# ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Кваліфікаційна робота
на тему:

**Методи рішення задачі маршрутизації транспорту з урахуванням додаткових обмежень**

Виконала:
ст. гр. СПм-23-2 Бондаренко К. В.

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

# Актуальність проблеми

Завдання маршрутизації транспортних засобів (VRP) використовується у системах транспортування, розподілу та логістики. Одним із основних завдань планування транспортних перевезень є створення оптимальних маршрутів зі складу до набору географічно розташованих клієнтів з урахуванням додаткових обмежень.

При вирішенні практичних транспортних завдань планувальникам потрібна система, яка дозволяє точно обчислювати обсяги вантажоперевезень у необхідний час, розраховувати кількість одиниць транспорту, необхідних для забезпечення вантажопотоків, визначати раціональні маршрути руху, а також скоротити сумарні витрати на транспортування.

Тому актуальною є проблема вибору алгоритму пошуку, який забезпечує оптимальні маршрути з урахуванням певних обмежень.

## Задача пошуку оптимальних маршрутів



Для математичного опису завдання CVRPTW можна використовувати орієнтований граф G = (N,E), де N = {0,…,n} – множина вершин, E – множина дуг, що з'єднують вершини графа. V – множина транспортних засобів з однаковою вантажопідйомністю Q.

Цільова функція CVRPTW представлена формулою (1):

$$\min \sum_{k\in V} \sum_{i\in N} \sum_{j\in N} c_{i,j} x_{i,j}^k \quad (1)$$

А обмеження, представлені формулами (2-10):

$$\sum_{k\in V} \sum_{j\in N} x_{i,j}^k = 1,\ i \in \{1, \dots, n\} \quad (2)$$

$$\sum_{i=1}^{n} d_i \sum_{j\in N} x_{i,j}^k \le Q,\ i \in \{1, \dots, n\} \quad (3)$$

$$\sum_{j\in N} x_{0,j}^k = 1, \forall k \in V \quad (4)$$

$$\sum_{i\in N} x_{i,h}^k - \sum_{j\in N} x_{h,j}^k = 0, \forall h \in \{1, \dots, n\}, \forall k \in V \quad (5)$$

$$\sum_{i\in N} x_{i,n}^k = 1, \forall k \in V \quad (6)$$

$$x_{i,j}^k \left(s_i^k + t_{i,j} - s_j^k\right) \le 0, \forall i, j \in N, \forall k \in V \quad (7)$$

$$a_i \le s_i^k \le b_i, \forall i \in N, \forall k \in V \quad (8)$$

$$x_{i,j}^k \in \{0,1\}, \forall i, j \in N, \forall k \in V \quad (9)$$

$$\sum_{k\in V} \sum_{j\in N} x_{0,i}^k \le |V|, \forall j \in N, \forall k \in V \quad (10)$$

де, $C = \{1, \dots, n\}$ – множина клієнтів;

$D = \{1, \dots, d_n\}$ – множина запитів клієнтів;

$(i, j) \in E$ – дуга, що з'єднує вершину i з вершиною j;

$t_{i,j}$ – час перевезення дугою (i, j), з урахуванням часу обслуговування клієнта i;

$c_{i,j}$ – вартість перевезення вантажу від клієнта i до клієнта j;

$[a_i, b_i]$ – часовий проміжок, у який повинен бути обслужений i-ий клієнт;

$s_i^k$ – час прибуття транспортного засобу до клієнту (час відправлення будь-якого транспортного засобу з депо завжди дорівнює 0);

$x_{i,j}^k = \{0,1\}$ – змінна, що характеризує напрямок руху, яка приймає значення 1, якщо транспортний засіб рухається з i до j, та значення 0 транспортний засіб рухається з j до i.

## Постановка задачі

Метою роботи є проведення аналізу наближених алгоритмів рішення задачі маршрутизації транспорту на тестових даних різних розмірів.

Для дослідження роботи алгоритмів необхідно виконати такі завдання:

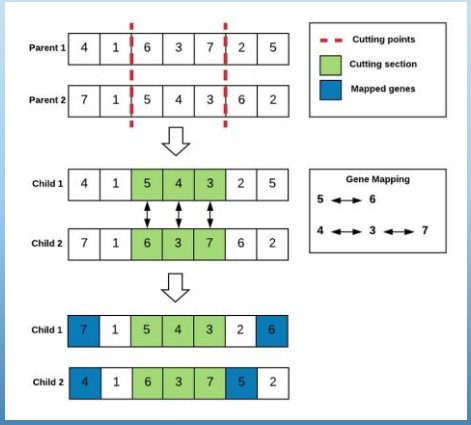
- провести аналіз актуальних наукових досліджень, присвячених вирішенню задач маршрутизації транспорту;
- обрати та проаналізувати поширені наближені алгоритми для рішення VRP;
- визначити необхідні значення параметрів для обраних алгоритмів;
- реалізувати модуль для зберігання та завантаження вхідних і вихідних даних;
- забезпечити можливість запускати серії алгоритмів для вирішення наборів VRP, із можливістю перегляду, аналізу та порівняння отриманих результатів;
- забезпечити можливість візуалізації результатів роботи алгоритмів;
- провести порівняльний аналіз роботи реалізованих алгоритмів.

