ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

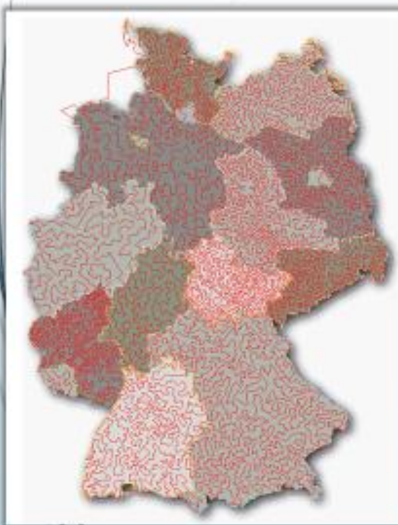
Кафедра ЕОМ

Методи рішення задачі комівояжера на основі обчислювального інтелекту

Виконав:
ст. гр. СПм-22-3
Онищенко О.І.

Науковий керівник:
доц. каф. ЕОМ
Іващенко Г.С.

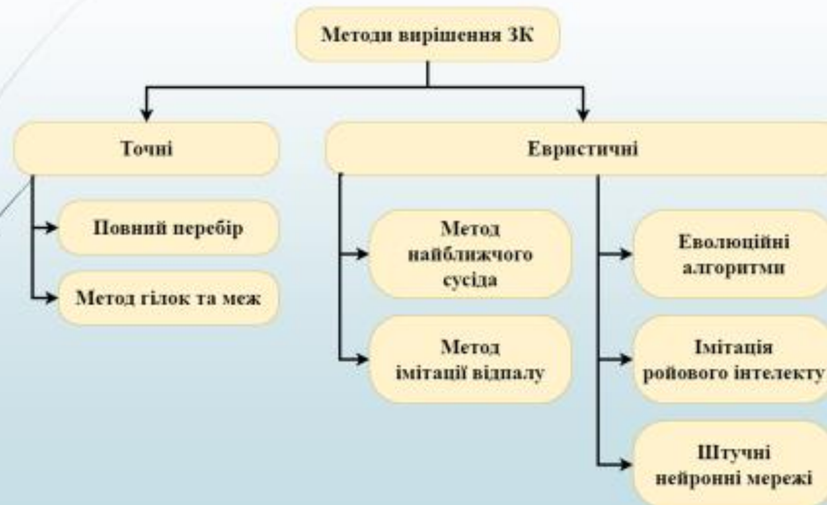
2 Актуальність проблеми



- Задача комівояжера (ЗК) полягає в пошуку найкоротшого можливого маршруту на множині точок (міст), що перетинає кожне місто тільки один раз, після чого повертається в початкову точку шляху.

- ЗК – одна з найвідоміших задач комбінаторної оптимізації, яка є актуальною в сучасному світі через постійне зростання складності процесів та обсягів даних у багатьох сферах діяльності, де ЗК має практичні застосування, серед яких логістика, телекомунікації, обчислювальні системи.

3 Існуючі методи вирішення задачі



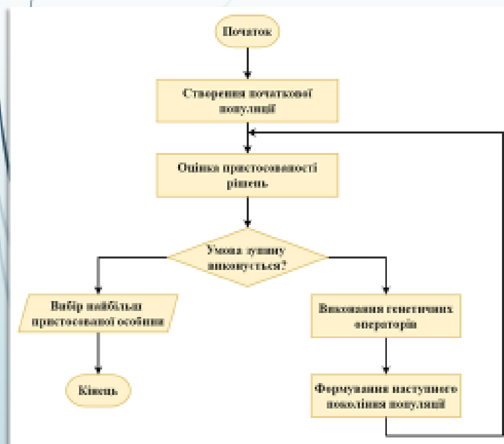
4 Постановка задачі

Метою роботи є дослідження використання таких підходів обчислювального інтелекту, як **генетичні алгоритми**, **мурашині алгоритми** та **самоорганізаційна карта Кохонена** для вирішення задачі комівояжера.

Для проведення експериментів у ході дослідження необхідно було створити програмний застосунок, який буде включати:

- засоби для роботи з графами ЗК, що включають можливість їх завантаження з бібліотеки TSPLIB, або формування випадкових графів із заданої кількістю вершин;
- реалізації алгоритмів для пошуку рішення на завантаженому чи згенерованому графі;
- інструмент автоматизації запуску експериментів для дослідження впливу окремих параметрів налаштувань на ефективність роботи алгоритмів, із можливістю подальшого збереження результатів у файл.

