

A software application has been developed that allows you to find a solution to the Steiner problem on the plane for a small number of points. The C# language was chosen as the programming language, which allows you to create a convenient interface, as well as visualize the Steiner tree, which is the result of construction.

It is worth admitting that Melzack's algorithm is far from optimal, as evidenced by the results of the program for different numbers of points. The algorithm does a great job with the $n \leq 9$ number of points. The algorithm needs less than 1 second to build a Steiner tree for tasks consisting of 6 points, but for 8 points, the algorithm takes more than 9 minutes.

The developed software application allows you to design real networks with a small number of nodes, with the shortest path, which improves their performance and increases the functionality of routing protocol.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

BGP-4 – Border Gateway Protocol – протокол граничного шлюзу, призначений для обміну інформацією про маршрутизацію та доступність між автономними системами в Інтернеті.

CDN – Content Delivery Network або мережа доставки контенту – група зв'язаних між собою серверів, завдяки яким здійснюється швидка доставка інтернет-контенту.

GGC – Google Global Cache – програмно-апаратний комплекс компанії Google, призначений для оптимізації навантаження на потужності Інтернет-провайдера під час роботи з сервісами Google (один із елементів Content Delivery Network).

HGG – Hanan grid graph – кінцевої множини точок на площині виходить проведенням вертикальних і горизонтальних ліній через кожну точку з множини.

MST – мінімальне остовне дерево.

RST – Rectilinear Steiner Tree – прямолінійне дерево Штейнера.

IXP – Internet Exchange Points – фізична інфраструктура, через яку постачальники Інтернет-послуг та мережі доставки контенту обмінюються Інтернет-трафіком між своїми мережами (автономні системи).

P2P – peer-to-peer networks – однорангова мережа.

ДШ – дерево Штейнера.

ЗШ – задача Штейнера.

ВСТУП

Оцінка швидкості передачі інформації та її надійність у мережах Інтернет залежить від топологічної відстані, критерій оцінки швидкості передачі інформації є число транзитних переходів між вузлами. А ймовірність збоїв як раз має зворотну залежність від числа транзитних вузлів. І це враховують при розробці протоколів маршрутизації.

В даний час проблема максимального зближення джерел інформації та користувачів надзвичайно актуальна у віртуальному Інтернет-просторі. Про це свідчить інтенсивний розвиток мереж обміну трафіком піринговых мереж P2P та мережі доставки контенту CDN. В даний час топологічна відстань між вузлами мережі Інтернет, до яких належать і інформаційні ресурси та користувачі, в різних сегментах мережі становить у середньому від 4 до 5 вузлів, а в деяких випадках менше чотирьох, або, навпаки, більше шести вузлів. Це означає, що підключення до Інтернету (процес приєднання нового вузла до існуючих вузлів глобальної мережі) та розміщення інформаційних ресурсів у мережі дає технологічну користь.

Глобальні мережі не є стабільними: змінюється їхня структура, кількість учасників, вартість послуг. Тому розробити мережу один раз і назавжди неможливо. Під час побудови комп'ютерних мереж необхідно враховувати відстані між вузлами. Конфігурація фізичних зв'язків визначається мережевими з'єднаннями комп'ютерів і може бути представлена у вигляді графа, вузлами якого є комп'ютери та комунікаційне обладнання, а ребра відповідають відрізкам кабелю, які зв'язують пари вузлів. Мета фізичної структуризації – забезпечити побудову мережі не з одного, а з декількох фізичних відрізків кабелю. Причому ці різні у фізичному відношенні відрізки мали, як і раніше, працювати як єдине середовище, що розділяється.

Для постійного перерахунку мережі використовують алгоритми пошуку найкоротшого шляху у графі. Одним зі способів побудови

найкоротшого шляху є використання дерев Штейнера. Ця задача має безліч способів розв'язання. Суть задачі Штейнера полягає у знаходженні найкоротшої мережі прямолінійних відрізків, що пов'язують між собою задану множину точок. Для розв’язання задачі передбачається створення нових точок, званих точками Штейнера і які використовуються як вузли шуканої найкоротшої мережі.

Таким чином, проблему побудови топології локальної мережі можна звести до проблеми розв'язання задачі Штейнера, де термінальні точки – це вузли мережі, які необхідно з'єднати, а точки Штейнера – це місця розташування маршрутизаторів.

Метою роботи є розробка застосунку для побудови топології мережі мінімальної протяжності на основі розв'язання задачі Штейнера для покращення надійності працездатності мережі.

Основними задачами дипломної роботи є:

- огляд методів розв’язання задачі Штейнера;
- формування вимог до програмного застосунку;
- реалізація алгоритму Мелзака для побудови дерева Штейнера для терміналів;
- реалізація алгоритму Прима;
- тестування програмного застосунку в режимі реального часу;
- порівняння результатів алгоритму Мелзака та алгоритму Прима.

1 ОГЛЯД МЕТОДІВ ПРОЕКТУВАНННЯ ТОПОЛОГІЇ МЕРЕЖІ

1.1 Технології зближення інформації. Мережі IXP, P2P, CDN

Технології зближення інформації та користувачів розвиваються протягом усього часу існування Інтернету як глобальної мережі. Розглянемо найбільш відомі в даний час напрямки.

Зокрема, в єдиному протоколі глобальної маршрутизації BGP-4 основним транзитивним критерієм вибору маршруту є порівняння відстаней до місця призначення, що вимірюються кількістю транзитних вузлів – так званих автономних систем. Протокол BGP забезпечує передачу пакета в напрямку того вузла, через який проходить найкоротший маршрут [1,2].

Мережі IXP сприяють скороченню транзитів, забезпечуючи можливість фізичної взаємодії багатьох вузлів при скороченні капіталовкладень. IXP – це інфраструктура фізичного, канального або мережевого рівня, що дозволяє об'єднувати мережі на відповідному рівні за єдиною технологією. Всі учасники IXP анонсують маршрути до своїх мереж в єдину точку обміну маршрутами, яку надає IXP [2].

Мережі P2P – це віртуальні мережі, або сервіси зі створення де-централізованих віртуальних мереж, що виконують децентралізоване зберігання та поширення інформації, розподілені обчислення, обмін повідомленнями та ін. У такій розподіленій архітектурі окремі вузли-сусіди є як постачальниками, так і споживачами ресурсів. У піринговій мережі завдання (наприклад, пошук файлів або трансляція потокового аудіо-або відеоконтенту) розподіляються між кількома з'єднаними між собою сусідами. Частина ресурсів (обчислювальні потужності, дисковий простір або пропускна здатність мережі) кожного з них безпосередньо доступна іншим учасникам мережі без централізованої координації серверами [2].

Мережі CDN – географічно розподілена мережева інфраструктура, що дозволяє оптимізувати доставку та дистрибуцію контенту кінцевим

користувачам в Інтернеті. Використання CDN скорочує кількість транзитних вузлів, що суттєво збільшує швидкість завантаження контенту з мережі Інтернет. При цьому відсутні різкі зміни швидкості завантаження та підвищується якість потоку даних [2].

Прагнучи до наближення власних інформаційних ресурсів до споживача, корпорація Google розвиває власний сервіс GGC, що є CDN глобального масштабу. Для цього Google встановлює власним коштом кешируючі телекомунікаційно-інформаційні комплекси у «перспективних».

Інтернет-сервіс-провайдерів, організує взаємодію з ними за протоколом BGP і перенаправляє запити користувачів до служб Google на найближчий до їх провайдеру комплекс GGC [2].

Для ефективної роботи даних мереж важливою є найкоротша відстань між вузлами.

В даний час різні постановки задачі Штейнера застосовуються в таких областях як дискретна оптимізація, обчислювальна геометрія, проблеми трасування при проектуванні СБІС, комунікаційних мереж, механічних та електричних систем і т.д [1].

1.2 Постановка задачі Штейнера

Задача Штейнера полягає у пошуку мінімального дерева Штейнера, яке можна використовувати для побудови найкоротшої мережі, що з'єднує заданий кінцевий набір вузлів.

П'єр Ферма вивів таку постановку задачі: для заданих трьох точок знайти таку четверту, що якщо з неї провести три відрізки в дані точки, то сума цих трьох відрізків дасть найменшу величину [3].

Розв’язанням цієї задачі є побудова точки S, званої точкою Ферма (іноді точкою Торрічеллі), яка для трьох заданих точок A, B і C дає мінімально можливу суму довжин відрізків AS, BS, CS.

В. Ярник і О. Кесслер сформулювали узагальнення задачі Ферма,

замінивши три точки на довільне кінцеве число. А саме, задача полягає в описі зв'язних плоских графів найменшої довжини, що проходять через дану скінченну множину точок площини [3].

Отже, задача Штейнера має таке формулювання: на площині задано N точок. Потрібно з'єднати ці точки ламаними лініями таким чином, щоб кожна точка була з'єднана з кожною і щоб сумарна довжина всіх проведених ліній була мінімальною.

Задача Штейнера є NP- повною задачею [3], тому метод знаходження точного мінімуму не існує навіть на даний момент. Зростання складності пов'язане з втратою дискретності, оскільки розташування перехресть вже не збігається з самими об'єктами, і строго кажучи, на площі в 1 кв.м. існує нескінченна кількість можливих місць розташування розвилок. Точне й ефективне рішення існує, якщо розглядаються відносно прості мережі.

1.3 Евклідова задача Штейнера

Евклідову задачу Штейнера – геометрична задача, у якій потрібно безліч точок P на евклідовій площині з'єднати лініями так, щоб сума довжин відрізків була мінімальною [4].

Якщо допускається на площині введення додаткових «штучних» вершин (званих точками Штейнера), то довжину зв'язувального дерева можна зменшити відповідним підбором точок.

Таким чином, для розв'язання задачі Штейнера можна додати в будь-яких місцях площини стільки точок Штейнера, скільки необхідно для побудови найкоротшого дерева, що стягує множину з P точок.

Отримане дерево називають найкоротшим деревом Штейнера [5].

Незважаючи на ту увагу, яку приділяли і приділяють евклідовій задачі Штейнера, за допомогою наявних алгоритмів розв'язання можливе тільки для задач, де вихідних точок (термінальних) не більше 10, в іншому випадку рекомендуються вирішувати евристичними алгоритмами.

1.3.1 Окремий випадок задачі Штейнера: три точки

Якщо метою побудови мережі Штейнера є множина, що складається з трьох точок, то це постановка задачі Торрічеллі – Ферма: у трикутнику ABC знайти точку F таку, що сума відстаней від F до вершин A, B і C мінімальна [5].

Одне з найвідоміших рішень запропонував Наполеон Бонапарт. Для отримання розв'язку даним методом має виконуватися одна єдина умова: найбільший кут трикутника має бути меншим за 120 градусів.

Нехай F – довільна точка всередині трикутника. Повернемо трикутник ABF навколо вершини B назовні на 60 градусів.

У цьому випадку AF = A'F' і BF = B'F' за побудовою, BF = F'F, бо трикутник BFF' рівносторонній, отже, сума відстаней від F до A, B, C дорівнює довжині ламаної A'F'FC (рис. 1.1).