# Існуючі алгоритми пошуку маршрутів




# Генетичний алгоритм

- Початкова популяція ініціалізується з використанням кількох підходів: жадібний та мурашиний алгоритми, випадкове генерування;
- Селекція – турнірна селекція випадкових особин з популяції для вибору батьків;
- Схрещування (двоточкове) – комбінування генів батьків для створення нащадків;
- Мутація – вставка, обмін між маршрутами та локальна перестановка;
- Оцінка придатності – розрахунок загальної пройденої відстані та балансування робочого навантаження між транспортними засобами.



# Мурашиний алгоритм

Використовується елітарна стратегія – на кожному етапі роботи обирає вершину з найбільшою ймовірністю переходу в неї. Розрахунок ймовірності переходу з поточної вершини у кожну наступну здійснюється за наступною формулою :

$$P_{ij} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{k\in list(i)}\tau_{ik}^{\alpha}\eta_{ik}^{\beta}} \quad (11)$$

де, $P_{ij}$ – ймовірність переходу з пункту i в пункт j;

$\tau$ – кількість феромону на шляху;

$\eta$ – величина зворотна вартості переміщення;

$\alpha$ – коефіцієнт, регулюючий вплив феромонів;

$\beta$ – коефіцієнт, що регулює вплив евристичної інформації.

7

# Алгоритм симуляції відпалу

Шанс переходу до нового стану визначається за формулою (12):

$$P_{s+1} = e^{-\Delta/T} \quad (12)$$

де $P_{s+1}$ – шанс переходу у новий стан;

$\Delta$ – різниця вартості шляхів між новим результатом та попереднім (може бути від'ємною);

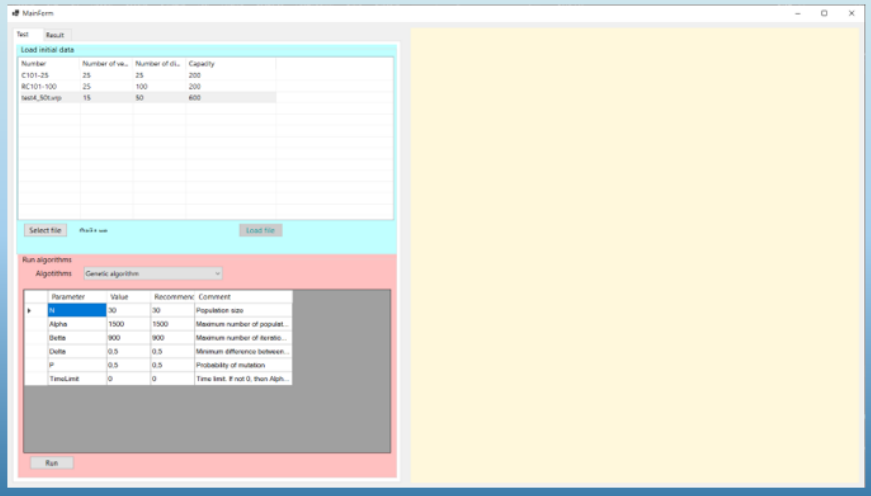
T – «температура» стану рішення.

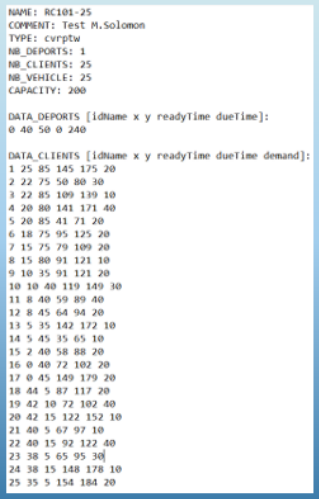
8

# Використані технології та засоби розробки


9

# Інтерфейс програмного забезпечення



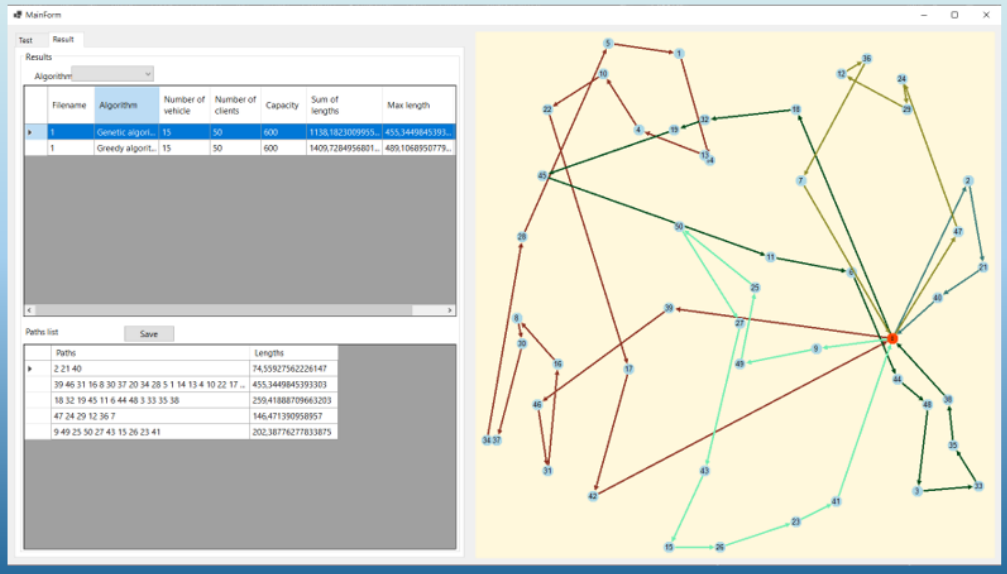
```
NAME: RC101-25
COMMENT: Test M.Solomon
TYPE: cvrptw
NB_DEPORTS: 1
NB_CLIENTS: 25
NB_VEHICLE: 25
CAPACITY: 200

DATA_DEPORTS [idName x y readyTime dueTime]:
0 40 50 0 240

DATA_CLIENTS [idName x y readyTime dueTime demand]:
1 25 85 145 175 20
2 22 75 50 80 30
3 22 85 109 139 10
4 20 80 141 171 40
5 20 85 41 71 20
6 18 75 95 125 20
7 15 75 79 109 20
8 15 80 91 121 10
9 10 35 91 121 20
10 10 40 119 149 30
11 8 40 59 89 40
12 8 45 64 94 20
13 5 35 142 172 10
14 5 45 35 65 10
15 2 40 58 88 20
16 0 40 72 102 20
17 0 45 149 179 20
18 44 5 87 117 20
19 42 10 72 102 40
20 42 15 122 152 10
21 40 5 67 97 10
22 40 15 92 122 40
23 38 5 65 95 30
24 38 15 148 178 10
25 35 5 154 184 20
```