5 Генетичні алгоритми (ГА)



- Є імітацією природних еволюційних процесів та явищ генетики.
- В основі алгоритму лежать популяція особин – множина рішень, які поступово вдосконалюються, та генетичні оператори: схрещування (кросовер), відбір (селекція), мутація.
- Особина популяції – один варіант маршрута, що задовольняє умовам ЗК.
- Генотипом особини є перелік вершин маршруту в порядку їх обходу.
- Показник пристосованості (fitness function) – довжина маршруту, який представляє особина.

6 ГА – генетичні оператори та їх ініціалізація

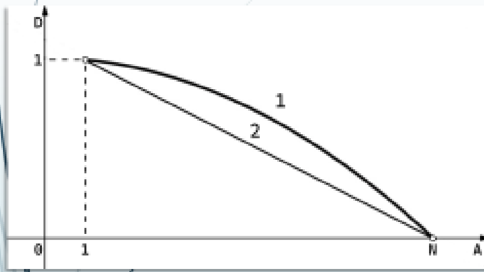
В роботі розглядалися наступні реалізації генетичних операторів:

- Кросовери: order-based (OBX), inver-over (IOX), order (OX), single-point ordered (SpOX), partially-mapped (PMX), cycle (CX);
- Селекції: методом рулетки, турнірний метод;
- Мутації: обміну (swap), зсуву (shift) та інверсії (inversion).

Для операторів передбачені різні підходи до ініціалізації внутрішніх параметрів:

- One time – значення задаються одноразово перед початком виконання алгоритму;
- Every generation – параметри оновлюються для кожного покоління популяції;
- Every individual – нові значення параметрів встановлюються при опрацюванні кожної особини (пари особин) на кожному поколінні ГА.

7 Аналіз збіжності ГА

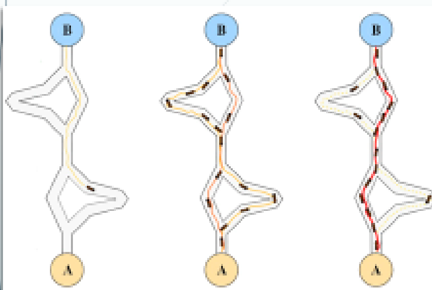


$$D = \frac{A - 1}{N - 1}, A = \overline{1, N}, D \in [0; 1]$$

$$D = \frac{A^2 - 1}{N^2 - 1}, A = \overline{1, N}, D \in [0; 1]$$

- В процесі роботи ГА спостерігається поступове зменшення різноманіття генотипу, поява особин з однаковими маршрутами – відбувається виродження популяції.
- Темпи виродження залежать від ряду чинників – кількості міст, розміру популяції, числа поколінь, ймовірності мутації, особливостей реалізацій операторів ГА та ін.
- З метою аналізу та пошуку методів запобігання передчасної збіжності, запропоновано критерій оцінки – показник виродження популяції **D**, представлений в двох варіантах.

8 Мурашині алгоритми (МА)

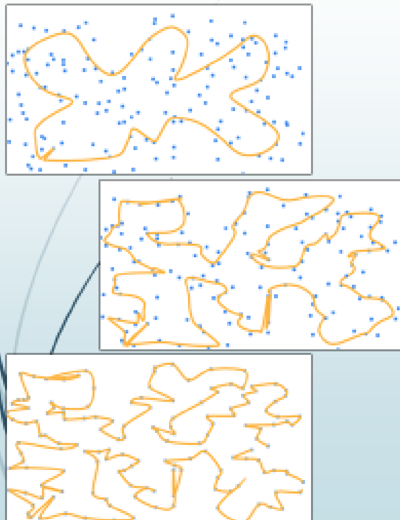


$$p_{ij} = \frac{t_{ij}^{\alpha} + \frac{1}{w_{ij}^{\beta}}}{\sum_{l \in S_i}(t_{ij}^{\alpha} + \frac{1}{w_{ij}^{\beta}})}$$

