ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Кваліфікаційна робота
на тему:

Гібридний метод рішення задачі маршрутизації транспорту з урахуванням додаткових обмежень

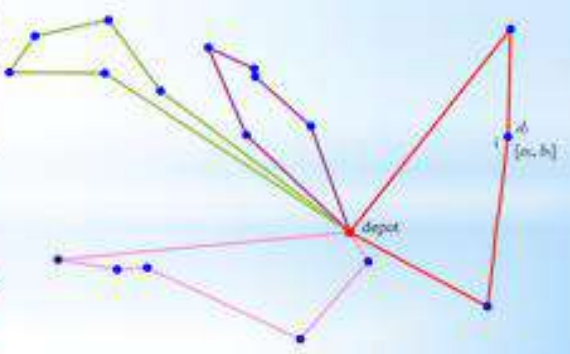
Виконав:
ст. гр. СПм-21-2
Скляров А. С.

Керівник:
доц. каф. ЕОМ
Іващенко Г. С.

Актуальність проблеми

В умовах реального часу зменшення витрат на доставку вантажів є актуальною проблемою для великої кількості підприємств. Зі збільшенням обсягів виробництва зростає складність розподілу ресурсів, транспортні засоби можуть використовуватись неефективно. У зв'язку з цим в області транспортної логістики є потреба у застосуванні автоматизованих систем для вирішення ЗМТ, з метою заощадження ресурсів.

У класичному варіанті ЗМТ є необмежений парк ідентичних транспортних засобів і кінцева множина споживачів. Потрібно побудувати такий набір маршрутів, щоб усі клієнти були обслужені, а сумарна пройдена відстань була мінімальною. Кожен маршрут починається та закінчується в депо.



Аналіз існуючих рішень

3

Постановка задачі

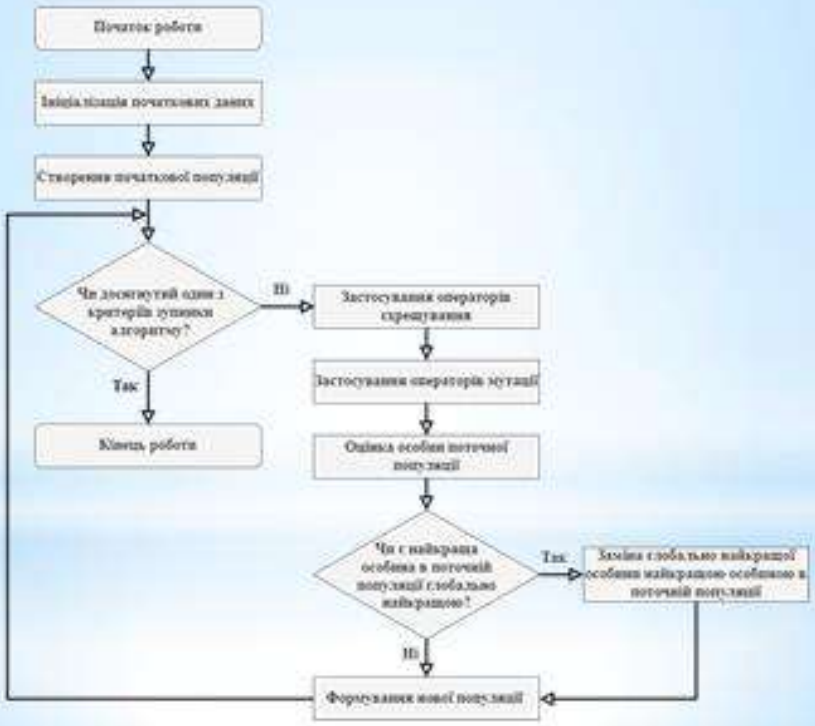
Метою роботи є розробка гібридних методів вирішення ЗМТ з урахуванням обмежень на вантажопідйомність та часових вікон, а також програмна реалізація розроблених методів та проведення експериментальних досліджень.

Для тестування та дослідження алгоритмів необхідно створити програмний застосунок, який дозволить:

- генерувати ЗМТ з заданими параметрами;
- налаштовувати параметри розроблених алгоритмів та функції оцінки рішення;
- запускати набори налаштованих алгоритмів для вирішення наборів ЗМТ, з можливістю подальшого перегляду та порівняння отриманих результатів роботи;
- отримувати детальну інформацію про внутрішню роботу обраного алгоритму.