10

# Інтерфейс програмного забезпечення




11

# Результати тестів M. Solomon

| Тест | Кращий відомий розв'язок | Жадібне рішення, од. та с | | Рішення GA, од. та с | | Рішення ACO, од. та с | | Рішення TS, од. та с | | Рішення SA, од. та с | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R101-25 | 617,3 | 623,5 | 0,0072 | 625,2 | 0,72 | 634,4 | 0,059 | 641,06 | 0,086 | 632,5 | 0,057 |
| R101-50 | 1044 | 1065,3 | 0,0449 | 1046,5 | 1,56 | 1087,2 | 0,58 | 1054,4 | 0,9 | 1064,6 | 0,67 |
| R101-100 | 1637,7 | 1786,1 | 0,131 | 1657,37 | 3,2 | 1735,6 | 1,2 | 1687,7 | 1,05 | 1708,2 | 1,28 |
| C101-25 | 191,3 | 193,9 | 0,0066 | 196,5 | 0,083 | 196,5 | 0,0254 | 196,5 | 0,053 | 196,5 | 0,0278 |
| C101-50 | 362,4 | 372,3 | 0,0365 | 364,86 | 1,44 | 371,9 | 0,43 | 366,54 | 0,72 | 377,3 | 0,51 |
| C101-100 | 827,3 | 911,4 | 0,0987 | 833,4 | 2,7 | 885,6 | 0,94 | 842,8 | 0,8 | 865,7 | 1,12 |
| RC101-25 | 461,1 | 463,2 | 0,0068 | 467,35 | 0,098 | 469,8 | 0,054 | 471,12 | 0,074 | 468,81 | 0,054 |
| RC101-50 | 944 | 976,8 | 0,0423 | 950,7 | 1,62 | 987,25 | 0,51 | 957,3 | 0,86 | 984,68 | 0,64 |
| RC101-100 | 1619,8 | 1723,1 | 0,123 | 1649,8 | 2,96 | 1717,68 | 1,1 | 1657,44 | 1,92 | 1698,17 | 1,26 |

12

## Результати тестів Homberger and Gehring

13

| Тест | Кращий відомий розв'язок | Жадібне рішення, од. та с | | Рішення GA, од. та с | | Рішення ACO, од. та с | | Рішення TS, од. та с | | Рішення SA, од. та с | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C1_2_1 | 2704,57 | 3084,8 | 0,1 | 2784,2 | 6,8 | 2861,7 | 2 | 2857,8 | 2,53 | 2883,5 | 2,13 |
| R1_2_1 | 4784,11 | 5231,2 | 0,135 | 4812,4 | 7,23 | 4894,2 | 2,3 | 4835,4 | 2,96 | 4851,6 | 2,6 |
| RC1_2_1 | 3602,80 | 4001 | 0,11 | 3687,3 | 6,97 | 3746,2 | 2,17 | 3721,1 | 2,64 | 3728,8 | 2,24 |
| C1_4_1 | 7152,02 | 7678,1 | 0,15 | 7204,3 | 10,78 | 7326,5 | 3,93 | 7265,6 | 4,97 | 7288,2 | 4,29 |
| R1_4_1 | 10372,31 | 10876,4 | 0,18 | 10465,2 | 12,25 | 10502,2 | 4,21 | 10487,3 | 5,64 | 10528,3 | 4,51 |
| RC1_4_1 | 8571,32 | 9012,3 | 0,13 | 8701,3 | 11,67 | 8764,2 | 4,08 | 8732,6 | 5,27 | 8761,3 | 4,18 |
| C1_6_1 | 14095,64 | 15123,3 | 0,14 | 14356,2 | 12,62 | 14531,3 | 5,58 | 14462,3 | 8,61 | 14498,3 | 7,1 |
| R1_6_1 | 21394,95 | 22678,3 | 0,17 | 21645,5 | 13,43 | 21789,6 | 6,49 | 21712,4 | 9,45 | 21765,2 | 7,96 |
| RC1_6_1 | 16982,86 | 18121,2 | 0,13 | 17214,3 | 12,22 | 17513,2 | 5,75 | 17457,3 | 8,86 | 17487,1 | 6,84 |
| C1_8_1 | 25030,36 | 27568,1 | 0,18 | 25514,2 | 27,46 | 26432,1 | 7,5 | 25764,3 | 14,83 | 25983,1 | 9,4 |
| R1_8_1 | 36767,92 | 38956,3 | 0,24 | 37202,3 | 29,38 | 38365,3 | 8,8 | 37568,6 | 15,6 | 38123,4 | 10,64 |
| RC1_8_1 | 30464,65 | 33791,2 | 0,21 | 30986,6 | 28,43 | 31823,6 | 8 | 31245,8 | 15,03 | 31768,9 | 10,12 |
| C1_10_1 | 42478,95 | 45896,7 | 0,22 | 42968,6 | 90,84 | 44302,1 | 13,2 | 43264,3 | 28,76 | 43897,7 | 18,74 |
| R1_10_1 | 53380,18 | 57987,6 | 0,24 | 53921,7 | 112,6 | 54887,8 | 14,65 | 54179,2 | 32,4 | 54589,3 | 21,4 |
| RC1_10_1 | 45830,62 | 50647,2 | 0,27 | 46234,8 | 105,2 | 47668,6 | 13,85 | 46873,9 | 30,84 | 47102,3 | 20,78 |