$$\tau'_{ij} = (1 - \rho)\tau_{ij} + \frac{k}{L}$$

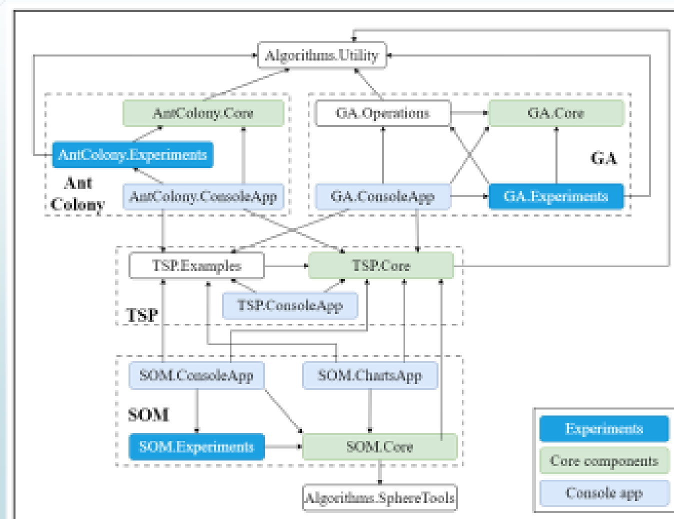
- В основі алгоритму – поведінка однойменних комах, які здатні залишати та відчувати на своєму шляху спеціальну речовину – феромон.
- Найдовші шляхи поступово відсіюються по мірі обходу графу агентами та випаровування феромону.
- На вибір наступної вершини маршруту впливає кількість феромону на переході, довжина переходу та коефіцієнти α і β, що визначають пріоритет вказаних величин.
- Ефективність алгоритму може бути підвищена при використанні елітарних агентів та підходу Min-Max.

9 Самоорганізуюча карта Кохонена (SOM)



- Використання топології у вигляді замкнутого кільця дозволяє інтерпретувати повністю налаштовану мережу як маршрут ЗК.
- Навчання мережі продовжується поки залишаються **неасоційовані нейрони**, для яких відстань до найближчої вершини ЗК перевищує поріг ε.
- Вплив появи відгалужень на маршруті на перших ітераціях зменшується за рахунок використання функції сусідства, яка враховує як відстань між нейронами, так і кількість переходів між ними.
- Для уникнення відгалужень на завершальних етапах навчання, кількість нейронів у мережі може збільшуватися пропорційно до кількості вершин ЗК.

10 Структура застосунку

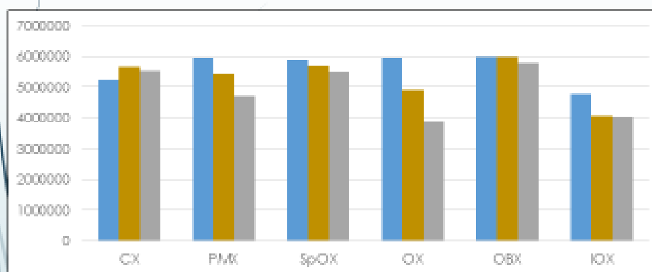


11 Результати досліджень – Збіжність ГА

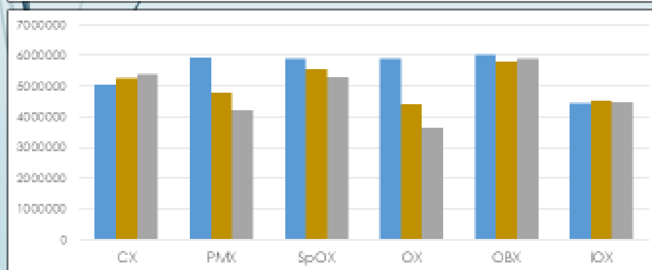
№ п/п	Кросовер	Селекція	Метод иніц.	Кільк. ітерацій	D
1	IOX	Турнір	Every ind.	5000	0.0120
2	IOX	Рулетка	Every ind.	5000	0.0201
3	IOX	Турнір	Every gen.	5000	0.0375
4	IOX	Рулетка	Every gen.	5000	0.0924
5	OBX	Турнір	Every ind.	2536	0.8393
6	OBX	Турнір	Every gen.	2306	0.8755
7	IOX	Турнір	One time	434	0.8568

№ п/п	Кросовер	Селекція	Метод иніц.	Кільк. ітерацій	D
1	IOX	Рулетка	Every ind.	5000	0,0186
2	IOX	Турнір	Every ind.	5000	0,0343
3	IOX	Рулетка	Every gen.	5000	0,0396
4	IOX	Турнір	Every gen.	5000	0,1505
5	OBX	Турнір	Every ind.	2548	0,9050
6	OBX	Турнір	Every gen.	2181	0,9048
7	IOX	Рулетка	One time	597	0,8318

