**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по практической работе №1**

**по дисциплине «Основы искусственного интеллекта»**

Тема: Имитация отжига

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9302 |  | Ширнин К.В. |
|  |  | Квитко Д.В. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2022

**Содержание**

[Цель работы 3](#_Toc121424506)

[UML-диаграмма 4](#_Toc121424507)

[Описание классов 4](#_Toc121424508)

[Ход работы 7](#_Toc121424509)

[Пример работы программы 8](#_Toc121424510)

[Вывод 10](#_Toc121424511)

[Список используемой литературы 10](#_Toc121424512)

[Приложение 11](#_Toc121424513)

# Цель работы

Изучить принцип работы алгоритма имитации отжига. Написать ПО с пользовательским графическим интерфейсом, находящее кратчайший маршрут среди всех узлов данного графа, с использованием имитации отжига.

**Теория**

Имитационный отжиг - это вероятностный метод аппроксимации глобального оптимума заданной функции. В частности, это метаэвристика для аппроксимации глобальной оптимизации в большом пространстве поиска для задачи оптимизации. Он часто используется, когда пространство поиска является дискретным (например, проблема коммивояжера, проблема булевой выполнимости, предсказание структуры белка и планирование работы магазина). Для задач, в которых нахождение приблизительного глобального оптимума более важно, чем нахождение точного локального оптимума за фиксированный промежуток времени, имитированный отжиг может быть предпочтительнее точных алгоритмов, таких как градиентный спуск или переход и граница.

Название алгоритма происходит от отжига в металлургии, метода, включающего нагрев и контролируемое охлаждение материала для изменения его физических свойств. Оба являются атрибутами материала, которые зависят от их термодинамической свободной энергии. Нагрев и охлаждение материала влияет как на температуру, так и на термодинамическую свободную энергию или энергию Гиббса. Имитированный отжиг может быть использован для решения очень сложных задач вычислительной оптимизации, где точные алгоритмы терпят неудачу; несмотря на то, что обычно достигается приближенное решение глобального минимума, этого может быть достаточно для многих практических задач.

Это понятие медленного охлаждения, реализованное в алгоритме имитированного отжига, интерпретируется как медленное уменьшение вероятности принятия худших решений по мере изучения пространства решений по формуле:

Принятие худших решений позволяет более тщательно искать глобальное оптимальное решение. В общем случае алгоритмы имитации отжига работают следующим образом. Температура постепенно снижается от начального положительного значения до нуля. На каждом временном шаге алгоритм случайным образом выбирает решение, близкое к текущему путем расчета вероятности по формуле:

Измеряет его качество и переходит к нему в соответствии с зависящими от температуры вероятностями выбора лучшего или худшего решения, которые во время поиска соответственно остаются равными 1 (или положительными) и уменьшаются до нуля. Задача коммивояжёра— одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов.

# UML-диаграмма

На рисунке 1 представлена UML-диаграмма структуры программы

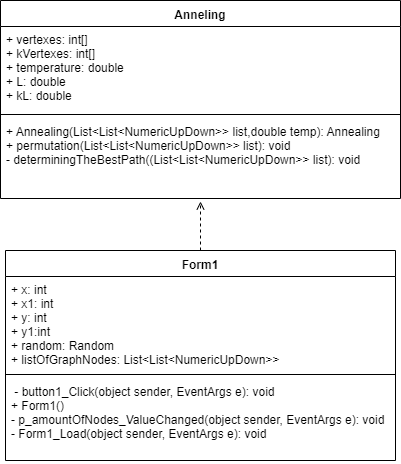


Рис. 1. UML-диаграмма

# Описание классов

Таблица 1 Описание методов класса Annealing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Annealing | - | public | List<List<NumericUpDown>> list,double temp | - | Конструктор |
| permutation() | void | public | List<List<NumericUpDown>> list | - | Делает парную перестановку 2 случайных индексов городов |
| determiningTheBestPath () | void | private | List<List<NumericUpDown>> list | - | Считает длину нового маршрута и сравнивает со старым |