## Результати тестів

14

| Розмірність | Жадібне рішення, од. та с | | Рішення GA, од. та с | | Рішення ACO, од. та с | | Рішення TS, од. та с | | Рішення SA, од. та с | |
|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 1062,94 | 0,0054 | 676,52 | 2,75 | 703,24 | 0,29 | 733,41 | 0,49 | 688,32 | 0,379 |
| 70 | 2315,11 | 0,0067 | 1260,52 | 4,67 | 1354,42 | 0,87 | 1265,59 | 1,2 | 1344,28 | 0,94 |
| 100 | 2361,32 | 0,007 | 1376,78 | 8,28 | 1501,37 | 1,34 | 1420,25 | 1,67 | 1465,16 | 1,59 |
| 300 | 5868,37 | 0,0073 | 4542,48 | 17,56 | 4768,26 | 4,26 | 4642,87 | 5,46 | 4702,45 | 4,58 |
| 500 | 16634,26 | 0,0078 | 14736,45 | 28,21 | 14957,21 | 7,63 | 14786,96 | 10,38 | 14824,3 | 8,49 |
| 700 | 27684,39 | 0,0084 | 24678,36 | 37,62 | 25766,98 | 9,77 | 25182,45 | 18,42 | 25436,35 | 12,75 |
| 1000 | 49348,32 | 0,012 | 43679,63 | 116,28 | 44963,87 | 15,48 | 44246,38 | 34,92 | 44629,54 | 23,71 |

# Висновки

Робота присвячена аналізу використання алгоритмів для вирішення задачі CVRPTW, яка полягає у знаходженні оптимальних маршрутів для транспорту з урахуванням обмежень щодо місткості та часових вікон.

На основі проведеного аналізу можна зробити наступні висновки:

- кращим вибором для вирішення задач VRP є наближені методи, які дозволяють уникнути застрягання в локальному оптимумі у процесі пошуку рішень і знаходити розв'язання NP-повних завдань;
- евристичні алгоритми підходять для задач, де важливо швидко отримати рішення, але вони менш ефективні при необхідності адаптації до змін умов;
- метаевристичні методи, демонструють високу ефективність у задачах CVRPTW, особливо у випадках, коли побудова маршруту має виконуватися в реальному часі;
- недоліком метаевристичних алгоритмів є тривалий час роботи на задачах великої розмірності, оскільки випадковий характер пошуку оптимального рішення може вимагати значних витрат часу через велику кількість можливих варіантів.

Результати роботи представлені у рамках Дванадцятої міжнародної науково-технічної конференції «Проблеми інформатизації» та ІІІ Всеукраїнської студентської наукової конференції «Науковий простір: аналіз, сучасний стан, тренди та перспективи».