Ця сума стане мінімальною, якщо F прийме таке положення, що ламана стане прямою. Для цього необхідно, щоб ділянка AF'F стала прямою, тобто щоб ∠A'F'B і, отже, ∠AFB дорівнювала 120°. Ще потрібно, щоб ділянка F'FC стала прямою, тобто. ∠BFC дорівнював 120°. Третій кут при точці F автоматично стане рівним 120°. Отже, всі три кути при шуканій точці F мають дорівнювати 120°.

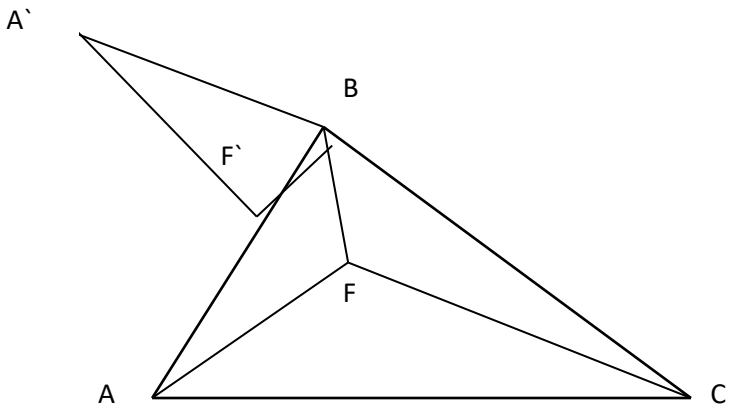

Рисунок 1.1 – Рішення Бонапарта

Наступний варіант розв'язання цієї задачі запропонував Торрічеллі [5]: необхідно провести відрізки прямих, що з'єднують вихідні точки (назвемо їх *A*, *B* і *C*), з точкою у вершині найбільшого кута (припустімо, це *B*) (рис. 1.2). Якщо кут *B* більший або дорівнює 120°, то шукана точка *P* збігається з точкою *B*. Інакше кажучи, найкоротша мережа в цьому випадку являє собою просто два відрізки прямих між точками *A* і *B* и точками *B* і *C*.

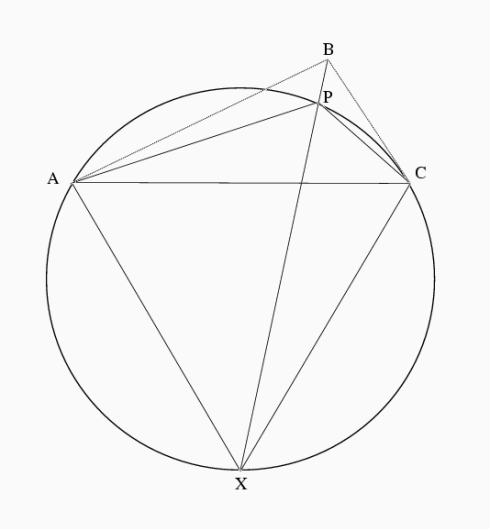

Рисунок 1.2 – Рішення Торрічеллі

Якщо кут у точці *B* менший за 120°, то точка *P* має знаходитися десь усередині трикутника. Щоб знайти її, слід побудувати рівносторонній трикутник з основою на найбільшій стороні трикутника *ABC*, а саме на відрізку *AC*. Третя вершина рівностороннього трикутника *(X*) розташована на протилежному боці від точки *B* відносно *AC*. Навколо побудованого рівностороннього трикутника описуємо коло і проводимо пряму, що з'єднує точки *B* і *X*. Точка *P* буде на перетині цієї прямої та кола. Поєднавши точки *A, B* і *C* з точкою *P*, отримуємо три кути, які точно дорівнюють 120° кожен, і шукану найкоротшу мережу. Більш того, довжина відрізка *BX* виявляється рівною довжині найкоротшої мережі.

Альтернативне рішення запропонував Ферма. На сторонах трикутника ABC у зовнішню сторону необхідно побудувати рівносторонні

трикутники ABR1, BCR2 и ACR3. Відрізки AR2, BR3 і CR1 перетинаються в точці F – точці Ферма.

Точку F трикутника, сума відстаней від якої до вершин трикутника є мінімальною, називають точкою Ферма, точкою Торрічеллі. Коли всі кути трикутника менші за 120°, то точка Ферма – це така точка F у трикутнику, з якої всі сторони трикутника видно під одним і тим самим кутом 120°. Якщо ж один із кутів трикутника більший або дорівнює 120°, то точкою Ферма в такому трикутнику буде вершина цього кута.

1.3.2 Основні властивості задачі Штейнера

Задача Штейнера для трьох точок дає деяку інформацію про геометрію найкоротших дерев Штейнера. Розберемо основні властивості точок Штейнера і всього дерева Штейнера загалом для довільного числа точок [6,7].

Властивість 1. Найкоротше дерево Штейнера складається з відрізків.

Перша властивість досить очевидна, і випливає з того, що найкоротшим шляхом з однієї точки в іншу є відрізок прямої.

Таким чином, найкоротша мережа є плоским графом – об'єднанням кінцевого числа відрізків. Кінці цих відрізків – вершини графа, а самі відрізки – його ребра. Вихідні точки називають справжніми вершинами графа, а всі інші його вершини (перехрестя) – додатковими або власне точками Штейнера. Цей граф зв'язний, тобто з будь-якого вузла можна дістатися по ребрах у будь-який інший вузол. Більш того, цей граф однозв'язний, тобто для будь-якої пари вершин існує єдиний шлях по ребрах, що їх зв'язує. Зв'язний граф є однозв'язним тоді й тільки тоді, коли він не містить замкнутих шляхів. Якби найкоротша система доріг не була однозв'язною, то існував би замкнутий шлях. Прибравши будь-яке ребро з цього шляху, отримали б зв'язний граф меншої довжини. Отже, найкоротшу мережу треба шукати серед однозв'язних графів, які містять дані точки як

вершини, а також можуть мати й додаткові вершини.

Властивість 2. Будь-які два ребра, що виходять з однієї вершини, утворюють кут не менше 120°. Якщо з вершини A виходять ребра AB і AC і кут між ними менший за 120°, то ми можемо замінити цю пару ребер іншими, які так само пов'язують точками A, B і C, але мають меншу сумарну довжину. Якщо в трикутнику ABC усі кути менші за 120°, то поставимо одну додаткову вершину F – точку Торрічеллі – Ферма цього трикутника, з'єднаємо її з вершинами A, B і C, а ребра AB і AC приберемо. Отримаємо зв'язний граф меншої довжини. Якщо ж у трикутнику ABC, припустимо, кут при вершині B більший або дорівнює 120°, то прибираємо ребро AC, а замість нього ставимо BC. Знову отримаємо зв'язний граф меншої довжини.

Властивість 3. Зі справжньої вершини може виходити одне, два або три ребра. Якщо виходить два ребра, то кут між ними більший або дорівнює 120°; якщо три, то вони утворюють між собою кути в 120°. Більше трьох ребер виходити не може, інакше один із кутів буде меншим за 120°. Отже, справжні вершини бувають трьох типів.

Властивість 4. З кожної додаткової вершини (точки Штейнера) виходять три ребра з кутами 120°. Перший тип для додаткової вершини неможливий (якщо з додаткової вершини виходить тільки одне ребро, то ця вершина не потрібна, бо її можна прибрати разом із ребром), другий – також неможливий (якщо додаткова вершина M з'єднана ребрами тільки з двома вершинами B і C, то видалимо ці ребра разом із самою вершиною M, а точки B і C з'єднаємо ребром; отримаємо граф меншої довжини).

Властивість 5. Якщо в досліджуваній мережі N об'єктів, то кількість k точок Штейнера буде в межах $0 \le k \le N-2$.

Найкоротша система доріг, що зв'язує k точок, є мережею Штейнера. Мережа Штейнера – довільний зв'язний граф, ступені якого не перевершують трьох. Мережа Штейнера без вершин ступеня два називається невиродженою. Вершини мережі Штейнера ступеня 1 і 2 називаються нерухомими точками цієї мережі.

1.3.3 Алгоритм Мелзака

В алгоритмі Мелзака розглядаються багато можливих з'єднань між заданими точками і багато можливих розташувань точок Штейнера. Алгоритм можна умовно розбити на дві частини. У першій його частині множину вихідних точок просто підрозділяють на різноманітні підмножини. У другій частині для кожної такої підмножини створюється низка можливих дерев Штейнера з використанням побудови, аналогічної тій, що застосовується до задачі з трьома точками [5].

Так само як і для трьох точок, замість двох вихідних точок можна підставити одну точку, що їх замінює, не змінюючи результату розв'язання. Однак у загальному випадку алгоритм має вгадати, яку пару слід замінити, і тому він перебирає всі можливі пари. Більш того, точка, що замінює, може розміщуватися по будь-який бік від прямої, що з'єднує дві точки, що замінюються, оскільки рівносторонній трикутник, який використовують під час побудови, може бути орієнтований в одному з двох напрямків. Після того як одну з точок у підмножині замінено однією з двох можливих точок, що замінюють, на кожному наступному кроці алгоритму заміщують або дві інші початкові точки, або одну початкову та одну замісну, або дві замісні з іншою точкою-замісницею; і так доти, доки всю підмножину не буде зведено до трьох точок.

Щойно для цих трьох точок знайдено точку Штейнера, алгоритм починає працювати у зворотному напрямі, намагаючись визначити точку Штейнера, відповідну кожній замісній точці.

Спроба може закінчитися невдачею. Однак успішна спроба призводить до виникнення дерева Штейнера, що з'єднує кожну вихідну точку підмножини з деревом одним ребром. Розглянувши, таким чином, усі послідовності, що заміщають, алгоритм обирає найкоротше з цих дерев Штейнера для підмножини. Комбінуючи між собою всілякими способами найкоротші дерева Штейнера для підмножин так, щоб охопити вихідну множину точок, можна побудувати всілякі локально мінімальні дерева Штейнера і визначити геометрію найкоротшої мережі.

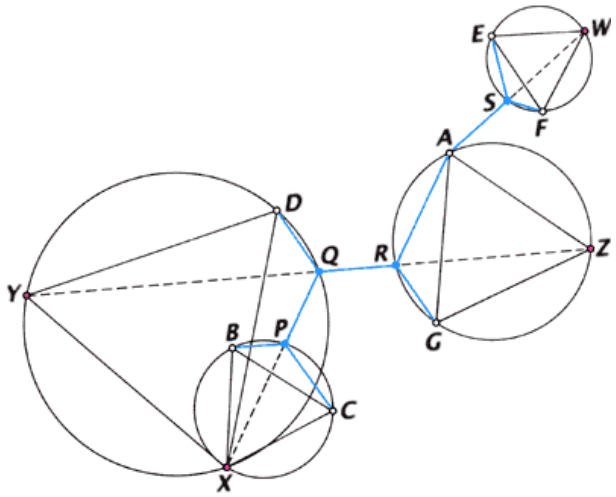

Рисунок 1.4 – Алгоритм Мелзака [5]

Алгоритм Мелзака може потребувати колосального часу навіть для невеликих задач, оскільки в ньому розглядається багато варіантів. Наприклад, задачу для 10 точок може бути розподілено на 512 підмножин вихідних точок. І хоча двоточкові підмножини не потребують великого обсягу роботи, кожна з 45 підмножин із вісьмома точками має два мільйони послідовностей, що заміщають. Крім того, існують ще понад 18 000 способів об'єднати ці підмножини в дерева.