12 Результати досліджень – рішення ГА



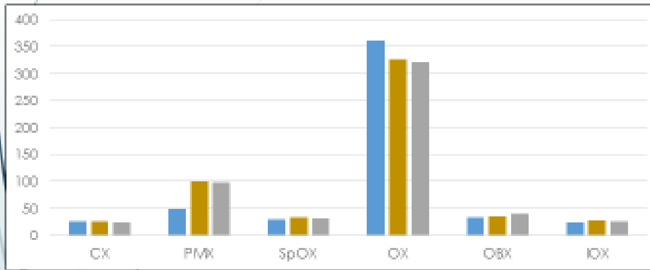
Для селекціїметодом рулетки



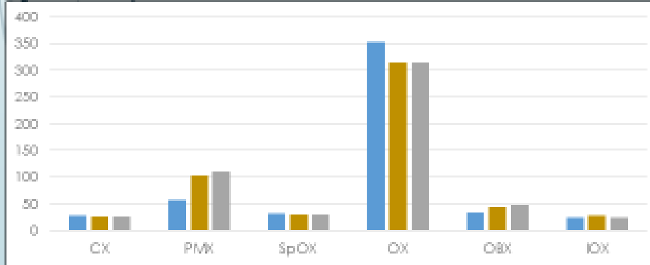
Для селекціїтурнірним методом

One time
Every generation
Every individual

13 Результати досліджень – час роботи ГА



Для селекції методом рулетки

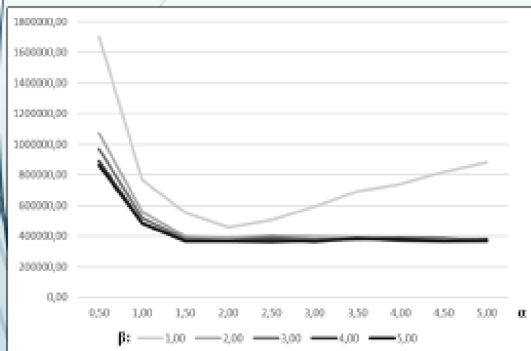


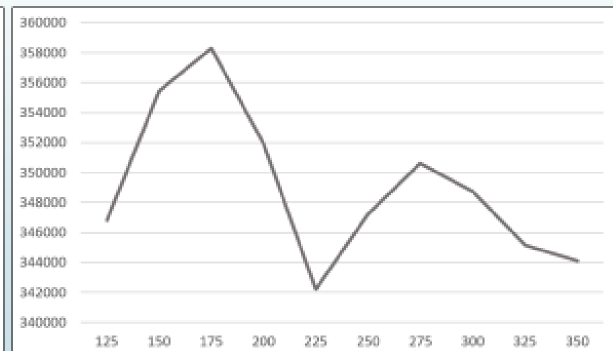
Для селекції турнірним методом

One time
Every generation
Every individual

14 Результати досліджень – МА

- Пошук кращих параметрів α і β
- Налаштування феромону елітних агентів

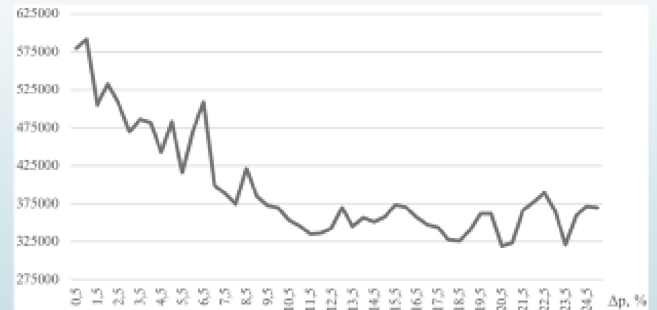




15 Результати досліджень – SOM

- Визначення кращого множника кількості нейронів

- Залежність отримуваного маршруту від темпу зростання штрафних коефіцієнтів



16 Результати досліджень – Підсумки

Граф			tsp225	pcb442	pr1002
L_{best}			3919	50778	259045
ГА	L		4252	58076	342513
	Error, %		8,5	14,37	32,22
	t, c		9	68	2944
МА	AS	L	4968	66595	361885
		Error, %	26,77	31,15	39,7
		t, c	8	32	144
	EA	L	4754	66646	349750
		Error, %	21,31	31,15	35,01
		t, c	6	23	127
	MM	L	4385	62581	320163
		Error, %	11,89	23,24	23,59
		t, c	8	29	137
SOM	L		5514	60681	309255
	Error, %		40,7	19,5	19,38
	t, c		3	8	24

17

Висновки

В ході кваліфікаційної роботи розглянуті підходи до вирішення задачі комівояжера з використанням методів обчислювального інтелекту, таких як генетичні алгоритми, мурашині алгоритми та самоорганізуюча карта Кохонена.