# ДОДАТОК Б
# ВИХІДНИЙ КОД ЗАСТОСУНКУ

## Б.1 Реалізація алгоритмів пошуку оптимальних маршрутів

### Б.1.1 Реалізація генетичних операторів

```
public static Chromosome Crossover(Chromosome parent1,
Chromosome parent2, CvrptwGraph input, List<Node> allVertex)
        {
            var par1 = parent1;
            var par2 = parent2;
            if (ClassUtils.GenInt(0, 1) == 1)
            {
                var temp = par1;
                par1 = par2;
                par2 = temp;
            }

            int n = par1.Seq.Count;
            int tl = ClassUtils.GenInt(0, n - 1);
            int tr = ClassUtils.GenInt(0, n - 1);

            if (ClassUtils.GenInt(0, 1) == 1) {
                tl = 0;
            }
            else {
                tr = n - 1;
            }

            if (tr < tl) {
                var temp = tl;
                tl = tr;
                tr = temp;
            }

            Array.Fill(used, false);

            List<int> child = new List<int>(new int[n]);

            for (int i = tl; i <= tr; ++i)
            {
                child[i] = par1.Seq[i];
                used[par1.Seq[i]] = true;
            }
```

```
            int curChildInd = (tr + 1) % n;
            for (int i1 = (tr + 1); i1 < tr + n + 1; i1++) {
                int i = i1;
                while (i >= n) {
                    i -= n;
                }

                if (!used[par2.Seq[i]]) {
                    used[par2.Seq[i]] = true;
                    child[curChildInd] = par2.Seq[i];
                    curChildInd++;
                    if (curChildInd >= n) {
                        curChildInd -= n;
                    }
                }
            }

            return new Chromosome(child, input, allVertex);
        }

public static Chromosome Mutation(Chromosome c, double p,
CvrptwGraph input, List<Node> allVertex, int cntImproves = 3)
        {
            double x = ClassUtils.GenDouble(0, 1);
            if (x > p) {
                return c;
            }

            List<int> seq = c.Seq.ToList();
            List<List<int>> paths = SplitPaths(seq, allVertex,
input).Select(path => new List<int>(path)).ToList();

            bool improved = true;

            while (improved && cntImproves > 0)
            {
                improved = false;
                improved |=
Optimizations.GlobalInsertOptimization(paths, allVertex);
                improved |=
Optimizations.GlobalSwapOptimization(paths, allVertex);
                foreach (var path in paths)
                {
                    improved |=
Optimizations.LocalSwapOptimization(path, allVertex);
                }
                cntImproves--;
            }

            return new Chromosome(paths, input, allVertex);
        }

public static Population InitPopulation(CvrptwGraph input, int
```

```
populationSize, double delta, List<Node> allVertex)
        {
            Population population = new Population(delta);

            // Greedy + greedy with local optimizations
            var solGreedy = GreedySolution.Run(input,
ProblemSolution.ProblemMode.MINSUM, new List<double>());
            if (solGreedy.solutionExists)
            {
                var chromosomeGreedy = new Chromosome(solGreedy,
allVertex);
                if (chromosomeGreedy.IsValid())
                {
                    population.Add(chromosomeGreedy);
                    population.Add(Mutation(chromosomeGreedy, 1,
input, allVertex, 60));
                }
            }

            // Fill args for ACO
            List<List<double>> argss = new List<List<double>>
            {
                //{ 3, 4, 0.4, 2, 3, 2, 5, 0.5, 0 },
            };

            // Ant Colony Algo
            foreach (var args in argss)
            {
                if (population.Size() == populationSize)
                {
                    break;
                }
                var sol = ACOSolution.Run(input, args);
                if (!sol.solutionExists)
                {
                    continue;
                }
                var chromo = new Chromosome(sol, allVertex);
                if (chromo.IsValid())
                {
                    population.Add(chromo);
                }
            }

            int leftTries = populationSize * populationSize +
100;
            while (population.Size() < populationSize)
            {
                var chromosomeCur =
GenRandomChromosome(input.NbClients, input, allVertex);

                if (chromosomeCur.IsValid())
                {
```

```
                    if (ClassUtils.GenInt(0, 5) > 0)
                    {
                        population.Add(Mutation(chromosomeCur,
1, input, allVertex, 30));
                    }
                    else
                    {
                        population.Add(chromosomeCur);
                    }
                }

                leftTries--;
                if (leftTries <= 0)
                {
                    //Console.WriteLine($"GA. Error: No tries
left in init population. Number of done chromosomes:
{population.Size()}");
                    if (input.NbClients < 10)
                    {
                        //Console.WriteLine("GA. It's fine for
small inputs");
                    }
                    break;
                }
            }

            foreach (var x in population.Chromosomes)
            {
                Debug.Assert(x.Fitness < ClassUtils.INF -
ClassUtils.EPS);
            }

            //Console.WriteLine("GA. Population initialized");
            return population;
        }
```