1.4 Огляд інших алгоритмів розв'язання задачі Штейнера на площині

Зрозуміло, дослідники знайшли ефективніші шляхи організації обчислень і зуміли підвищити швидкодію алгоритму. Замість того щоб розглядати геометрію задачі, вони фокусують увагу на можливих конфігураціях з'єднань у мережі, тобто на її топології. Топологія вказує, які точки з'єднані одна з одною, а не дійсні розташування точок Штейнера. Прийнявши певну топологію, можна знайти відповідний ланцюжок заміщення відносно швидко. За такої організації процесу швидкість обчислення найкоротших дерев Штейнера для підмножин трохи зростає.

Наприклад, для підмножини з 8 точок алгоритм має розглянути лише близько 10 000 різних топологій замість двох мільйонів різних послідовностей заміщення.

Оскільки кількість можливих топологій швидко зростає з розміром підмножини, задачі Штейнера можуть стати менш трудомісткими лише в тому разі, якщо потрібно розглядати тільки дуже невеликі підмножини вихідної множини точок.

У пізніших алгоритмах (ніж алгоритм Мелзака) проводять усічення обчислювальної процедури, тобто припиняють ті гілки обчислення, які заздалегідь мають призвести до порівняно довгих сіток. Нові методи усічення дійсно значно скорочують обсяг обчислень. Програми, засновані на алгоритмі Мелзака (наприклад, програма Е. Кокейна з Університету Вікторії), могли розв'язати будь-яку задачу для 9 точок і деякі задачі для 12 точок приблизно за півгодини. Пізніша програма, розроблена тим же Кокейном, змогла розв'язати всі задачі для 17 точок і більшість випадково згенерованих задач для 30 точок усього за кілька хвилин.

Хоча для задачі Штейнера знайдено експоненціальні алгоритми (наприклад, алгоритм Мелзака), жодного поліноміального алгоритму знайти для неї не вдалося. І шанси на те, що ефективний алгоритм буде коли-небудь знайдений, дуже малі. Задача Штейнера належить до класу NP-повних.

Е. Гілберт і Х. Поллак висловили припущення про те, що відношення довжини найкоротшого дерева Штейнера до довжини мінімального остовного дерева дорівнює, що найменше, $\sqrt{3}/2$, тобто дерево Штейнера не більше ніж на 13,4% коротше мінімального остовного дерева. Це відношення $\sqrt{3}/2$ виникає в простому прикладі, коли три вихідні точки є вершинами рівностороннього трикутника.

Для вирішення задачі великого розміру зазвичай використовують евристичні методи. Кельманс описав простий очевидний алгоритм побудови наближеного розв'язку задачі шляхом послідовного додавання

терміналів до вже опрацьованих і перебудови отриманого на попередньому кроці розв'язку. Одним із найефективніших евристичних алгоритмів розв'язання задачі Штейнера є генетичний алгоритм.

1.4.1 Генетичний алгоритм

Генетичний алгоритм – евристичний алгоритм пошуку, який використовують для розв'язання задач оптимізації та моделювання шляхом послідовного добору, комбінування та варіації шуканих параметрів із використанням механізмів, що нагадують біологічну еволюцію [6,8].

Популяція – набір рішень. Еволюція популяції – чергування поколінь, у яких хромосоми змінюють свої гени, щоб кожна нова популяція якнайкраще пристосовувалася до нового (зовнішнього) середовища.

Основною складністю застосування генетичних алгоритмів для побудови дерев Штейнера є оптимальне кодування та вибір ефективних генетичних операторів. Підвищення ефективності алгоритму досягається за рахунок введення в процедуру пошуку оператора мутації та рекомбінації.

Дана множина вершин у площині розбивається на тріади відповідно до розташування вершин на координатній площині. Далі відбувається побудова дерев Штейнера для кожної з тріад. Ген у хромосомі міститиме інформацію про одну з тріад.

На наступному етапі проводимо відбір. Дві хромосоми, що мають найменше значення цільової функції (сумарної довжини ребер) братимуть участь у відтворенні далі. На наступному етапі до відібраних хромосом застосовується модифікований точковий оператор кросинговера. Отримання нащадків досягається шляхом заміни одного з генів батьків (наприклад, третій ген першого з батьків стає на місце третього гена другого з батьків, а той, своєю чергою, на місце першого з батьків, унаслідок чого, отримавши двох нащадків). Підвищення ефективності алгоритму досягається за рахунок введення в процедуру пошуку оператора мутації,

транслокації та рекомбінації.

Оператор мутації випадково вибирає ген у хромосомі й обмінює його на поруч розташований ген.

У процесі транслокації випадковим чином проводиться розрив у кожній хромосомі у визначеному для обох хромосом місці. Під час формування нащадка береться ліва частина до розриву з одного з батьків та інверсія правої частини до розриву з другого з батьків. У процесі рекомбінації в результуючу популяцію додаються хромосоми, які є батьками на етапах кросинговера, мутації та транслокації. Далі проводиться елітна селекція. Відбираються дві найкращі хромосоми. У разі якщо критерій зупинки не досягнуто, процедура оптимізації повторюється. Критерієм зупинки є кількість популяцій, яку визначає користувач.

Застосування генетичних алгоритмів для розв'язання задачі побудови дерев Штейнера дає, можливо, не найкращий результат, але, за відповідного добору генетичних операторів, він може бути досить хорошим. Головна перевага застосування генетичних алгоритмів – відносно невеликий час вирішення. Особливо це важливо в умовах дуже великої кількості абонентів мережі, яка постійно зростає, і мереж, що змінюються.

1.5 Лінійна задача Штейнера

Лінійні (прямокутна) задачі Штейнера мають застосування під час монтажу друкованих схем електронних пристроїв і в логістиці практично не використовуються, на відміну від евклідових задач.

У цій постановці задачі Штейнера для визначення відстані між терміналами використовується манхеттенівська метрика:

$$L_1 = |x_1 - x_2| + |y_1 - y_2|\,, \tag{1.1}$$

де $(x_1, y_1), (x_2, y_2)$ – координати відповідних точок.

Для розв'язання прямолінійної задачі Штейнера використовують два основні підходи: розв'язання задачі Штейнера з використанням теорії графів і геометричні методи.

Хенен у роботі довів, що мінімальне RST завжди буде підграфом ґратчастого графа (Hanan grid graph), вершини якого формуються шляхом перетину горизонтальних і вертикальних прямих, проведених через кожен із терміналів.

Найбільший практичний інтерес становить алгоритм послідовного введення додаткових вершин у дерево Прима-Краскала. Цей алгоритм відомий як 1-Steiner algorithm. Використання запропонованого алгоритму дає змогу будувати дерева Штейнера, довжина яких у середньому не перевищує довжину оптимального дерева Штейнера більш ніж на 0,5 відсотка. Дерево Прима-Краскала не може перевищувати довжину відповідного дерева Штейнера більш ніж у 1,5 раза.

Ідея алгоритму: Цей алгоритм послідовно вводить у поточний ДПК кожну з додаткових вершин ґратчастого графа, будує нове дерево Прима-Краскала і запам'ятовує отриманий виграш у довжині. Після оцінки всіх додаткових точок у поточне дерево включається точка з максимальним виграшем. При цьому щоб уникнути в процесі перетворення появи надлишкових точок, необхідно враховувати, що додаткові точки в дереві Штейнера можуть мати тільки ступінь 3 і 4.

1.6 Задача Штейнера на графах

Обмеження на число точок Штейнера і можливості локалізації всіх точок задачі призводить до найбільш часто досліджуваної постановки задачі: задачі Штейнера на графах. У цій задачі потрібно знайти найкоротше дерево *T*, яке стягує (покриває) задану підмножину $P \subset X$ вершин графа *G*. Інші вершини, що належать *X*-*P*, можуть або стягуватися деревом *T*, або ні, залежно від вимоги мінімізації довжини дерева *T*. Ребрам графа приписано невід'ємні ваги, а під довжиною дерева розуміють суму ваг ребер, що входять у це дерево. Таким чином, задача Штейнера на графі еквівалентна

знаходженню найкоротшого кістякового дерева довільного підграфа $G' = (X', \Gamma)$ графа G за умови $P \subseteq X' \subseteq X$. Один з алгоритмів розв'язання задачі Штейнера на графах можна зустріти в роботі [7].

Хакімі вперше сформулював задачу і запропонував топологічний метод її розв'язання за аналогією з методом Мелзака для евклідової задачі Штейнера. Більшість алгоритмів засновані на методі динамічного програмування (алгоритми Левіна, Дрейфуса і Вагнера тощо). Так само як і під час розв'язання задачі Штейнера на площині, під час розв'язання задачі Штейнера на графах доцільно використовувати різні евристики.

1.7 Алгоритм Прима

Алгоритм Прима – алгоритм побудови мінімального остовного дерева зваженого зв'язного неорієнтованого графа [8].

Граф G(V, E) задано множиною вершин X_1, X_2, …, X_n і множиною ребер, що з'єднують між собою певні вершини. На вхід алгоритму подається зв'язний неорієнтований граф. Для кожного ребра задається його вартість.

Спочатку береться довільна вершина і знаходиться ребро, інцидентне даній вершині, що має найменшу вартість. Знайдене ребро і з'єднані ним дві вершини утворюють дерево. Потім, розглядаються ребра графа, один кінець яких – вершина, що вже належить дереву, а інший – ні; з цих ребер вибирається ребро найменшої вартості. Обране на кожному кроці ребро приєднується до дерева. Зростання дерева відбувається доти, доки не будуть вичерпані всі вершини вихідного графа.

Результатом роботи алгоритму є остовне дерево мінімальної вартості.

В результаті аналізу можна зробити висновок: побудова локальної мережі мінімальної протяжності можна вирішити на основі задачі Штейнера на Евклідовій площині. А розв'язання задачі Штейнера можливе за допомогою алгоритму Мелзака, який збудує дерева Штейнера. Довжину мінімального дерева Штейнера знайдемо за допомогою алгоритму Прима.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

Для розв'язання поставленої задачі (планування топології локальних мереж на основі розв'язання задачі Штейнера) необхідно розробити програмний застосунок, що дає змогу знаходити розв'язання задачі Штейнера на площині для невеликої кількості точок. Як алгоритм розв'язання задачі Штейнера обрано алгоритм Мелзака.

Програмний застосунок повинен відповідати вимогам, а саме:

– можливість створення і редагування мережі, тобто додавання і видалення вузлів і зв'язків;

– візуалізація схеми побудованого маршруту;

– виведення результатів розрахунків за побудованою мережею Штейнера;

– надання необхідної інформації про програму та правила роботи з нею (допомога);

– зручний користувальницький інтерфейс (можливість додавати вихідні точки як координатами (з клавіатури), так і за допомогою лівої кнопки миші).