Реалізовано програмний застосунок, що дозволяє запускати на виконання розглянуті алгоритми на графах ЗК, згенерованих випадковим чином або завантажених з бібліотеки TSPLIB, та автоматизувати процес проведення експериментів.

Проведене експериментальне дослідження використання розглянутих підходів для вирішення ЗК. По результатам дослідження, генетичний алгоритм показав найкращі результати на графах розміром 225 та 442 вершини, для графу розміром 1002 вершини найкоротший маршрут знайдено мережею Кохонена. Також SOM виявилася найшвидшим методом пошуку наближених рішень, що має важливе значення при практичному використанні ЗК.

Результати роботи були опубліковані в журналі «Системи управління, навігації та зв'язку».

ДОДАТОК Б

Вихідний код програмного забезпечення

Лістинг 1 – Файл GeneticAlgorithm.cs

```
using GA.Core.Models;
using GA.Core.Operations.Crossovers;
using GA.Core.Operations.Mutations;
using GA.Core.Operations.Selections;
using GA.Core.Utility;
using System;
using System.Collections.Generic;
using System.Linq;
namespace GA.Core
{
public class GeneticAlgorithm<TGene>
{
private ISelection selection;
private ICrossover crossover;
private IMutation mutation;
private IList<Individual<TGene>> population;
Func<Individual<TGene>, double> fitnessGetter;
Dictionary<Individual<TGene>, double> fitnesses;
private (Individual<TGene> Individual, double Fitness)
currentBestResult;
private GASettings settings;
private int iteration = 0;
private int stagnationCounter = 0;
private bool stopCondition = false;
public bool StopCondition { get { return stopCondition; } }
private FitnessSortEnum sort;
public IList<Individual<TGene>> Population =>
population.ToList();
public int Iteration => iteration;
/// <summary>
/// Build algorithm for individuals which distinguish by order
of genes
/// </summary>
/// <param name="genotype"></param>
public GeneticAlgorithm(
ISelection selection,
ICrossover crossover,
IMutation mutation,
IList<Individual<TGene>> population,
GASettings settings,
Func<Individual<TGene>, double> fitnessGetter,
FitnessSortEnum sort = FitnessSortEnum.Descending)
```

```
{
this.selection = selection;
this.crossover = crossover;
this.mutation = mutation;
this.population = population;
fitnesses = new Dictionary<Individual<TGene>, double>();
this.fitnessGetter = fitnessGetter;
this.settings = settings;
this.sort = sort;
foreach (var individual in population)
fitnesses.Add(individual, fitnessGetter(individual));
}
public virtual (Individual<TGene> Individual, double Fitness)
Run()
{
stopCondition = false;
if (settings.GenerationsMaxCount < 1)
throw new ArgumentOutOfRangeException(
nameof(settings.GenerationsMaxCount),
settings.GenerationsMaxCount,
$"{nameof(settings.GenerationsMaxCount)} must be greater than
0");
for (iteration = 0; iteration < settings.GenerationsMaxCount;
iteration++)
{
GetNextGeneration();

if (stopCondition)
break;
}
return currentBestResult;
}
public virtual IList<Individual<TGene>> GetNextGeneration()
{
if (settings.ElitePercent.HasValue)
{
if (settings.ElitePercent < 0D && settings.ElitePercent > 100D)
throw new ArgumentOutOfRangeException(
nameof(settings.ElitePercent),
settings.ElitePercent.Value,
"elite percent must be between 0 and 100");
var populationEliteCount = (int)Math.Ceiling(population.Count *
(settings.ElitePercent.Value / 100));
fitnesses =
(sort == FitnessSortEnum.Ascending ?
fitnesses.OrderByDescending(x => x.Value) : fitnesses.OrderBy(x
=> x.Value))
.Take(populationEliteCount)
.ToDictionary(x => x.Key, x => x.Value);
}
var parentPairs = selection.GetParentPairs(fitnesses, sort);
var children = crossover.GetNextGeneration(parentPairs);
mutation.ProcessMutation<Individual<TGene>, TGene>(children,
```

```
settings.MutationProbability);
if (settings.OnlyChildrenInNewGeneration)
{
population = children;
fitnesses = children.ToDictionary(x => x, x =>
fitnessGetter(x));
}
else
{
foreach (var child in children)
fitnesses.Add(child, fitnessGetter(child));
fitnesses =
(sort == FitnessSortEnum.Ascending ?
fitnesses.OrderByDescending(x => x.Value) : fitnesses.OrderBy(x
=> x.Value))
.Take(population.Count)
.ToDictionary(x => x.Key, x => x.Value);
population = fitnesses.Select(x => x.Key).ToList();
}
var currentIterationBestResult = (sort ==
FitnessSortEnum.Ascending ? fitnesses.OrderByDescending(x =>
x.Value) : fitnesses.OrderBy(x => x.Value)).FirstOrDefault();
var resetStagnation = false;
if ((sort == FitnessSortEnum.Ascending &&
currentIterationBestResult.Value > currentBestResult.Fitness ||
currentBestResult.Fitness == 0D)
 || (sort == FitnessSortEnum.Descending &&
currentIterationBestResult.Value < currentBestResult.Fitness ||
currentBestResult.Fitness == 0D))
{
currentBestResult = (currentIterationBestResult.Key,
currentIterationBestResult.Value);
resetStagnation = true;
}
if (settings.StagnatingGenerationsLimit.HasValue &&
settings.StagnatingGenerationsLimit.Value > 0)
{
if (resetStagnation)
stagnationCounter = 0;
else
stagnationCounter++;

if (stagnationCounter >= settings.StagnatingGenerationsLimit)
stopCondition = true;
}
if (settings.DegenerationMaxPercent.HasValue &&
settings.DegenerationMaxPercent.Value > 0D)
{
var degenerationCoef = population.GetDegenerationCoefficient() *
100D;

if (degenerationCoef > settings.DegenerationMaxPercent.Value)
stopCondition = true;
```

```
}
return population;
}
}
}
```