Таблица 2 Описание полей и свойств класса Annealing

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| vertexes | int[] | public | Хранит вершины маршрута |
| kVertexes | int[] | public | Хранит вершины при новой перестановке дистанции |
| temperature | double | public | Температура |
| L | double | public | Длина нового пути |
| kL | double | public | Длина пути до перестановки |

Таблица 3 Описание методов класса Form1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Form1() | - | public | - | - | Конструктор |
| private void button1\_Click() | void | private | object sender, EventArgs e | - | Выполняется алгоритм имитации отжига |
| p\_amountOfNodes\_ValueChanged() | void | private | object sender, EventArgs e | - | Изменяет количество NumericUpDown |
| Form1\_Load() | void | private | object sender, EventArgs e | - | Задает начальные координаты треугольной матрицы |

Таблица 4 Описание полей и свойств класса Form1

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| x | int | private | Начальная координата х |
| x1 | int | private | Промежуточное значение x |
| y | int | private | Начальная координата y |
| y1 | int | private | Промежуточное значение y |
| random | Random | private | Генерирует случайное число |
| listOfGraphNodes | List<List<NumericUpDown>> | private | Хранит веса ребер графа |

# Ход работы

Для выполнения данной лабораторной работы был выбран язык C#.

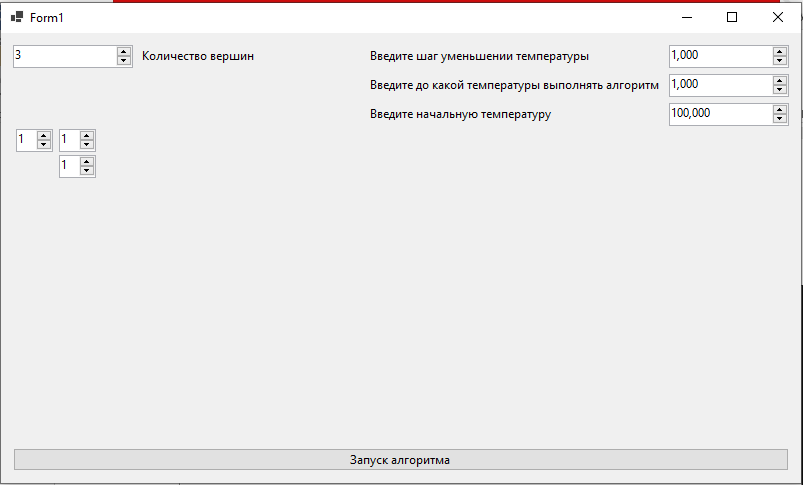


Рис.2. Главное окно приложения

В форме имеются следующие поля для ввода:

* «Количество вершин» – количество вершин
* «Введите шаг уменьшения температуры» – шаг уменьшения температуры
* «Введите до какой температуры выполнять алгоритм» – до какой температуры будет выполняться алгоритм
* «Введите начальную температуру» – Начальная температуру

Для задачи графа была реализована динамическая составляющая пользовательского интерфейса. Пользователю достаточно выставить в поле amount of nodes количество вершин в графе, после чего программа сама отобразит матрицу смежности нужного размера. Она представлена в виде треугольника, т.к. в имитации отжига будет производиться расчет на неориентированном графе (что указывает на симметричность матрицу относительно главной диагонали). Это позволит сэкономить время при вводе данных, а также в два раза сократит расходы на выделяемую память, так как хранение внутри программы производится тем же самым образом.

# Пример работы программы

Пример работы программы представлен на рисунках ниже. В поле Количество вершин вводится количество вершин графа. В нижних ячейках вводится расстояния от одной вершины к другой. В ячейке Шаг уменьшения температуры шаг уменьшения температуры, в ячейке До какой температуры выполняется алгоритм минимальную температуру, в ячейке Начальная температура начальную температуру. При нажатии пользователем кнопки Запуск алгоритма программа находит случайный маршрут и после на каждой итерации меняет случайные 2 ячейки местами.