З урахуванням сучасних технічних засобів, програма повинна мати графічний інтерфейс, що подається на кольоровому дисплеї, сумісність з операційною системою Windows, зручну роботу з клавіатурою і мишею.

2.1 Аналітика алгоритму Мелзака

Для розв'язання задачі Штейнера необхідно застосовувати алгоритм Мелзака [9]. Теоретичний опис цього алгоритму вже було наведено в попередньому розділі, тому зупинимося детальніше саме на його реалізації. Отже, алгоритм Мелзака на першому кроці своєї роботи замінює дві будь-які точки множини (припустімо, точки A і B) на одну (точку C), що є вершиною рівностороннього трикутника, побудованого на відрізку A, B. На

наступному кроці можуть бути замінені або дві початкові точки, або початкова і додаткова (така як точка С), або дві додаткові. Ця процедура заміни працює доти, доки в множині не залишиться трьох точок (функція AlgMelz). Далі для трьох точок необхідно знайти координати єдиної точки Штейнера (точки Ферма).

Для отримання точних координат точки Ферма –Торрічеллі необхідно для початку знайти координати точок R_1, R_2, R_3 (рис. 2.1).

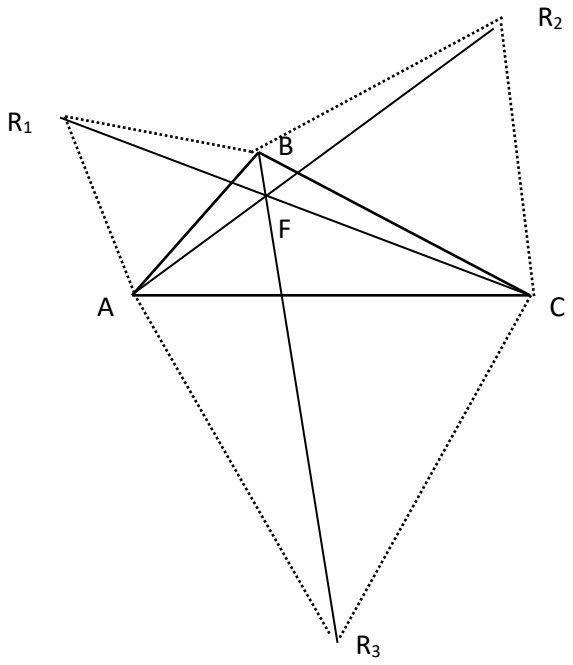

Рисунок 2.1 – Отримання координат точки Ферма

Знаючи, що трикутник AR_1B – рівносторонній, а отже, всі кути трикутника рівні між собою і дорівнюють 60°, отримуємо координати точки R_1 (2.1-2.2).

$$x_{R_1} = (x_B - x_A)\cos(60°) - (y_B - y_A)\sin(60°) + x_A\,, \tag{2.1}$$

$$y_{R_1} = (x_B - x_A)\sin(60°) + (y_B - y_A)\cos(60°) + y_A\,. \tag{2.2}$$

де (x_{R_1}, y_{R_1}) – координати точки R_1;

(x_A, y_A), (x_B, y_B), – координати точок А і В відповідно.

Існує дві точки Ферма, а нам потрібно знайти координати саме точки F, що лежить усередині трикутника. Це також необхідно враховувати. Отримавши необхідні координати точок R_1, R_2, R_3 досить просто знайти точку перетину прямих AR_2, CR_1 и BR_3. Для цього розв'яжемо систему рівнянь, що складається з двох рівнянь прямих (2.3).

$$\begin{cases} \dfrac{x - x_A}{x_{R_1} - x_A} = \dfrac{y - y_A}{y_{R_2} - y_A}; \\ \dfrac{x - x_C}{x_{R_1} - x_C} = \dfrac{y - y_C}{y_{R_1} - y_C}; \end{cases} \tag{2.3}$$

де (x, y) – координати точки Ферма F, які необхідно знайти;

(x_A, y_A), (x_C, y_C), (x_{R_1}, y_{R_1}) – координати точок A, C і R_1 відповідно.

Отримаємо такі координати точки Ферма (точки F) (2.4-2.5).

$$x_F = \frac{-1\left(\left(x_A y_{R_2} - x_{R_2} y_A\right)\left(x_{R_1} - x_C\right) - \left(x_C y_{R_1} - x_{R_1} y_C\right)\left(x_{R_2} - x_A\right)\right)}{\left(\left(y_A - y_5\right)\left(x_{R_2} - x_C\right) - \left(y_C - y_{R_2}\right)\left(x_{R_2} - x_A\right)\right)}; \tag{2.4}$$

$$y_F = \frac{\left(x_F - x_C\right)\left(y_{R_1} - y_C\right)}{\left(x_{R_1} - x_C\right) + y_C}. \tag{2.5}$$

де (x_F, y_F) – координати точки Ферма F;

(x_A, y_A), (x_C, y_C), (x_{R_1}, y_{R_1}), (x_{R_1}, y_{R_1}) – координати точок A, C, R_1 і R_2 відповідно.

Але дані координати точки Ферма будуть вірними лише в тому випадку, якщо всі кути трикутника ABC менші за 120°. Якщо ж один кут більший або дорівнює 120°, то точкою Ферма в цьому випадку буде вершина найбільшого кута. Маючи координати точки Ферма для трьох точок, що залишилися, можна приступати до наступного кроку роботи алгоритму. Ця

точка Ферма буде першою точкою Штейнера в задачі. Алгоритм починає працювати у зворотному напрямку, намагаючись відновити вихідні точки, а також точки Штейнера. Для цього всі додаткові точки по черзі починають замінюватися тими точками, які вони замінювали на першому етапі роботи алгоритму. Для точки Ферма, отриманої на попередньому кроці алгоритму, і двох відновлених точок будується нова точка Ферма. Отримана точка Ферма буде наступною точкою Штейнера для вихідної задачі. Процедуру знаходження точки Ферма було описано вище. Так триває доти, доки не будуть відновлені всі вихідні точки або не будуть видалені всі додаткові точки. Далі множину вихідних точок об'єднуємо з множиною отриманих точок Штейнера і будуємо для них мінімальне остовне дерево. Для його побудови можна скористатися алгоритмом Прима.

Побудоване таким чином дерево Штейнера не обов'язково буде мінімальним. Для отримання мінімального дерева Штейнера слід перебрати всі можливі варіанти заміни вихідних точок на додаткові на першому етапі роботи алгоритму [10,11]. Варіантів перебору для $n \geq 5$ вихідних точок може виявитися дуже багато, тому доцільно для збільшення швидкодії роботи програми використовувати процедуру відсікання тих гілок переборів, що заздалегідь призводять до неоптимального рішення [12].

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Опис інтерфейсу розробленої програми

Як мову програмування для створення програмного застосунку обрано мову C#, що дозволяє створити зручний та задоволений простий інтерфейс, а також реалізувати обраний алгоритм Мелзака, алгоритм Прима та обчислити час виконання алгоритму Мелзака.

На основній формі проекту розташовані такі елементи (рис. 3.1):

– 4 елементи типу button (кнопки під назвами «Нове дерево», «Обчислити», «Стоп» і «Додати точку»);

– один елемент picturebox (служить для представлення графічного вигляду побудованих дерев Штейнера в головному вікні програми);

– два елементи datagridview (таблиці, що містять інформацію про вихідні точки (введені користувачем за допомогою клавіатури або миші) у розв'язуваній задачі Штейнера, а також інформацію про додаткові точки (власне, точки Штейнера));

– один елемент progressbar (служить для виведення інформації про хід виконання роботи алгоритму, є наочним відображенням прогресу виконання операції пошуку найкоротшого дерева Штейнера);

– один елемент label (служить для виведення інформації про побудоване дерево Штейнера: часу роботи алгоритму, довжині дерева Штейнера і довжині мінімального остовного дерева для цього ж набору точок).

Кнопка «Обрати карту». З цієї кнопки необхідно починати роботу програми. Оброблювач цієї кнопки відкриває діалог "OpenDialog", за допомогою якого можна перейти в папку з картами та вибрати необхідну.

Спочатку з'явиться діалогове вікно, в якому потрібно вибрати карту або план приміщення. Карта з'явиться на формі (рис. 3.1).

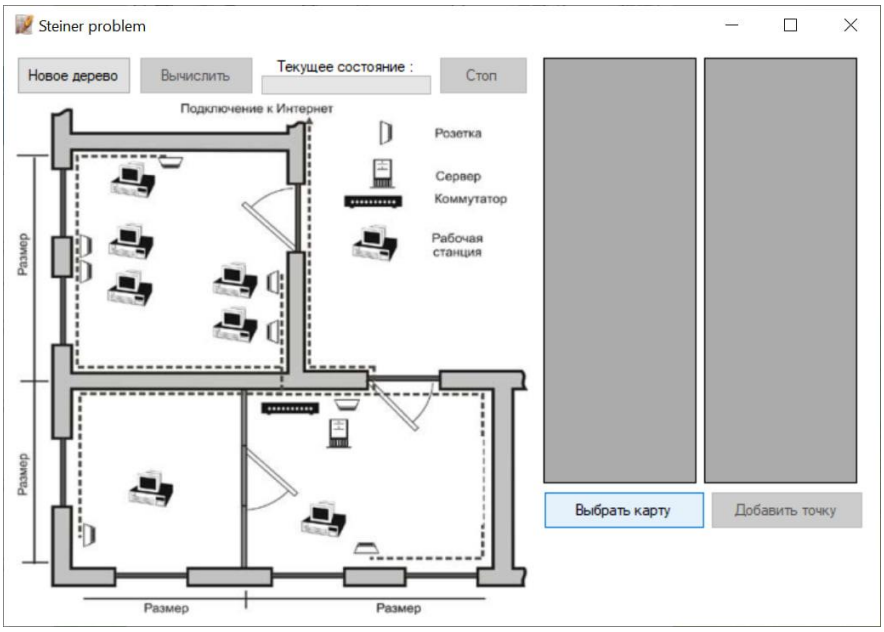

Рисунок 3.1 – Інтерфейс розробленого програмного застосунку

Кнопка «Нове дерево». Оброблювач цієї кнопки очищає всі робочі змінні, які використовувалися під час побудови попереднього дерева Штейнера, після чого накладається сітка, тепер можна приступати до введення точок для побудови наступного дерева форму (рис. 3.3).