Лістинг 2 – Файл ClassicAlgorithm.cs

```
using Algorithms.Utility;
using AntColony.Core.Utilities;
using System;
using System.Collections.Generic;
using System.Linq;
namespace AntColony.Core
{
public class ClassicAlgorithm<TNode> : BaseAlgorithm<TNode>
where TNode : class
{
public ClassicAlgorithm(IList<TNode> nodes, Func<TNode, TNode,
double> edgeDistanceGetter, AntColonySettings settings) :
base(nodes, edgeDistanceGetter, settings) { }
public override IList<IList<TNode>> Run(AntPopulationSettings
antSettings)
{
var ants = TravelAllPaths(antSettings.AntCount);
EvaporatePheromones();
foreach (var ant in ants)
{
for (var i = 0; i < ant.TravelledPathMemory.Count - 1; i++)
{
var currentPheromoneAmount =
pheromoneMap[ant.TravelledPathMemory[i]][ant.TravelledPathMemory
[i + 1]];
var pheromoneDelta = settings.UseCommonAntPheromoneAmount ?
settings.CommonAntPheromoneAmount : ant.PersonalPheromoneAmount;
pheromoneDelta /= edgeDistanceGetter(ant.TravelledPathMemory[i],
ant.TravelledPathMemory[i + 1]);
pheromoneMap[ant.TravelledPathMemory[i]][ant.TravelledPathMemory
[i + 1]] = currentPheromoneAmount + pheromoneDelta;
}
}
return ants.Select(x => x.TravelledPathMemory).ToList();
}
protected override void EvaporatePheromones()
{
for (int i = 0; i < nodes.Count; i++)
{
for (int j = 0; j < nodes.Count; j++)
{
if (i == j)
continue;
```

```
var edgeCurrentAmount = pheromoneMap[nodes[i]][nodes[j]];
pheromoneMap[nodes[i]][nodes[j]] = edgeCurrentAmount * (1 -
settings.EvaporationCoefficient);
}
}
}
private protected override void TravelPath(Ant<TNode> ant)
{
//write first node to ant memory
///TODO: always add first or random?
var nodes = this.nodes.ToList();
ant.TravelledPathMemory.Add(nodes[0]);
nodes.RemoveAt(0);
while (nodes.Any())
{
var probabilityIntervals = new List<double>();
//writing intervals:
//0: from 0 to probability[0]
//1: from probability[0] to (probability[0] + probability[1])
//...
//last: from (probability[0] + ... + probability[Count - 2]) to
(probability[0] + ... + probability[Count - 1])
//(probability[0] + ... + probability[Count - 1]) ==
Σ(probabilities)
//Example:
//t(ι1)^α * μ(ι2)^β = 10
//t(ι1)^α * μ(ι3)^β = 12
//t(ι1)^α * μ(ι4)^β = 15
//t(ι1)^α * μ(ι5)^β = 20
//t(ι1)^α * μ(ι6)^β = 19
//Σ(t(ι1)^α * μ(ι6)^β) = 76
//0: from 0                      to 10
| 0 - 10
//1: from (0 + 10)               to (0 + 10 + 12)
| 10 - 22
//2: from (0 + 10 + 12)          to (0 + 10 + 15)
| 22 - 37
//3: from (0 + 10 + 12 + 15)     to (0 + 10 + 12 + 15 + 20)
| 37 - 57
//4: from (0 + 10 + 12 + 15 + 20) to (0 + 10 + 12 + 15 + 20 +
19) | 37 - 76
//generating random value from 0 to 76, i.g. 53
//53 in 37-57 interval, so ant chose "1-5" edge
//if α and β weigths are common for each ant
for (var i = 0; i < nodes.Count; i++)
{
var travelProbabilityPart =
GetTravelProbability(ant.TravelledPathMemory.Last(), nodes[i]);
if (i == 0)
probabilityIntervals.Add(travelProbabilityPart);
else
probabilityIntervals.Add(probabilityIntervals.Last() +
travelProbabilityPart);
```

```
}
var randomValue = Random.Shared.NextDouble() *
probabilityIntervals.Last();
probabilityIntervals.Add(randomValue);
probabilityIntervals.Sort();
var randomValueIndex =
SearchHelper.BinarySearch(probabilityIntervals, randomValue);
if (randomValueIndex != -1)
{
var intervalIndex = (randomValueIndex ==
probabilityIntervals.Count - 1)
? randomValueIndex - 1
: randomValueIndex;
probabilityIntervals.RemoveAt(randomValueIndex);
ant.TravelledPathMemory.Add(nodes[intervalIndex]);
nodes.RemoveAt(intervalIndex);
}
}
}
}
}
```