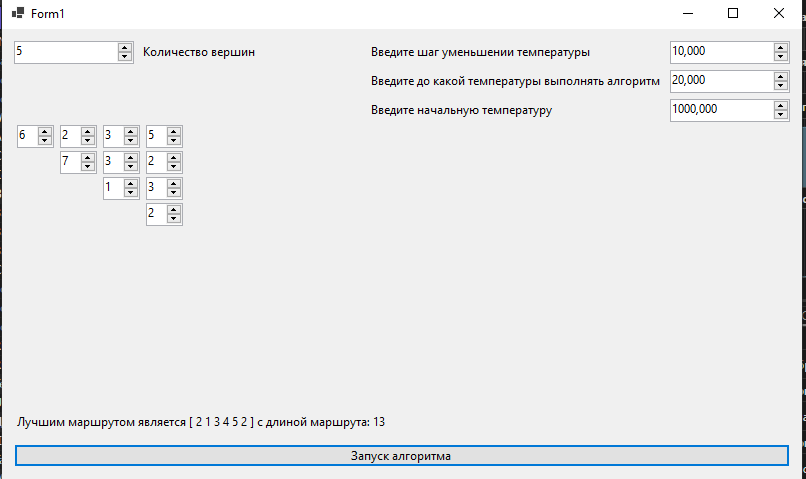


Рис.3.Пример работы программы для 5 вершин

Программа нашла наилучший маршрут, что подтверждает данные со стороннего сайт.

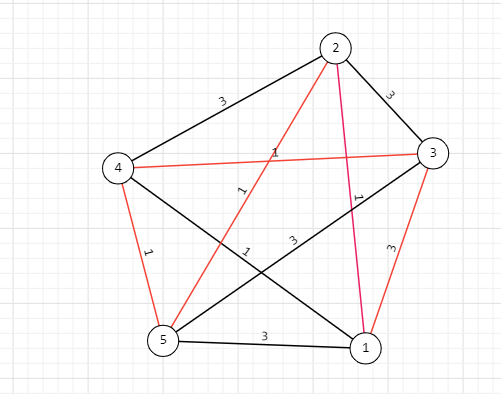


Рис.4.Лучший маршрут вычисленный с помощью стороннего сайта

Еще несколько примеров:

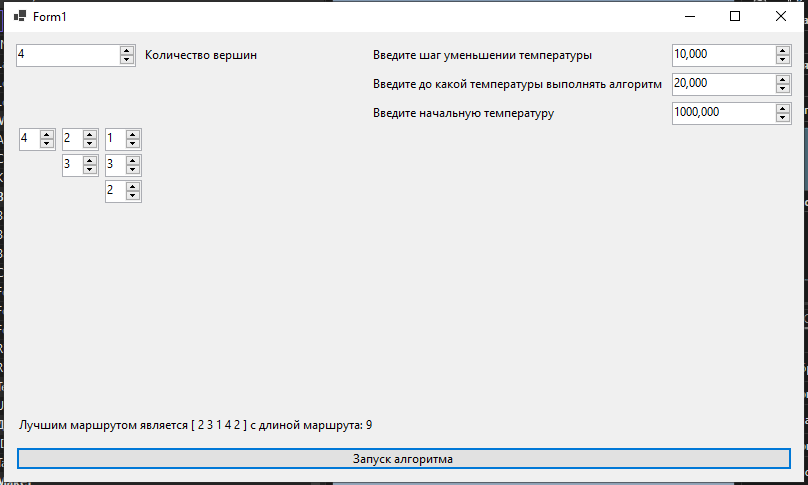


Рис.5.Пример работы программы для 4 вершин

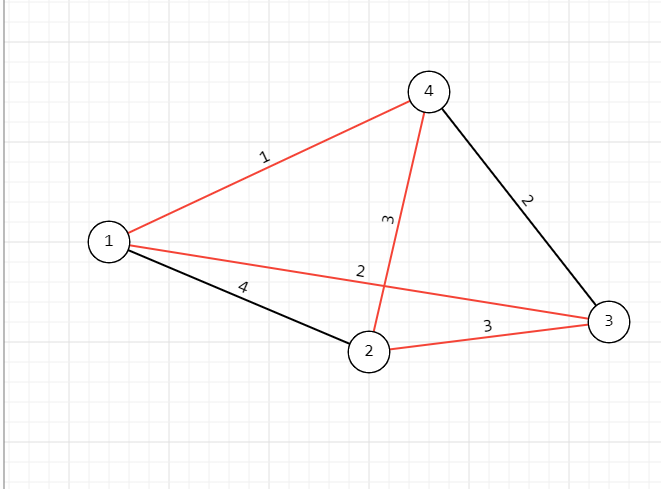


Рис.6.Лучший маршрут вычисленный с помощью стороннего сайта

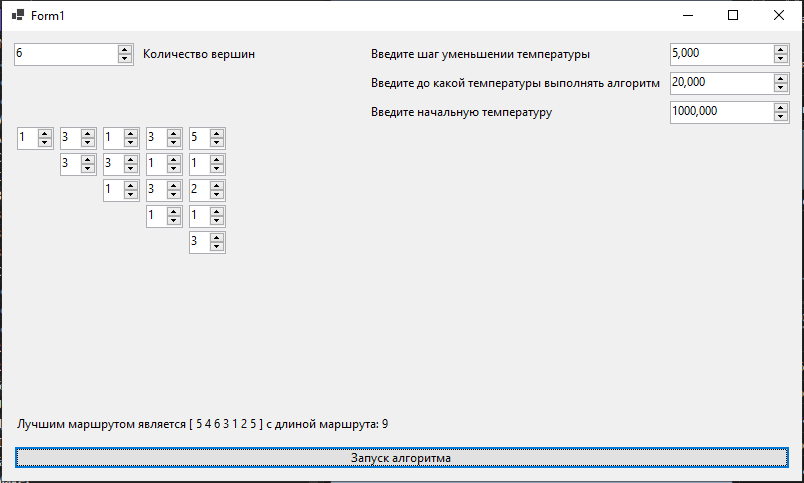


Рис.7. Пример работы программы для 4 вершин

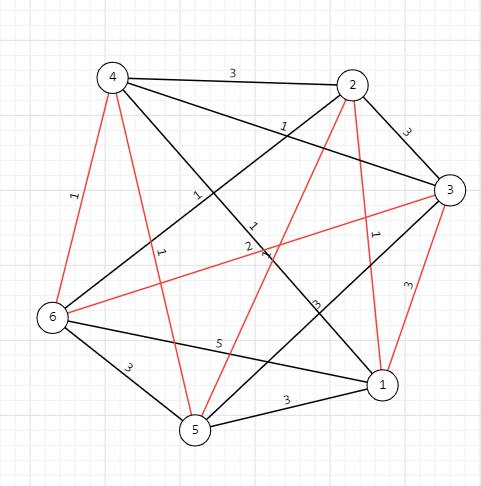


Рис.8. Лучший маршрут вычисленный с помощью стороннего сайта

# Вывод

В ходе выполнения работы стало ясно, что алгоритм имитации отжига очень сильно зависим от заданных параметров. Абсолютно к каждой новой задаваемой матрице смежности требуется подбирать свои индивидуальные параметры. Если подобрать конфигурацию неверно – алгоритм может проложить сильно неверный путь или зайти в тупик, что не имеет никакой практической ценности для решения поставленных задач.

# Список используемой литературы

1. Алгоритм имитации отжига // Википедия URL: <https://goo.su/ApxozO> (дата обращения: 21.09.2022).