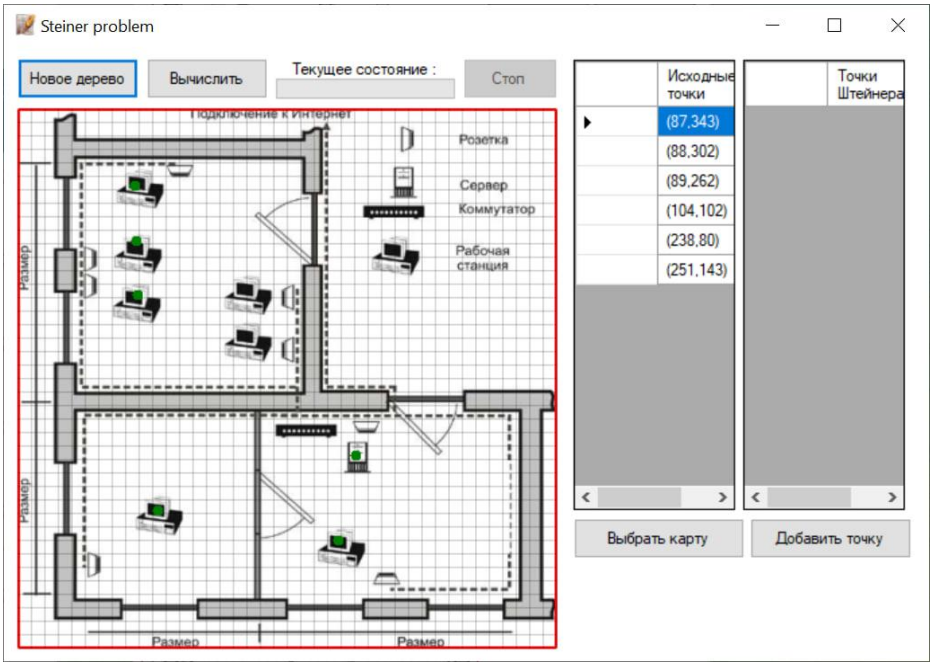

Рисунок 3.3 – На карту накладено сітку

Кнопка «Додати точку». Ця кнопка слугує для додавання нової точки до безлічі вихідних точок шляхом зазначення точних координат точки на площині. Для цього обробник кнопки створює нову форму (рис. 3.4).

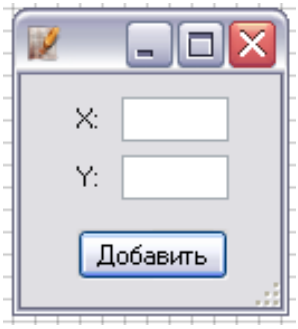

Рисунок 3.4 – Додаткове вікно для введення координат точки

На формі розташовано два textbox (служать для введення числових координат точки) і власне кнопка «Додати», яка і додає нову точку в масив вихідних точок, після чого зображення нової точки додається до зображення вже наявних точок на picturebox основного вікна програми. У разі введення некоректних даних (координат точок) користувача інформують про це за допомогою спливаючого messagebox із відповідним написом. Існує альтернативний спосіб задавання координат точки і додавання її в безліч точок для побудови дерева Штейнера – лівим кликом миші по елементу picturebox основного вікна програми. У цьому разі користувач задає розташування точки не точними координатами, а «на око».

Кнопка «Обчислити». В обробнику цієї кнопки виконується основна робота програми, тобто обчислення найкоротшого дерева Штейнера. Для коректної роботи програми під час роботи алгоритму та проведення обчислень використовується елемент backgroundworker.

Клас backgroundworker забезпечує можливість виконання тривалих операцій в асинхронному (фоновому) режимі в потоці, відмінному від основного користувацького інтерфейсу програми. Це «оберігає» користувацький інтерфейс від зависань під час роботи алгоритму і пошуку

мінімального дерева. Після завершення виконання методу backgroundworker в основне вікно програми виводиться інформація про знайдене найкоротше дерево Штейнера (час роботи алгоритму, довжина отриманого дерева, а також дерево відображається графічно на picturebox).

Обробка події натиснута кнопка «Обчислити»" реалізована наступним чином. Якщо кількість термінальних вершин n<= 9, то приступаємо до побудови дерева Штейнера, а саме.

1) Якщо точок більше 2 викликаємо функцію AlgMelz, яка рекурсивно доки не залишиться три точки, на першому кроці своєї роботи замінює дві будь-які точки великої кількості однією, яка є вершиною рівностороннього трикутника, побудованого на відрізку, що сполучає дві вибрані для заміни точки (причому на наступному кроці можуть бути замінені або дві вихідні точки, або початковий і додатковий, або два додаткові).

2) Далі для трьох точок знаходить координати єдиної точки Штейнера (точки Ферма).

3) Маючи координати точки Ферма для трьох точок, що залишилися, можна приступати до наступного кроку роботи алгоритму. Ця точка Ферма буде першою точкою Штейнера в дереві Штейнера.

4) Алгоритм починає працювати у зворотному напрямі, намагаючись відновити вихідні точки, а також точки Штейнера (функція ObrAlgMelz).

5) Для точки Ферма, отриманою на попередньому кроці алгоритму, і двох відновлених точок будується нова точка Ферма (функція TFerma).

6) Отримана точка Ферма буде наступною точкою Штейнера для початкового завдання. Так триває до тих пір, поки не будуть відновлені усі вихідні точки або не будуть видалені усі додаткові точки.

7) Далі множину вихідних точок об'єднуємо з множиною отриманих точок Штейнера і будуємо для них мінімальне остовне дерево. Для його

побудови скористалися алгоритмом Прима (його реалізація у функції PrimaTr). На рисунках 3.5 і 3.6 зображено програму безпосередньо під час роботи алгоритму Мелзака (після натискання кнопки «Обчислити»).

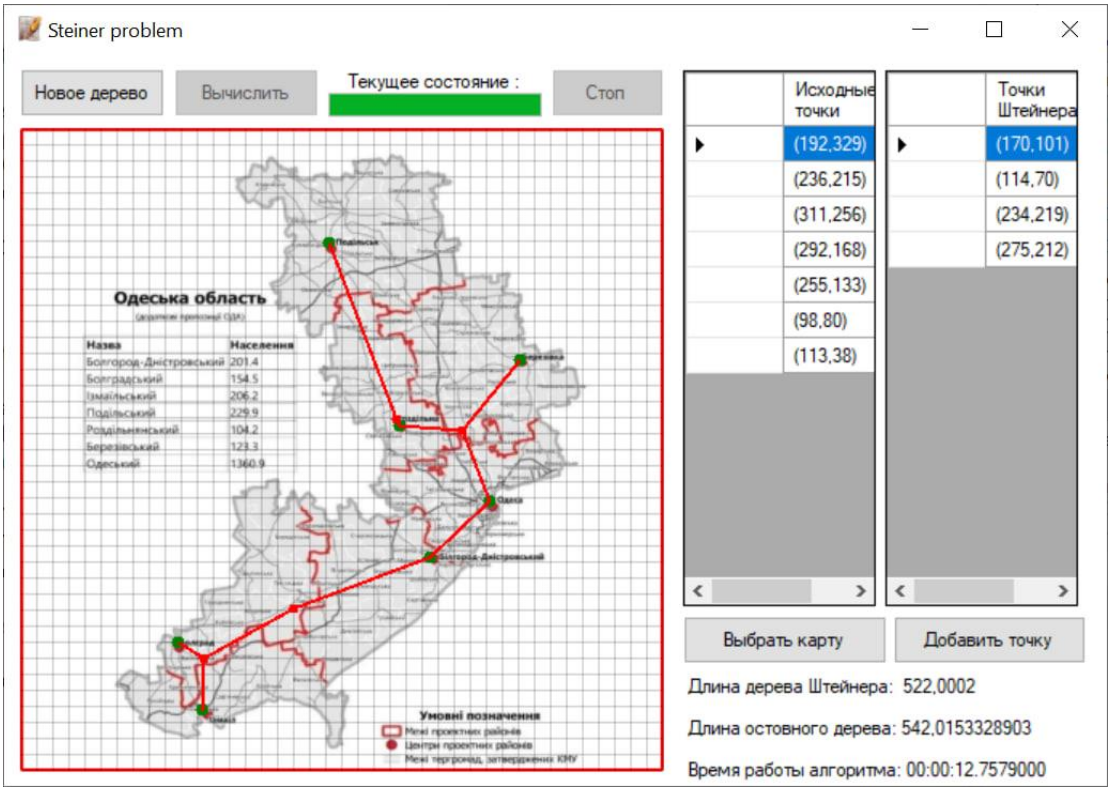

Рисунок 3.5 – Програма під час роботи алгоритму Мелзака

Побудоване дерево буде локально мінімальним деревом Штейнера, тому що алгоритм не доопрацював до кінця і не перебрав усі варіанти, що залишилися, серед яких могло бути і найкоротше дерево Штейнера. Фактично, не отримуємо найкоротших дерев Штейнера в даному випадку, але користь даної кнопки можна оцінити під час пошуку дерева Штейнера для $n \geq 8$ точок, коли час роботи алгоритму виявляється досить великим, і користувач може не дочекатися завершення роботи алгоритму.

Загальна довжина одержуваних локально мінімальних дерев Штейнера все ж таки буде меншою за довжину відповідного мінімального остовного дерева (у крайньому разі, дерево Штейнера буде збігатися з мінімальним остовним деревом).

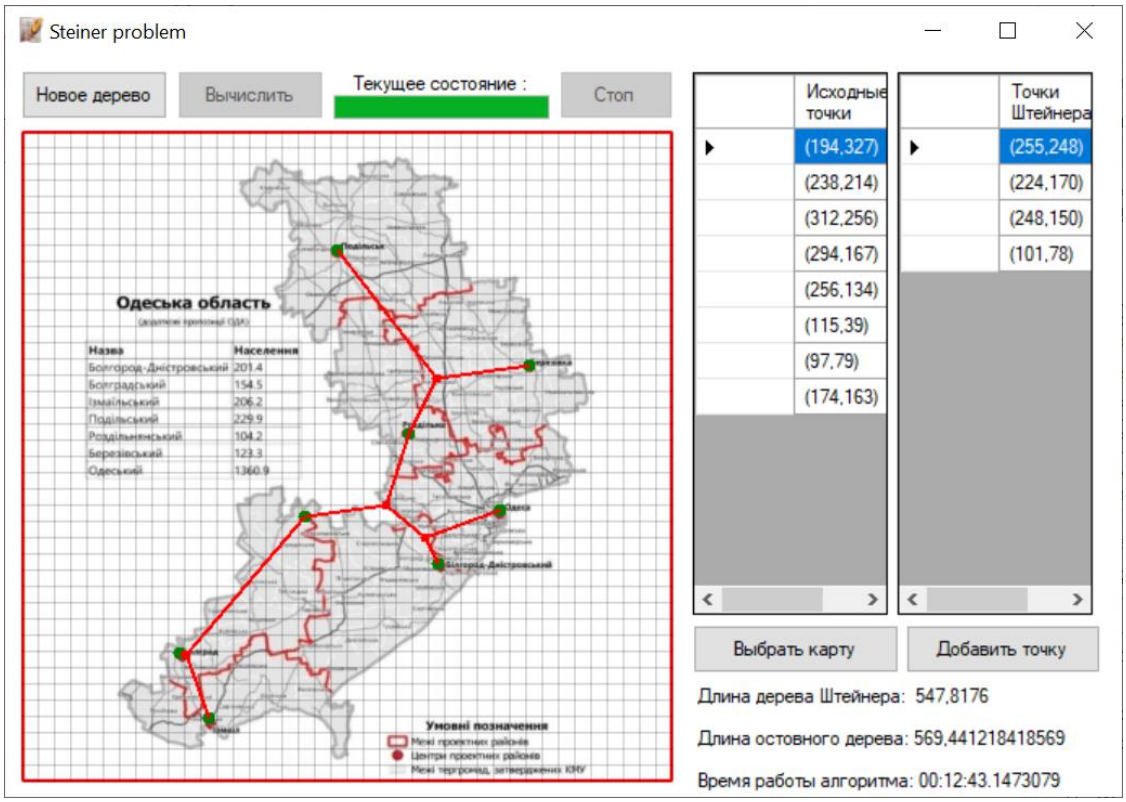

Рисунок 3.6 – Результат роботи програми для 8 точок

Також існує не менш важлива можливість видалення непотрібних точок з масиву шляхом видалення відповідного рядка з dataGridView. Це необхідно і корисно як при випадковому некоректному введенні координат точок, так і при спеціально запланованій перебудові мережі, тобто коли мережа Штейнера для даного набору точок вже побудована і необхідно внести деякі зміни у вихідні дані.