### Б.1.2 Реалізація мурашиного алгоритму

```
for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    double cur_dist =
CvrptwGraph.DistanceBetweenNodes(allVertex[i], allVertex[j]);
                    time_reserves[i][j] = allVertex[j].DueTime -
(allVertex[i].ReadyTime + cur_dist);
                    if (i == 0 && allVertex[j].DueTime <
cur_dist) {
                        //Console.WriteLine($"ACO. Vertex #{j}
is not attainable from depot because of its time window");
                        return new ProblemSolution();
                    }
                    dist[i][j] = (time_reserves[i][j] < 0 ?
```

```
ClassUtils.INF : cur_dist);
                    if (j > 0) {
                        candList[i][j - 1] =
Tuple.Create(dist[i][j], j);
                    }
                }
                Array.Sort(candList[i], (a, b) =>
a.Item1.CompareTo(b.Item1));
            }

            if (dist[0].Max() * 2 > input.Capacity) {
                //Console.WriteLine("ACO. There is an element no
attainable because of its distance and Capacity");
                return new ProblemSolution();
            }

            if (dist[0].Max() * 2 > input.Capacity)
            {
                //Console.WriteLine("ACO. There is an element no
attainable because of its distance and Capacity");
                return new ProblemSolution();
            }

            for (int i = 0; i < n; ++i)
            {
                for (int j = 0; j < n; ++j)
                {
                    double a = dist[i][0] + dist[0][j] - (etaG *
dist[i][j]);
                    double b = etaF * Math.Abs(dist[i][0] -
dist[0][j]);
                    double c = a + b;
                    if (Math.Abs(c) < 7e-10)
                    {
                        c = 0;
                    }
                    eta[i][j] = Math.Max(0, c);
                }
            }

            List<List<int>> bestSolution = new
List<List<int>>(n);
            double objectiveFBest = ClassUtils.INF;
            bool[] visited = new bool[n];
            double[] objectiveF = new double[n];

            List<List<int>>[] paths = new List<List<int>>[n];
            for (int i = 0; i < n; i++)
            {
                paths[i] = new List<List<int>>();
                bestSolution.Add(new List<int>());
            }
```

```
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    tau[i][j] = ClassUtils.GenDouble(0.1, 1);
                }
            }

            for (int iteration = 0; iteration < iterations;
++iteration)
            {
                if (input.NbClients < iterations && (iteration
== iterations / 3 || iteration == iterations / 3 * 2))
                {
                    //Console.WriteLine("ACO. Shuffle
pheromones");
                    for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                            tau[i][j] =
ClassUtils.GenDouble(0.1, 1);
                        }
                    }
                }

                if (timeLimit > ClassUtils.EPS &&
(Environment.TickCount - startClock) / 1000.0 > timeLimit)
                {
                    //Console.WriteLine("ACO. Time limit
finished. break");
                    break;
                }

                for (int i = 0; i < n; ++i)
                {
                    for (int j = 0; j < n; ++j)
                    {
                        p[i][j] = ClassUtils.PowInt(tau[i][j],
alpha) * ClassUtils.PowInt(eta[i][j], betta);
                    }
                }

                for (int i = 0; i < n; ++i)
                {
                    paths[i].Clear();
                }

                Array.Fill(objectiveF, 0);
                for (int antIndex = 1; antIndex < n; ++antIndex)
{
                    Array.Fill(visited, false);
                    visited[antIndex] = true;
                    visited[0] = true;
                    double curPathLen = dist[0][antIndex];
                    paths[antIndex] = new List<List<int>> { new
List<int> { 0, antIndex } };
```

```
                    double cur_time =
Math.Max(dist[0][antIndex], allVertex[antIndex].ReadyTime);
                    int cntVisitedTargets = 1;
                    int lastVisited = 0;

                    while (cntVisitedTargets < n - 1) {
                        if ((cntVisitedTargets == lastVisited &&
paths[antIndex].Last().Last() != 0) || cntVisitedTargets <
lastVisited) {
                            throw new Exception("Error");
                        }
                        lastVisited = cntVisitedTargets;
                        int v = paths[antIndex].Last().Last();
                        double probDenominator = 0;

                        int checkListSz = Math.Max(1, Math.Min(n
- 1, (int)(n * candListCoef)));
                        int cntToCheck = v == 0 ? n - 1 :
checkListSz;

                        for (int i = 0; i < cntToCheck; ++i) {
                            int u = candList[v][i].Item2;
                            if (!visited[u] && dist[v][i] <
ClassUtils.INF - 1)
                            {
                                probDenominator += p[v][u];
                                if (probDenominator >
ClassUtils.EPS && v == 0 && i >= cntToCheck)
                                {
                                    break;
                                }
                            }
                        }

                        double tmp = probDenominator >
ClassUtils.EPS ? ClassUtils.GenDouble(0, probDenominator) : 0;
                        int nextV = -1;
                        for (int i = 0; i < cntToCheck && nextV
== -1; ++i) {
                            int u = candList[v][i].Item2;
                            if (!visited[u]) {
                                tmp -= p[v][u];
                                if (tmp < ClassUtils.EPS) {
                                    nextV = u;
                                }
                            }
                        }

                        // if nextV is found and addidng edge
(v, nextV) is feasible
                        if (nextV != -1 && curPathLen +
dist[v][nextV] + dist[nextV][0] <= input.Capacity &&
                            Math.Max(cur_time,
```

```
allVertex[v].ReadyTime) + dist[v][nextV] <=
allVertex[nextV].DueTime) {
                            visited[nextV] = true;
                            curPathLen += dist[v][nextV];
                            paths[antIndex].Last().Add(nextV);
                            cntVisitedTargets++;
                            cur_time = Math.Max(cur_time +
dist[v][nextV], allVertex[nextV].ReadyTime);
                        }
                        else {
                            if (paths[antIndex].Last().Count <=
1) {
                                throw new
InvalidOperationException("Path should contain at least 2
vertices.");
                            }
                            curPathLen += dist[v][0];
                            objectiveF[antIndex] += curPathLen;
                            paths[antIndex].Add(new List<int> {
0 });
                            curPathLen = 0;
                            cur_time = 0;
                        }
                    }
                    curPathLen +=
dist[paths[antIndex].Last().Last()][0];
                    objectiveF[antIndex] += curPathLen;
                }

                for (int antIndex = 1; antIndex < n; ++antIndex)
{
                    antPathLen[antIndex - 1] =
Tuple.Create(objectiveF[antIndex], antIndex);
                }
                Array.Sort(antPathLen, 0, n - 1,
Comparer<Tuple<double, int>>.Create((x, y) =>
x.Item1.CompareTo(y.Item1)));


                for (int i = 0; i < sigma; ++i) {
                    antPathLen[i] = Tuple.Create(0.0,
antPathLen[i].Item2);
                    foreach (var path in
paths[antPathLen[i].Item2]) {

Optimizations.LocalSwapOptimization(path, allVertex);
                        for (int j = 0; j < path.Count - 1; ++j)
{
                            Debug.Assert(dist[path[j]][path[j +
1]] < ClassUtils.INF - 1);
                            antPathLen[i] =
Tuple.Create(antPathLen[i].Item1 + dist[path[j]][path[j + 1]],
antPathLen[i].Item2);
```

```
                        }
                        antPathLen[i] =
Tuple.Create(antPathLen[i].Item1 + dist[path.Last()][0],
antPathLen[i].Item2);

                    }
                }

                for (int i = 0; i < sigma; ++i) {

Optimizations.GlobalSwapOptimization(paths[antPathLen[i].Item2],
allVertex);

Optimizations.GlobalInsertOptimization(paths[antPathLen[i].Item2
], allVertex);
                    antPathLen[i] = Tuple.Create(0.0,
antPathLen[i].Item2);
                    foreach (var path in
paths[antPathLen[i].Item2]) {
                       Optimizations.LocalSwapOptimization(path,
allVertex);
                        for (int j = 0; j < path.Count - 1; ++j)
{
                            Debug.Assert(dist[path[j]][path[j +
1]] < ClassUtils.INF - 1);
                            antPathLen[i] =
Tuple.Create(antPathLen[i].Item1 + dist[path[j]][path[j + 1]],
antPathLen[i].Item2);

                        }
                        antPathLen[i] =
Tuple.Create(antPathLen[i].Item1 + dist[path.Last()][0],
antPathLen[i].Item2);

                    }
                }

                Array.Sort(antPathLen, 0, sigma,
Comparer<Tuple<double, int>>.Create((x, y) =>
x.Item1.CompareTo(y.Item1)));

                for (int i = 0; i < n; ++i)
                {
                    for (int j = 0; j < n; ++j)
                    {
                        tau[i][j] *= rho;
                    }
                }

                for (int mu = 1; mu < sigma; ++mu)
                {
                    foreach (var path in paths[antPathLen[mu -
```

```
1].Item2]) {
                        for (int i = 1; i < path.Count; ++i)
                        {
                            tau[path[i - 1]][path[i]] += (sigma
- mu) / antPathLen[mu - 1].Item1;
                        }

                    }

                }

                if (antPathLen[0].Item1 < objectiveFBest &&
paths[antPathLen[0].Item2].Count <= input.NbVehicle)
                {
                    bestSolution =
paths[antPathLen[0].Item2].Select(path => new
List<int>(path)).ToList();
                    objectiveFBest = antPathLen[0].Item1;
                }

                foreach (var path in bestSolution)
                {
                    for (int i = 1; i < (int)path.Count; ++i)
                    {
                        tau[path[i - 1]][path[i]] += sigma /
objectiveFBest;
                    }
                }

                if (iteration % 20 == 0 && timeLimit <
ClassUtils.EPS)
                {
                    //Console.WriteLine($"ACO. Iteration:
{iteration}/{iterations}. Best found value: {objectiveFBest}");

                }
                else if (iteration % 20 == 0)
                {
                    //Console.WriteLine($"ACO. Iteration done:
#{iterations}. Best found value: {objectiveFBest}");
                }

            }
```