4

Генетичний алгоритм



5

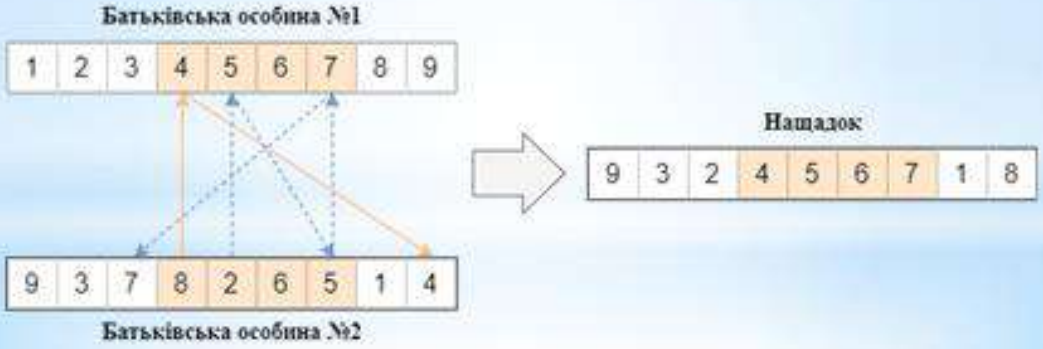
Використані алгоритми для генерації початкової популяції

- **Жадібний алгоритм** - заснований на виборі найкращого локального рішення на кожному етапі пошуку глобального рішення.
- **Модифікація жадібного алгоритму** - заснована на максимізації завантаження транспорту вантажем, не враховуючи вартості переміщень.
- **Алгоритм збережень (метод Кларка-Райта)** - заснований на процесі злиття коротших маршрутів у більш довгі, який проводиться доки є можливість зменшити сумарну вартість об'їзду.
- **Метод гілок з відсіканням** - заснований на частковому переборі множини допустимих рішень з відсіюванням підмножин, що завідомо не містять найкоротшого рішення.

6

Кросинговер

В роботі використаний PMX-кросинговер (англ. Partially Mapped Crossover) - метод схрещування, що створює нащадків шляхом вибору частини послідовності елементів від одного батька та зберігає порядок і положення якомога більшої кількості елементів від іншого батька. Частина послідовності елементів обирається шляхом вибору двох випадкових точок розрізу (які є межами обраної послідовності).



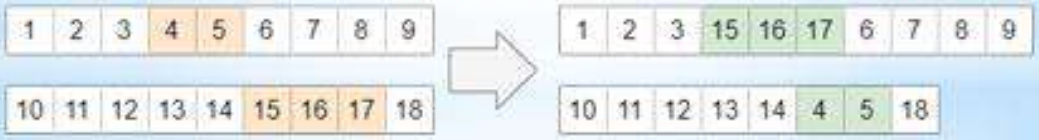
7

Мутація

Інверсна мутація:

Зсув підмножини вершин в межах одного маршруту:

Обмін підмножинами вершин між різними маршрутами одного рішення:



8

Оцінка знайдених рішень

Для оцінки знайдених рішень використовується фітнес-функція (1), яка враховує сумарну вартість переміщень, наповненість вантажем транспортних засобів, дотримання часових вікон та кількість маршрутів.

$$f = c_{заг.} C_{коеф.} + d_{заг.} D_{коеф.} + mM_{коеф.} + kK_{коеф.} \quad (1)$$

де $C_{коеф.}$, $D_{коеф.}$, $M_{коеф.}$, $K_{коеф.}$ - коефіцієнти, що впливають на значимість характеристик отриманого рішення;

$c_{заг.}$ - загальна вартість переміщень у рішенні;

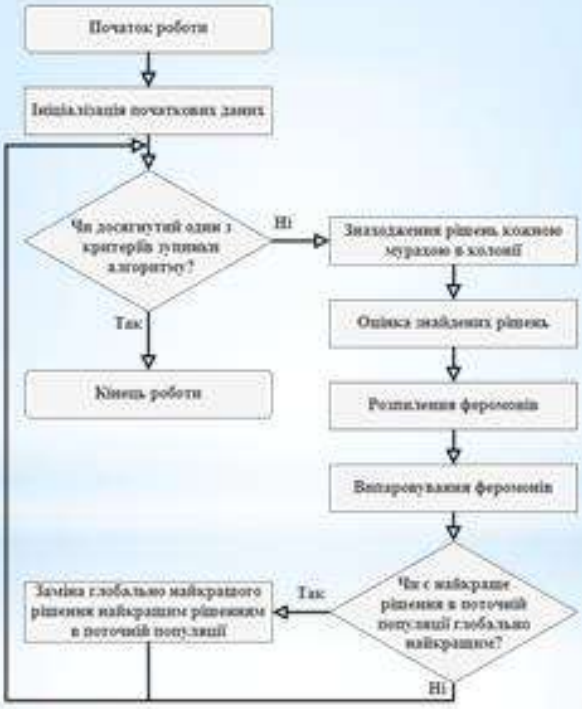
$d_{заг.}$ - загальний обсяг вільного місця у всіх транспортних засобів;

m - кількість маршрутів у рішенні;

k - кількість вершин, для яких обслуговування відбувається за межами часового вікна.