3.2 Тестування роботи алгоритма Мелзака

Під час реалізації програмного застосунку як основний алгоритм розв'язання задачі Штейнера обрано алгоритм Мелзака – найвідоміший і найдослідженіший з усіх алгоритмів розв'язання евклідової задачі Штейнера. Алгоритм прекрасно справляється з кількістю точок $n \leq 9$. І не дивлячись на те, що частину варіантів побудови найкоротшого дерева

Штейнера алгоритм не розглядає (рішення, які заздалегідь призводять до подовження мережі), на побудову дерева Штейнера для більшості задач, що складаються з 6 точок, алгоритму потрібно більше ніж 1 секунда (рис. 3.7), а для 7 точок час роботи алгоритму і перевищує 1 хвилину (рис. 3.8).

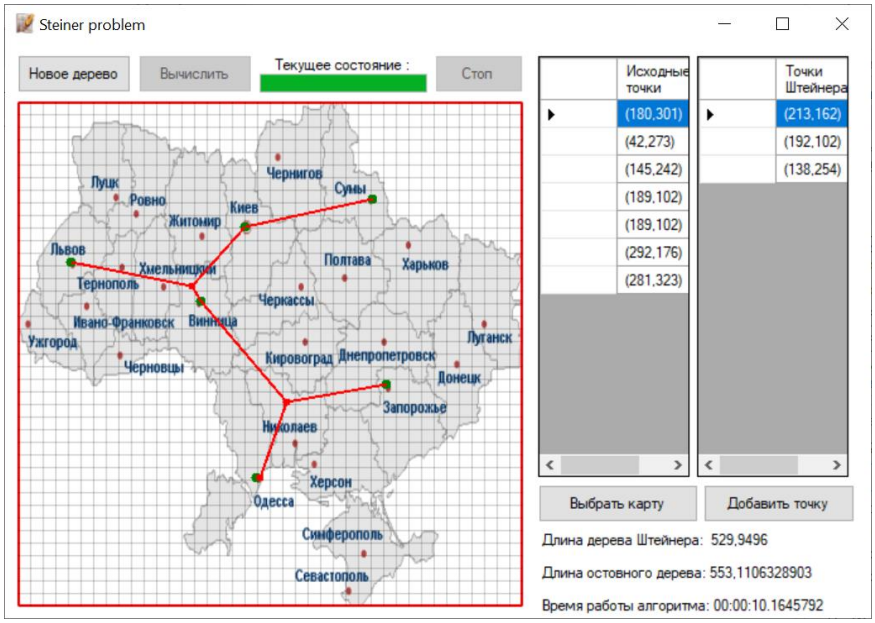

Рисунок 3.7 – Результат роботи програми для 6 точок

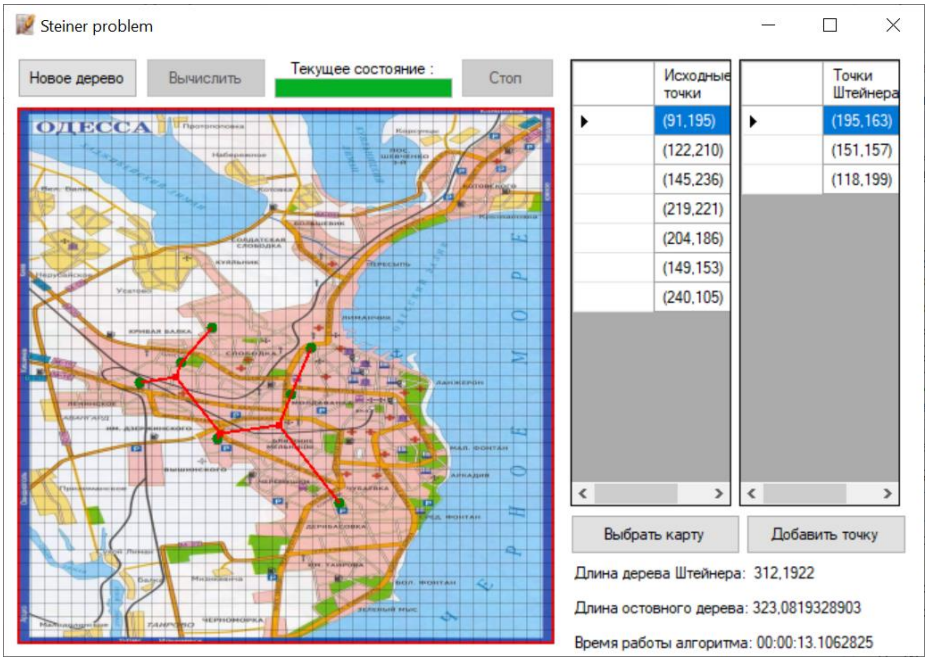

Рисунок 3.8 – Результат роботи програми для 7 точок

Така різниця в часі роботи програми для 6 і 7 точок цілком зрозуміла й обґрунтована: для 3 точок алгоритм розглядає 1 варіант розташування точки Штейнера, для 4 точок – 12 варіантів, для 5 – 240, для 6 – 7200, для 7– 302400 варіантів розташування точок Штейнера.

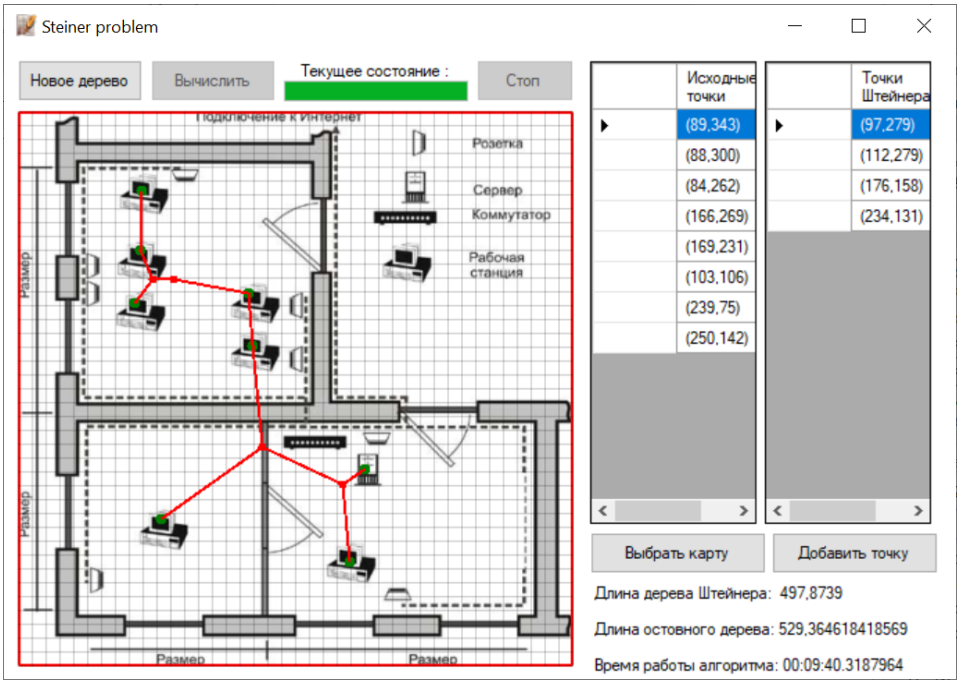

Рисунок 3.9 – Результат роботи програми для 8 точок

Розглянемо застосунок, який шукає мінімальне остове дерево за допомогою алгоритму Прима. Отримаємо повний зважений граф для восьми термінальних точок (рис 3.10).

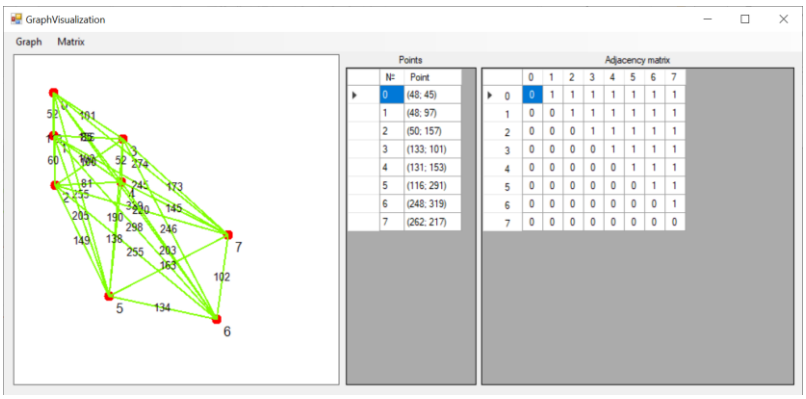

Рисунок 3.10 – Повний зважений граф

Побудуємо мінімальне остовне дерево (рис 3.11).

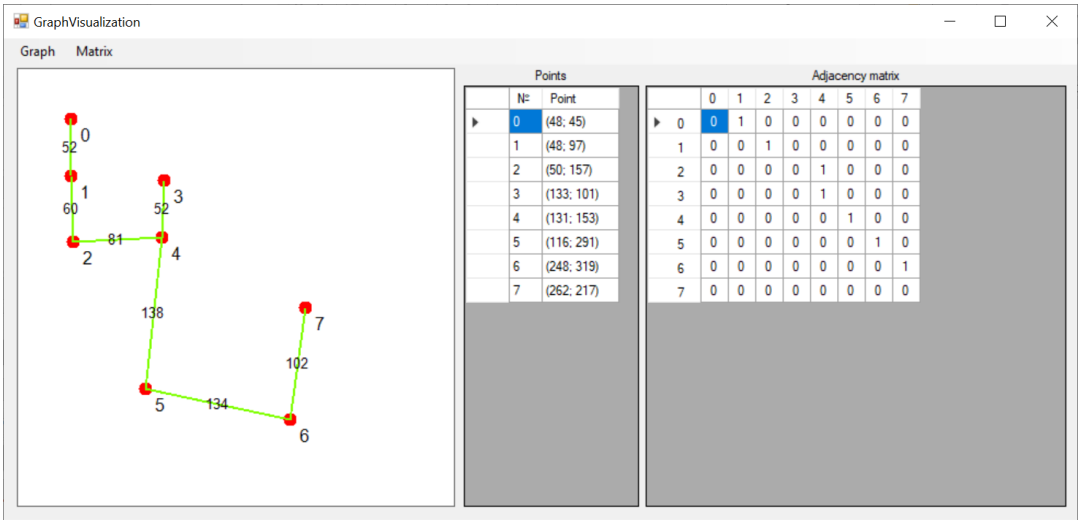

Рисунок 3.11 – Мінімальний остов (довжина 622)

Далі, такі самі термінальні точки, з такими координатами ввели в застосунок з алгоритмом Мелзака. Отримали чотири точки Штейнера (рис 3.12).