### Б.1.3 Реалізація табу пошуку

```
public static ProblemSolution Run(CvrptwGraph input,
List<double> args)
        {
            CreateInitialSolution(allVertex, distanceMatrix,
input);
```

```
            double cost = input.Vehicles.Sum(v => v.cost);
            tabuListSet_.Clear();
            for (int i = 0; i < nTabu_; i++)
            {
                tabuListQueue_.Enqueue((0, 0));
            }

            var bestVehicles = new
List<Vehicle>(input.Vehicles).ToList();
            bestCost_ = cost;
            newCost_ = cost;

            for (int cIt = 0; cIt < maxIt_; cIt++)
            {
                int bestC = -1, bestR = -1;
                double delta = double.MaxValue;

                Vehicle vTemp = null;
                Vehicle vTemp2 = null;

                foreach (var v in input.Vehicles)
                {
                    for (int cur = 1; cur < v.nodes.Count - 1;
cur++)
                    {
                        int vCur = v.nodes[cur];
                        int vPrev = v.nodes[cur - 1];
                        int vNextC = v.nodes[cur + 1];

                        double costReduction =
distanceMatrix[vPrev][vNextC] - distanceMatrix[vPrev][vCur] -
distanceMatrix[vCur][vNextC];
                        bool isTabu1 = IsTabu((vPrev, vCur),
tabuListSet_) || IsTabu((vCur, vPrev), tabuListSet_) ||
                            IsTabu((vCur, vNextC), tabuListSet_)
|| IsTabu((vNextC, vCur), tabuListSet_);

                        foreach (var v2 in input.Vehicles)
                        {
                            double dist = 0;
                            for (int rep = 0; rep <
v2.nodes.Count - 1; rep++)
                            {
                                int vRep = v2.nodes[rep];
                                int vNextR = v2.nodes[rep + 1];

                                if (vRep != vCur && (v.id !=
v2.id || vRep != vPrev))
                                {
                                    bool isTabu2 = IsTabu((vRep,
vCur), tabuListSet_) || IsTabu((vCur, vNextR), tabuListSet_);

                                    dist +=
```

```
Math.Max(distanceMatrix[vRep][vNextR],
allVertex.FirstOrDefault(n => n.Id == vNextR).ReadyTime);
                                    double costIncrease =
distanceMatrix[vRep][vCur] + distanceMatrix[vCur][vNextR] -
distanceMatrix[vRep][vNextR];

                                    if ((costIncrease +
costReduction < delta) && (v2.cost + costIncrease <=
input.Capacity) &&
                                        (Math.Max(dist +
costIncrease, allVertex.FirstOrDefault(n => n.Id ==
vCur).ReadyTime)
                                        <=
allVertex.FirstOrDefault(n => n.Id == vCur).DueTime || v.id ==
v2.id) &&
                                        (!(isTabu1 || isTabu2)
|| Aspiration(costIncrease, costReduction, newCost_,
bestCost_))) {
                                        delta = costIncrease +
costReduction;
                                        bestC = cur;
                                        bestR = rep;
                                        vTemp = v;
                                        vTemp2 = v2;
                                    }
                                }
                            }
                        }


                    }
                }
                if (delta == double.MaxValue || vTemp == null ||
vTemp2 == null)
                {
                    Console.WriteLine($"На итерации {cIt},
перемещения не найдены. Возможно, стоит изменить размер табу-
списка.");
                    break;
                }
                int valBestC = vTemp.nodes[bestC];
                vTemp.nodes.RemoveAt(bestC);
                vTemp.CalculateCost(distanceMatrix, allVertex);

                if (vTemp.id == vTemp2.id && bestC < bestR)
                {
                    vTemp2.nodes.Insert(bestR, valBestC);
                    tabuListSet_.Add((vTemp2.nodes[bestR - 1],
valBestC));
                    tabuListQueue_.Enqueue((vTemp2.nodes[bestR -
1], valBestC));
                }
                else
```

```
                {
                    vTemp2.nodes.Insert(bestR + 1, valBestC);
                    tabuListSet_.Add((vTemp2.nodes[bestR],
valBestC));
                    tabuListQueue_.Enqueue((vTemp2.nodes[bestR],
valBestC));
                }

                vTemp2.CalculateCost(distanceMatrix, allVertex);

                newCost_ = input.Vehicles.Sum(v => v.cost);

                if (newCost_ < bestCost_)
                {
                    bestVehicles = input.Vehicles.ToList();
                    bestCost_ = newCost_;
                }

                while (tabuListQueue_.Count > nTabu_)
                {
                    var expired = tabuListQueue_.Dequeue();
                    tabuListSet_.Remove(expired);
                }
            }

            input.Vehicles.Clear();
            input.Vehicles.AddRange(bestVehicles);

            cost = input.Vehicles.Sum(v => v.cost);
            Console.WriteLine($"TOTAL COST: {cost}");

        }
```