9

Мурашиний алгоритм



10

Мурахи

Реалізований мурашиний алгоритм дозволяє використовувати більше одного типу мурахи одночасно і окремо для кожного типу налаштовувати параметри.

Використовуються наступні типи мурах:

Класична - на кожному етапі роботи обирає вершину з усіх доступних. Вибір відбувається на основі розрахованих ймовірностей переходу в кожну вершину.

Елітарна - на кожному етапі роботи обирає вершину з найбільшою ймовірністю переходу в неї.

Рангова - розпилює феромон в залежності від рангу знайденого рішення.

Лінива - на кожному етапі роботи для вибору наступної вершини розглядається тільки частина вершин.

11

Розрахунок ймовірності переходу

Розрахунок ймовірності переходу з поточної вершини у кожну наступну здійснюється за формулою (2).

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{k=1}^{n} \tau_{ik}^{\alpha} \eta_{ik}^{\beta} x_k} \qquad (2)$$

де P_{ij} - ймовірність переходу з пункту i в пункт j;

τ - кількість феромону на шляху;

η - величина зворотна вартості переміщення;

α, β - константі значення, що є параметрами мурахи;

x - бінарна змінна, що показує, чи доступна вершина для переміщення;

n - загальна кількість вершин.

12

Параметри ЗМТ

Для проведення експериментів були використані згенеровані ЗМТ, які описуються повнозв'язними графами, що складаються з 200, 400, 600, 800 та 1000 вершин.

Кожна згенерована ЗМТ має наступні параметри:

- вантажопідйомність кожного транспортного засобу 3000;
- вартість переміщення між двома вузлами лежить в діапазоні від 1 до 2000, матриця - асиметрична;
- потреби клієнтів у вантажі лежать в діапазоні від 1 до 1000;
- час переміщення між двома вузлами в межах від 1 до 1000;
- кожен клієнт має часове вікно, яке встановлюється в діапазоні від 1 до 3000, протяжність часового вікна в межах від 1 до 2000.

13

Параметри ГА

В роботі ГА налаштований наступним чином:

- для генерації початкової популяції використано жадібний алгоритм та модифікацію на його основі, алгоритм Кларка-Райта, метод гілок та меж;
- для схрещування застосовується оператор PMX. Підмножина вершин кожен раз генерується в межах від 2 до 10;
- вибір батьківських особин відбувається шляхом турнірного відбору з 3 учасників турніру;
- використовуються три типи мутації: інверсна, випадкова перестановка вершин в одному маршруті та обмін підмножин вершин між різними маршрутами. З ймовірністю 0.1, 0.1 та 0.4 відповідно.

Для оцінки рішень застосовується фітнес-функція (1) з наступними коефіцієнтами: $C_{коеф.} = 0.1$, $D_{коеф.} = 1$, $M_{коеф.} = 100$, $K_{коеф.} = 10$.

Результати фіксувалися при досягненні одного з критеріїв зупинки: алгоритм виконав 5000 ітерацій або 150 без покращення поточного найкращого рішення.

14

Параметри мурашиного алгоритму

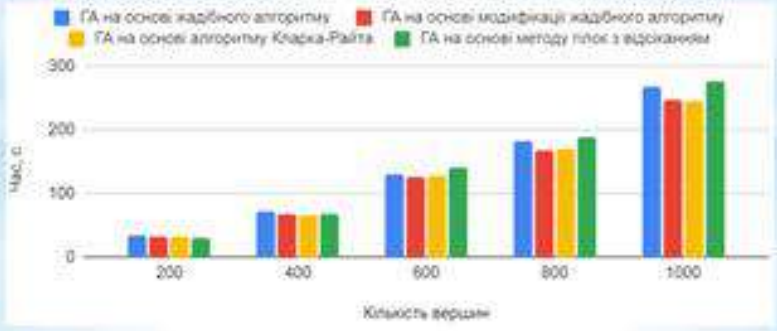
В даній роботі мурашині алгоритми налаштовані наступним чином:

- початкова кількість феромону на кожному переміщенні 1;
- коефіцієнт випаровування 0.2;
- константи α та β мають значення 0.9 та 1.1 відповідно;
- модифікатор близькості 300;
- модифікатор розпилення феромонів 50000;
- кількість мурах 40.