Лістинг 3 – Файл TwoDimensionalSOM.cs

```
using Algorithms.SphereCovering;
using Algorithms.Utility.StructuresLinking;
using SOM.Configuration;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
namespace SOM
{
public class TwoDimensionalSOM<TPoint2D> : BaseSOM<TPoint2D>
where TPoint2D : IVector<double>
{
public TwoDimensionalSOM(SOMSettings settings, IList<TPoint2D>
dataVectors, Topology topology)
: base(settings, dataVectors, topology)
{
}

protected override void InitNetwork(Topology topology)
{
switch (topology)
{
case Topology.Sphere:
default:
InitSphereNetwork(this.dataVectors.Count);
break;
}
```

```
}
protected override void InitNetwork(bool[,] topology)
{
throw new NotImplementedException();
}
protected override void InitSphereNetwork(int networkSize)
{
networkSize = (int)Math.Round(networkSize *
settings.NetworkSizeMultiplier);
base.InitSphereNetwork(networkSize);
var xValues = this.dataVectors.Select(v => v["x"]);
var yValues = this.dataVectors.Select(v => v["y"]);
var xWidth = xValues.Max() - xValues.Min();
var yWidth = yValues.Max() - yValues.Min();
var centerX = xValues.Average();
var centerY = yValues.Average();
var radius = (xWidth < yWidth)
? xWidth / 100D * settings.NetworkRadiusPercent
: yWidth / 100D * settings.NetworkRadiusPercent;
this.networkVectors = new List<TPoint2D>(networkSize);
this.networkTopologyDistances = new Dictionary<TPoint2D,
Dictionary<TPoint2D, double>>(networkSize);
this.networkReadiness = new Dictionary<TPoint2D,
bool>(networkSize);
var circlePoints = SphereTools.GetCirclePoints(networkSize,
radius, (centerX, centerY));
for (int i = 0; i < networkSize; i++)
{//creating new cell
this.networkVectors.Add(Activator.CreateInstance<TPoint2D>());
this.networkVectors.Last()["x"] = circlePoints[i].X;
this.networkVectors.Last()["y"] = circlePoints[i].Y;
this.networkReadiness.Add(this.networkVectors[i], false);
this.networkTopologyDistances.Add(this.networkVectors[i], new
Dictionary<TPoint2D, double>());
for (int j = 0; j < i; j++)
{
if (i < networkSize / 2 + j)
{
this.networkTopologyDistances[this.networkVectors[i]].Add(this.n
etworkVectors[j], i - j);
this.networkTopologyDistances[this.networkVectors[j]].Add(this.n
etworkVectors[i], i - j);
}
else
{
this.networkTopologyDistances[this.networkVectors[i]].Add(this.n
etworkVectors[j], networkSize - (i - j));
this.networkTopologyDistances[this.networkVectors[j]].Add(this.n
etworkVectors[i], networkSize - (i - j));
}}}
if (settings.UseDistancePenalties)
{
this.networkDistancePenalties = new Dictionary<TPoint2D,
```

```
double>();
foreach (var vector in this.networkVectors)
this.networkDistancePenalties.Add(vector,
networkDistancePenaltiesDefaultValue);
}
this.workingNetworkVectors = this.networkVectors.ToList();
}
public override double GetDistance(TPoint2D first, TPoint2D
second)
{
return Math.Sqrt(Math.Pow(first["x"] - second["x"], 2) +
Math.Pow(first["y"] - second["y"], 2));
}
Func<TPoint2D, TPoint2D, double> getDistanceFunc;
IList<TPoint2D> workingNetworkVectors;