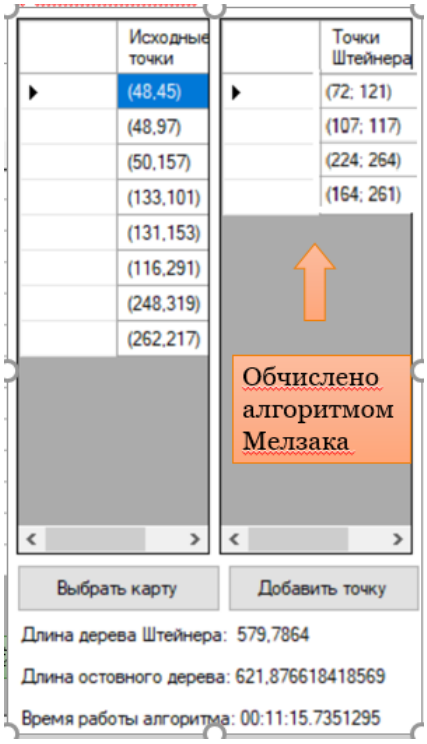

Рисунок 3.12 – Алгоритм Мелзака знайшов 4 точки Штейнера

Об'єднали множину термінальних точок та точок Штейнера. Побудували повний граф (рис 3.13).

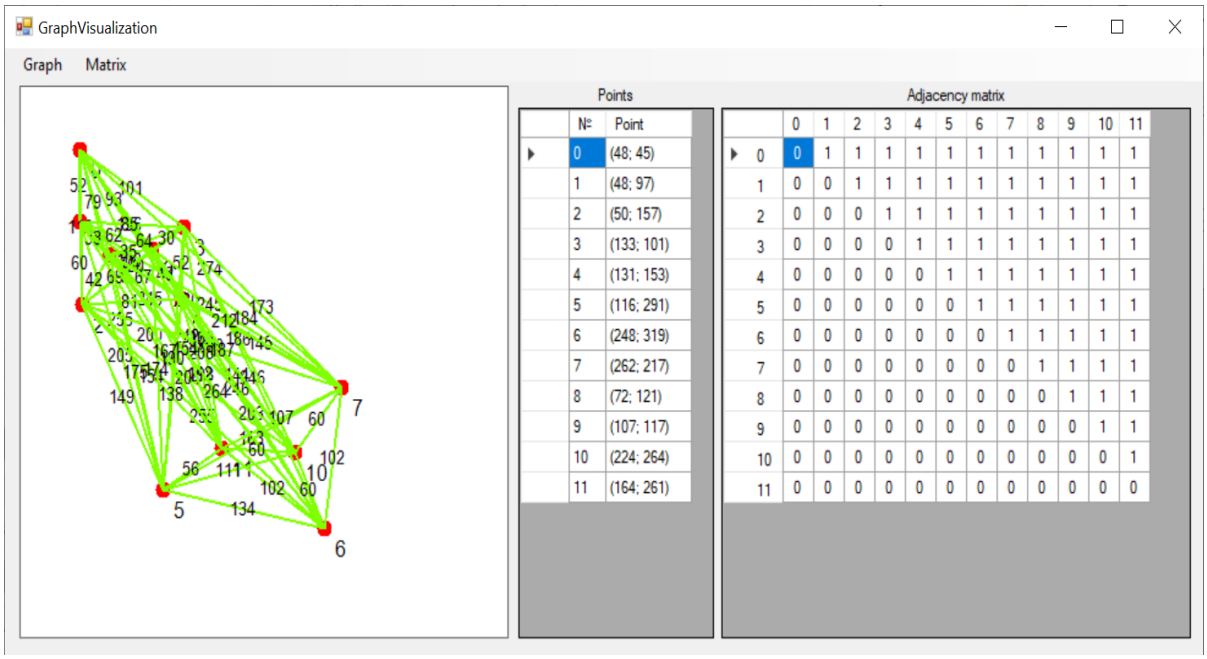

Рисунок 3.13 – Повний граф з усіма точками

Запускаємо алгоритм Прима, який будує дерево Штейнера (рис.3.14).

Рисунок 3.14 – Дерево Штейнера – довжина 580

Довжина MST -Довжина ДШ = 622-580 = 42

Як визначити довжину в метрах: (реальна площа приміщення (кв.м))/ (розмір нашої сітки (400 * 400 = 160000)), беремо корінь квадратний (отримуємо скільки в клітині метрів). Потім довжину обчислених дерев множимо на отриманий результат-отримуємо довжину кабелю в метрах.

1) Якщо площа приміщення 64 кв.
   - Масштаб ширини клітини = 0,02 м.
   - Економія = 0,84 м.
2) Якщо така топологія біля 64 кв.км.
   - Економія = 0,84 км = 840м.

Алгоритму Мелзака працює за експоненціальний час, а кількість варіантів розташування точок Штейнера в загальному вигляді дорівнює (3.1).

$$Count\ (N) = N * (N - 1) * Count(N - 1), \qquad (3.1)$$

де N – кількість вихідних точок.

Усереднені результати роботи програми для різної кількості точок наведено в таблиці 3.1.

Таблиця 3.1 – Результати роботи програми. Порівняння часу роботи алгоритму прі різної кількості вихідних точок у задачі Штейнера

	Кількість вихідних точок у задачі Штейнера					
	3 точки	4 точки	5 точок	6 точок	7 точок	8 точок
Кількість варіантів	1	12	320	7200	302400	16934400
Час роботи	0 с	0.0005 с	0.002 с	0.3 с	6 с	11 хв

Час роботи алгоритму значно скорочується, коли кількість точок у найкоротшій мережі Штейнера менша за N-2, де N – кількість вихідних точок. У прикладах, коли найкоротше дерево Штейнера збігається з

мінімальним остовним деревом, можна домогтися зменшення часу роботи алгоритму в 2 рази завдяки відсіканню тих гілок перебору, які точно не приведуть до оптимального рішення.

У таблиці 3.2 показані результати поранень довжини дерева Штейнера та довжини остовного дерева обчисленого алгоритмом Прима.

Таблиця 3.2 – Результати роботи програми-порівняння довжини

	Кількість вихідних точок у задачі Штейнера					
	3 точки	4 точки	5 точок	6 точок	8 точок	8 точок
Довжина дерева Штейнера	297	469	540	660	598	855
Довжина остова	337	504	582	700	641	903

ВИСНОВКИ

У кваліфікаційній роботі розглянути різні постановки задачі Штейнера, проведено аналіз найбільш відомих алгоритмів розв'язання задачі Штейнера.

У роботі для планування топології мереж обрано задача Штейнера в евклідовому просторі, яка вирішується алгоритмом Мелзака.

Як мову програмування обрано мову C#, що дозволяє створити зручний інтерфейс, а також візуалізувати дерево Штейнера.

Розроблений програмний застосунок дає змогу наочно представити результати роботи алгоритму пошуку найкоротшого дерева Штейнера та обчислення довжини остовного дерева алгоритмом Прима.

Тестування алгоритму Мелзака показало, шо алгоритм чудово будує дерево для $n \leq 9$ точок, про що свідчать результати роботи програми для різної кількості точок. На побудову дерева Штейнера для 6 точок, алгоритму необхідно менше 1 секунди, а для 8 точок час роботи алгоритму вже перевищує 9 хвилин.

Розроблений програмний застосунок дозволяє проектувати реальні мережі з найкоротшим шляхом, що покращує їх працездатність і збільшує функціональні можливості протоколів маршрутизації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fostering Sustainable Peering Infrastructure[Електронний ресурс] – Режим доступу: https://www.Internetsociety.org/action-plan/2023/?gclid= EAIaIQobChMIu9Wez_Ou_wIVmRh7Ch2SfQafEAAYASAAEgJC2fD_BwE#peering-infrastructure
2. Зубок В.Ю. Оптимизация связей между узлами Интернет как частній случай задачи Штейнера // Электронное моделирование, Киев, 2014., т. 36 №1, с. 29 - 39
3. Курант Р., Роббинс Г. Что такое математика? – М.: Просвещение, 1967г.
4. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи – М.:Мир, 1982 г.
5. Гордеев Э.Н., Тарасцов О.Г. Задача Штейнера. Обзор. – Дискретная математика, 1993 г., том 5, выпуск 2.
6. Кристофидес Н. Теория графов. Алгоритмический подход. – М: Мир, 1978 г., с. 166 - 168.
7. Маршалл У.Берн, Рональд Л. Грэм. Поиск кратчайших сетей – В мире науки. Scientific American, №3, 1989, с. 64 - 70.
8. Ольшевский А.И., Починский М.Ю. Решение задачи Штейнера с помощью генетического алгоритма – Бионика интеллекта, 2008 г., №2 (69), с.145 - 151.
9. Романовский И.В. Задача Штейнера на графах и динамическое программирование – Компьютерные инструменты в образовании, 2004 г., №2
10. Крістофідес Н. Теорія графів. Алгоритмічний підхід. – М: Мир, 1978 р., с. 166 - 168.
11. Yang, Z.-X. Geometry Experiment Algorithm Steiner Minimal Tree Problem // Journal of Applied Mathematics. –Vol. 2013, p. 1-10.
12. Marcelo, Z.N. An Interactive Programme for Steiner trees [Електронний ресурс] – Режим доступу: http://arxiv.org/abs/1210.7788

ДОДАТОК А

Реалізація алгоритмів Мелзака та Прима

```
private void button5_Click(object sender, EventArgs e)
        {
            button1.Enabled = true;
            string path = "img";
            if (Directory.Exists(path))
            {
                openFileDialog1.InitialDirectory = path;
                openFileDialog1.Filter = "Bitmap
files(*.jpg)|*.jpg|All files(*.*)|*.*";
                if (openFileDialog1.ShowDialog() ==
DialogResult.OK)
                {
                    System.IO.StreamReader sr = new
System.IO.StreamReader(openFileDialog1.FileName);
                }
                pictureBox1.BackgroundImage =
(Bitmap)Bitmap.FromFile(openFileDialog1.FileName);
               }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            button4.Enabled = false;
            Last.Clear();
            progressBar1.Value = 0;
            button3.Enabled = true;
            label1.Text = "";
            points.Clear();
            Draw();
            dataGridView1.Columns.Clear();
            dataGridView2.Columns.Clear();
            dataGridView1.Columns.Add("steiner", "Исходные
точки");
            dataGridView2.Columns.Add("points", "Точки
Штейнера");
            dataGridView1.Columns[0].Width = 75;
            dataGridView2.Columns[0].Width = 75;
            button2.Enabled = false;        }
private void Draw()
        {            G.Clear(Color.Transparent);
```