# Приложение

**Annealing.cs**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;

namespace SimulatedAnnealing

{

internal class Annealing

{

public int[] vertexes;

public int[] kVertexes;

public double temperature;

public double L,kL;

public Annealing(List<List<NumericUpDown>> list,double temp)

{

Random random = new Random();

vertexes = new int[list.Count + 1];

kVertexes = new int[list.Count + 1];

bool ind = false;

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

{

vertexes[i] = random.Next(1, vertexes.Length + 1);

for (int j = i - 1; j >= 0; j--)

if (vertexes[i] == vertexes[j]) ind = true;

if (ind) i--;

ind = false;

}

temperature = temp;

determiningTheBestPath(list);

kL = L;

kVertexes = vertexes;

}

public void permutation(List<List<NumericUpDown>> list)

{

Random random = new Random();

int i = random.Next(0, vertexes.Length - 1);

int j = i;

while (i == j)

j = random.Next(0, vertexes.Length - 1);

int num = kVertexes[i];

kVertexes[i] = kVertexes[j];

kVertexes[j] = num;

determiningTheBestPath(list);

}

void determiningTheBestPath(List<List<NumericUpDown>> list)

{

L= 0;

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

{

if (vertexes[i + 1] < vertexes[i])

L += (int)list[vertexes[i] - 2][vertexes[i + 1] - 1].Value;

else

L += (int)list[vertexes[i + 1] - 2][vertexes[i] - 1].Value;

}

if (vertexes[0] < vertexes[list.Count])

L += (int)list[vertexes[list.Count] - 2][vertexes[0] - 1].Value;

else

L += (int)list[vertexes[0] - 2][vertexes[list.Count] - 1].Value;

}

}

}

**Form1.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;

namespace SimulatedAnnealing

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

int x, x1, y, y1;

Random random = new Random();

List<List<NumericUpDown>> listOfGraphNodes = new List<List<NumericUpDown>>();

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

{

Annealing annealing = new Annealing(listOfGraphNodes,(double)p\_startTemp.Value);

for (; annealing.temperature > (double)p\_minTemp.Value;annealing.temperature -= (double)p\_temp.Value)

{

annealing.permutation(listOfGraphNodes);

if (annealing.temperature < 0)

annealing.temperature = 1;

if ((100 \* Math.Exp(-(annealing.kL - annealing.L) / annealing.temperature)) > random.Next(1, 100))

{

annealing.kL = annealing.L;

annealing.vertexes = annealing.kVertexes;

}

else

{

annealing.L = annealing.kL;

annealing.kVertexes = annealing.vertexes;

}

}

label3.Text = "Лучшим маршрутом является [ ";

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

{

label3.Text += annealing.vertexes[i] + " ";

}

label3.Text += annealing.vertexes[0] + " ";

label3.Text += "] с длиной маршрута: " + annealing.L;

label3.Visible = true;

}

private void p\_amountOfNodes\_ValueChanged(object sender, EventArgs e)

{

y1 = y;

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

for (int j = 0; j < listOfGraphNodes[i].Count(); j++)

this.Controls.Remove(listOfGraphNodes[i][j]);

listOfGraphNodes.Clear();

for (int i = 1; i < p\_amountOfNodes.Value; i++)

{

x1 = x;

listOfGraphNodes.Add(new List<NumericUpDown>());

for (int j = 0; j < p\_amountOfNodes.Value - 1; j++)

{

if (i > j)

{

listOfGraphNodes[i - 1].Add(new NumericUpDown());

listOfGraphNodes[i - 1][j].Size = new Size(37, 20);

listOfGraphNodes[i - 1][j].Location = new Point(y1, x1);

listOfGraphNodes[i - 1][j].Minimum = 1;

Controls.Add(listOfGraphNodes[i - 1][j]);

x1 += 26;

}

}

y1 += 43;

}

}

private void Form1\_Load(object sender, EventArgs e)

{

x = 96;

y = 15;

p\_amountOfNodes\_ValueChanged(sender, e);

label3.Visible = false;

}

}

}