public override void ProcessIteration()
{
if (settings.UseDistancePenalties)
getDistanceFunc = (networkPoint, dataPoint) =>
GetDistance(networkPoint, dataPoint) *
this.networkDistancePenalties[networkPoint];
else
getDistanceFunc = (networkPoint, dataPoint) =>
GetDistance(networkPoint, dataPoint);
foreach (var dataVector in dataVectors.Where(x =>
this.dataReadiness[x] == false)) //except matched ones
{
var orderedNetworkVectors = workingNetworkVectors
.Select(x => (x, getDistanceFunc(x, dataVector)))
.OrderBy(x => x.Item2);
var (closestNetworkVector, distance) =
orderedNetworkVectors.FirstOrDefault();
if (closestNetworkVector != null)
{
if (distance > settings.RoundPrecision)
{
for (var i = 0; i < closestNetworkVector.Count; i++)
closestNetworkVector[i] = closestNetworkVector[i] +
settings.LearningCoefficient * (dataVector[i] -
closestNetworkVector[i]);
}
else
{
for (var i = 0; i < closestNetworkVector.Count; i++)
closestNetworkVector[i] = dataVector[i];
this.networkReadiness[closestNetworkVector] = true;
this.dataReadiness[dataVector] = true;
this.workingNetworkVectors.Remove(closestNetworkVector);
}
if (settings.UseElasticity && settings.CooperationCoefficient >
settings.CooperationThreshold)
{
var networkVectorsToAdjust = workingNetworkVectors.ToList();
```

```
networkVectorsToAdjust.Remove(closestNetworkVector);
var elasticityCoefs = networkVectorsToAdjust.Select(x =>
GetNeighbourFunction(x, closestNetworkVector)).ToArray();
for (int i = 0; i < networkVectorsToAdjust.Count; i++)
{
if (elasticityCoefs[i] > 0D)
for (var j = 0; j < closestNetworkVector.Count; j++)
networkVectorsToAdjust[i][j] = networkVectorsToAdjust[i][j] +
settings.LearningCoefficient *
(dataVector[j] - networkVectorsToAdjust[i][j]) *
elasticityCoefs[i];
}
settings.CooperationCoefficient *= 1D -
settings.CooperationFading / 100D;
}
if (settings.UseDistancePenalties)
this.networkDistancePenalties[closestNetworkVector] *= 1D +
settings.PenaltiesIncreasingCoefficient / 100D; //1% -> newValue
= oldValue * (100% - 1%)
if (settings.LearningFadingCoefficient.HasValue)
settings.LearningCoefficient *= 1D -
settings.LearningFadingCoefficient.Value / 100D;}}}
protected List<TPoint2D> data;
protected override double GetNeighbourFunction(TPoint2D
chosenVector, TPoint2D otherVector)
{
var distance = GetDistance(chosenVector, otherVector);
var pointsDistance =
this.networkTopologyDistances[chosenVector][otherVector];
var coef = Math.Exp(-1 * Math.Pow(pointsDistance, 2) * distance
/ settings.CooperationCoefficient);
return coef;
}
public override IEnumerable<TPoint2D> BuildMap()
{
var stopwatch = Stopwatch.StartNew();

while (!FinishCondition)
{
Console.WriteLine($"{ProcessedVectors} - {stopwatch.Elapsed} |
coef: {settings.LearningCoefficient} | length: {GetFullLength()}
| n: {settings.CooperationCoefficient}");
ProcessIteration();
}
stopwatch.Stop();
Console.WriteLine(stopwatch.Elapsed);
return this.networkVectors.Where(x =>
this.networkReadiness[x]).ToList();
}
}
}
```