Лістинг А.1 – Вибор карти, нанесення термінальних точок

```
            Pen.Color = Color.Red;
            Pen.Width = 2;

            Point P0 = new Point(1, 400);
            Point P1 = new Point(400, 400);
            Point P2 = new Point(1, 1);
            Point P3 = new Point(400, 1);
            G.DrawLine(Pen, P0, P1);
            G.DrawLine(Pen, P2, P3);
            G.DrawLine(Pen, P0, P2);
            G.DrawLine(Pen, P3, P1);
            Pen.Width = 1;
            Pen.Color = Color.FromArgb(50, 0, 0, 0);
            for (int i = 0; i < 401; i = i + 10)
            {
                G.DrawLine(Pen, i, 0, i, 400);
                G.DrawLine(Pen, 0, i, 400, i);
            }
            pictureBox1.Image = myBMP;
        }
        public void DrawPoints()
        {
            Brush pbrush = new SolidBrush(Color.Green);
            foreach (Point P in points)
                G.FillEllipse(pbrush, P.X - 4, -P.Y + 400 - 4,
8, 8);
            pictureBox1.Image = myBMP;
        }
```

Лістинг А.1, лист 2

```
public void AlgMelz(List<Point> IPoin, List<Point>
SteinerPoin, List<Point> DelPoin, List<Point> DopPoin)
        {        if (k++ < 2 * l && k % 2 == 0)
backgroundWorker1.ReportProgress(k / 2);
            if (!backgroundWorker1.CancellationPending)
            {
                List<Point> IPoint = new List<Point>();
                foreach (Point p in IPoin)
                    IPoint.Add(p);
                List<Point> SteinerPoint = new List<Point>();
                foreach (Point p in SteinerPoin)
```

Лістинг А.2 – Алгоритм Мелзака (основний цикл)

```
                    SteinerPoint.Add(p);
                List<Point> DelPoint = new List<Point>();
                foreach (Point p in DelPoin)
                    DelPoint.Add(p);
                List<Point> DopPoint = new List<Point>();
                foreach (Point p in DopPoin)
                    DopPoint.Add(p);
                Point P1 = new Point();
                Point P2 = new Point();
                Point P5 = new Point();
                Point R1 = new Point();
                Point R2 = new Point();
                if (IPoint.Count == 3)
                {
                    P5 = PointF(IPoint[0], IPoint[1],
IPoint[2]);
                    ObrAlgMelz(SteinerPoint, DelPoint,
DopPoint, P5, IPoint[0], IPoint[1], IPoint[2]);
                }
                else
                    for (int i = 0; i < IPoint.Count; i++)
                    {
                 for (int j = i + 1; j < IPoint.Count; j++)
                        {
                            R1 = IPoint[i];
                            R2 = IPoint[j];
                            DelPoint.Add(R1);
                            DelPoint.Add(R2);
                            P1 = X3(IPoint[i], IPoint[j]);
                            P2 = X3(IPoint[j], IPoint[i]);
                            IPoint.Remove(R1);
                            IPoint.Remove(R2);
                            IPoint.Add(P1);
                            DopPoint.Add(P1);
                 AlgMelz(IPoint, SteinerPoint, DelPoint,
DopPoint);
                            IPoint.Remove(P1);
                            DopPoint.Remove(P1);
                            IPoint.Add(P2);
                            DopPoint.Add(P2);
                            AlgMelz(IPoint, SteinerPoint,
DelPoint, DopPoint);
                            IPoint.Remove(P2);
                            DopPoint.Remove(P2);
```

Лістинг А.2, лист 2

```
                        DelPoint.Remove(R1);
                        DelPoint.Remove(R2);
                        IPoint.Add(R1);
                        IPoint.Add(R2);
                    }  }
            }   }
```

Лістинг А.2, лист3

```
   public void ObrAlgMelz(List<Point> SteinerPoin, List<Point>
DelPoin, List<Point> DopPoin, Point DPoin, Point Poin1, Point
Poin2, Point Poin3)
        {
            bool f = false;
            foreach (Point P in points)
            {
                if (DPoin.Y > Y) f = true;
                if (DPoin.X > X) f = true;
                if (DPoin.Y < YY) f = true;
                if (DPoin.X < XX) f = true;
            }
            if (!f)
            {
                //PrintTr(DPoin);
                List<Point> SteinerPoint = new List<Point>();
                foreach (Point p in SteinerPoin)
                    SteinerPoint.Add(p);
                List<Point> DelPoint = new List<Point>();
                foreach (Point p in DelPoin)
                    DelPoint.Add(p);
                List<Point> DopPoint = new List<Point>();
                foreach (Point p in DopPoin)
                    DopPoint.Add(p);
                Point DPoint = new Point();
                DPoint = DPoin;
                Point Point1 = new Point();
                Point1 = Poin1;
                Point Point2 = new Point();
                Point2 = Poin2;
                Point Point3 = new Point();
                Point3 = Poin3;
                Point P3 = new Point();
```

Лістинг А.3 – Алгоритм Мелзака (зворотій хід)

```
                Point P4 = new Point();
                Point P6 = new Point();
                if (!SteinerPoint.Contains(DPoint))
SteinerPoint.Add(DPoint);
                if (DopPoint.Count == 0 && SteinerPoint.Count
== points.Count - 2)
                {
                    List<Point> AllPoints = new List<Point>();
                    foreach (Point p in points)
                        AllPoints.Add(p);
                    foreach (Point p in SteinerPoint)
                        if (!AllPoints.Contains(p))
AllPoints.Add(p);
                    double M = PrimaTr(AllPoints);
                    if (Minim >= M)
                    {
                        Minim = M;
                        Last.Clear();
                        foreach (Point p in AllPoints)
                            Last.Add(p);
                    };
                    AllPoints.Clear();
                }
                else
                {
                    if (DopPoint.Contains(Point3))
                    {
              P3 = DelPoint[DopPoint.IndexOf(Point3) * 2];
            P4 = DelPoint[DopPoint.IndexOf(Point3) * 2 + 1];
                        DopPoint.Remove(Point3);
                        DelPoint.Remove(P3);
                        DelPoint.Remove(P4);
                        P6 = PointF(DPoint, P3, P4);
                    ObrAlgMelz(SteinerPoint, DelPoint,
DopPoint, P6, P3, P4, DPoint);
                    }
                    if (DopPoint.Contains(Point2))
                    {
                 P3 = DelPoint[DopPoint.IndexOf(Point2) * 2];
              P4 = DelPoint[DopPoint.IndexOf(Point2) * 2 + 1];
                        DopPoint.Remove(Point2);
                        DelPoint.Remove(P3);
                        DelPoint.Remove(P4);
                        P6 = PointF(DPoint, P3, P4);
```

Лістинг А.3, лист2

```
  ObrAlgMelz(SteinerPoint, DelPoint, DopPoint, P6, P3, P4,
DPoint);
                    }
                    if (DopPoint.Contains(Point1))
                    {
            P3 = DelPoint[DopPoint.IndexOf(Point1) * 2];
               P4 = DelPoint[DopPoint.IndexOf(Point1) * 2 + 1];
                         DopPoint.Remove(Point1);
                         DelPoint.Remove(P3);
                         DelPoint.Remove(P4);
                         P6 = PointF(DPoint, P3, P4);
                    ObrAlgMelz(SteinerPoint, DelPoint,
DopPoint, P6, P3, P4, DPoint);
                    }                }
             }        }
```

Лістинг А.3, лист 3

```
public Point TFerma(Point A, Point B, Point C)
      {
            double x1 = A.X, y1 = A.Y,
                x2 = B.X, y2 = B.Y,
                x3 = C.X, y3 = C.Y;
            double x4 = (x2 - x1) * 0.5 - (y2 - y1) *
Math.Sqrt(3) / 2 + x1;
            double y4 = (x2 - x1) * Math.Sqrt(3) / 2 + (y2 -
y1) * 0.5 + y1;
            double x5 = (x3 - x2) * 0.5 - (y3 - y2) *
Math.Sqrt(3) / 2 + x2;
            double y5 = (x3 - x2) * Math.Sqrt(3) / 2 + (y3 -
y2) * 0.5 + y2;
            double x, y;
            if ((y1 - y5) * (x4 - x3) - (y3 - y4) * (x5 - x1)
!= 0)
                x = -1 * ((x1 * y5 - x5 * y1) * (x4 - x3) -
(x3 * y4 - x4 * y3) * (x5 - x1)) / ((y1 - y5) * (x4 - x3) -
(y3 - y4) * (x5 - x1));
            else
                x = 0;
            if (x4 - x3 != 0)
                y = (x - x3) * (y4 - y3) / (x4 - x3) + y3;
            else
```

Лістинг А.4 – Алгоритм Мелзака (знаходження точки Ферма)

```
                y = y3;
            R = Math.Sqrt(Math.Pow((x - x1), 2) + Math.Pow((y
- y1), 2)) +
                Math.Sqrt(Math.Pow((x - x2), 2) + Math.Pow((y
- y2), 2)) +
                Math.Sqrt(Math.Pow((x - x3), 2) + Math.Pow((y
- y3), 2));
            return new Point(Convert.ToInt32(x),
Convert.ToInt32(y));
        }
```

Лістинг А.4, лист 2

```
public double PrimaTr(List<Point> Point)
        {
            List<Point> B = new List<Point>();
            int n = Point.Count;
            double min, Min = Minim, M = 0;
            Point rab1 = new Point(), rab2 = new Point();
            for (int i = 0; i < n; i++)
            {
                for (int j = i + 1; j < n; j++)
                {
                    min = Rasst(Point[i], Point[j]);
                    if (min < Min)
                    {
                        rab1 = Point[i];
                        rab2 = Point[j];
                        Min = min;
                    }
                }
            }
            B.Add(rab1);
            B.Add(rab2);
            M = Min;
            while (B.Count < n)
            {
                Min = Minim;
                for (int i = 0; i < n; i++)
                { for (int j = i + 1; j < n; j++)  {
```

Лістинг А.5 – Алгоритм Прима

```
    if ((B.Contains(Point[i]) && !B.Contains(Point[j]))
        || (B.Contains(Point[j]) && !B.Contains(Point[i])))
```

```
                    {
                        min = Rasst(Point[i], Point[j]);
                        if (min < Min)
                        {
                            rab1 = Point[i];
                            rab2 = Point[j];
                            Min = min;
                        }
                    }
                }
            }
            if (!B.Contains(rab1)) B.Add(rab1);
            if (!B.Contains(rab2)) B.Add(rab2);
            M += Min;
            if (M > Minim) return Minim + 1;
        }
        return M;
}
```

Лістинг А.5, лист2