### Б.1.4 Реалізація симуляції відпалу

```
public static ProblemSolution Run(CvrptwGraph input,
List<double> args)
        {
            CreateInitialSolution(allVertex, distanceMatrix,
input);


            double cost = input.Vehicles.Sum(v => v.cost);
            var bestVehicles = new
List<Vehicle>(input.Vehicles).ToList();
            double bestCost = cost;
            double currentCost = cost;

            for (int r = 0; r < reheats; r++)
            {
                int stag = stagLimit;
```

```
                double temperature = initTemp;

                while (--stag >= 0)
                {
                    temperature *= coolingRate;
                    var random = new Random();
                    Vehicle v1 =
input.Vehicles[random.Next(input.NbVehicle)];
                    Vehicle v2 =
input.Vehicles[random.Next(input.NbVehicle)];
                    double dist = 0;

                    if (v1.nodes.Count <= 2) continue;

                    int cur = random.Next(1, v1.nodes.Count -
1);
                    int rep = random.Next(v2.nodes.Count - 1);

                    if (v1.id == v2.id && (cur == rep + 1 || cur
== rep)) continue;

                    int prev = cur - 1;
                    int nextC = cur + 1;
                    int nextR = rep + 1;

                    double costReduction =

distanceMatrix[v1.nodes[prev]][v1.nodes[nextC]] -

distanceMatrix[v1.nodes[prev]][v1.nodes[cur]] -

distanceMatrix[v1.nodes[cur]][v1.nodes[nextC]];

                    double costIncrease =

distanceMatrix[v2.nodes[rep]][v1.nodes[cur]] +

distanceMatrix[v1.nodes[cur]][v2.nodes[nextR]] -

distanceMatrix[v2.nodes[rep]][v2.nodes[nextR]];

                    double delta = costIncrease + costReduction;

                    dist +=
Math.Max(distanceMatrix[v2.nodes[rep]][v2.nodes[nextR]],
allVertex[v2.nodes[nextR]].ReadyTime);

                    if ((v2.cost + costIncrease <=
input.Capacity) &&
                                            (Math.Max(dist +
costIncrease, allVertex[v1.nodes[cur]].ReadyTime)
                                            <=
allVertex[v1.nodes[cur]].DueTime || v1.id == v2.id) &&
```

```
AllowMove(delta, temperature)) {
                        v1.cost += costReduction;
                        v2.cost += costIncrease;

                        int val = v1.nodes[cur];
                        v1.nodes.RemoveAt(cur);
                        if (v1.id == v2.id && cur < rep)
                            v2.nodes.Insert(rep, val);
                        else
                            v2.nodes.Insert(rep + 1, val);

                        currentCost += delta;
                    }

                    if (currentCost < bestCost)
                    {
                        stag = stagLimit;
                        bestVehicles = input.Vehicles.ToList();
                        bestCost = currentCost;
                    }
                }
            }
            input.Vehicles.Clear();
            input.Vehicles.AddRange(bestVehicles);
            cost = input.Vehicles.Sum(v => v.cost);
            Console.WriteLine($"Cost: {cost}");

    }
```