Результати фіксувалися при досягненні одного з критеріїв зупинки: алгоритм виконав 1000 ітерацій або 200 без покращення поточного найкращого рішення.

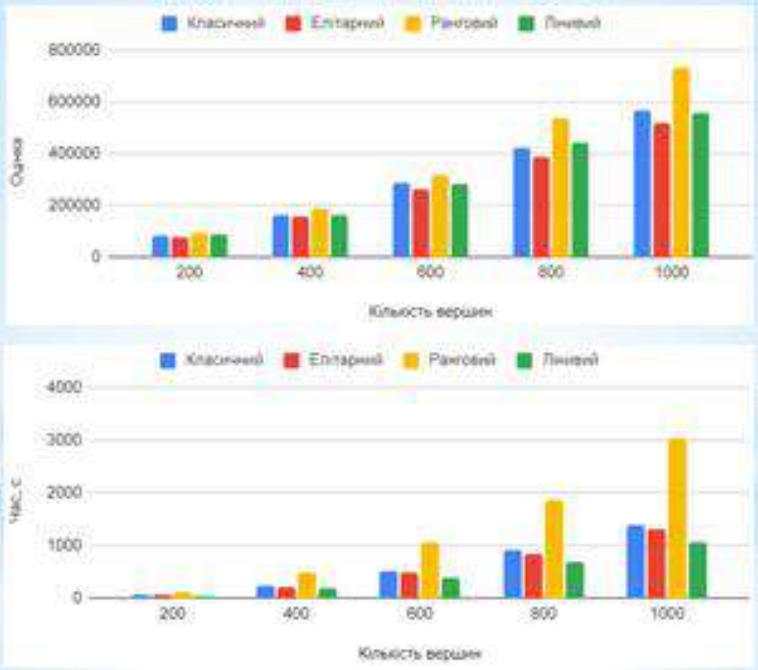
15

Аналіз роботи гібридного ГА



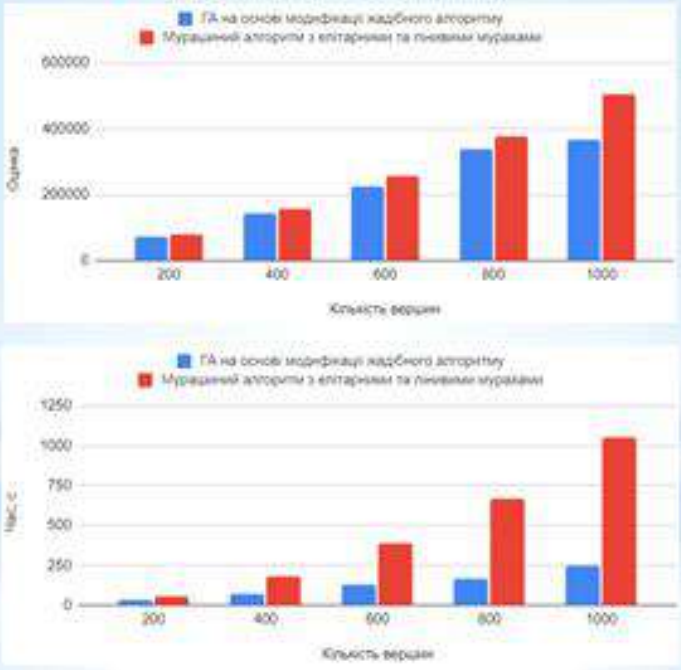
16

Аналіз роботи різновидів мурашиних алгоритмів



17

Порівняння результатів роботи ГА та мурашиного алгоритму

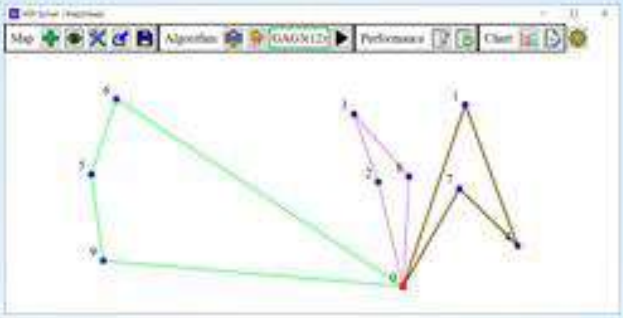


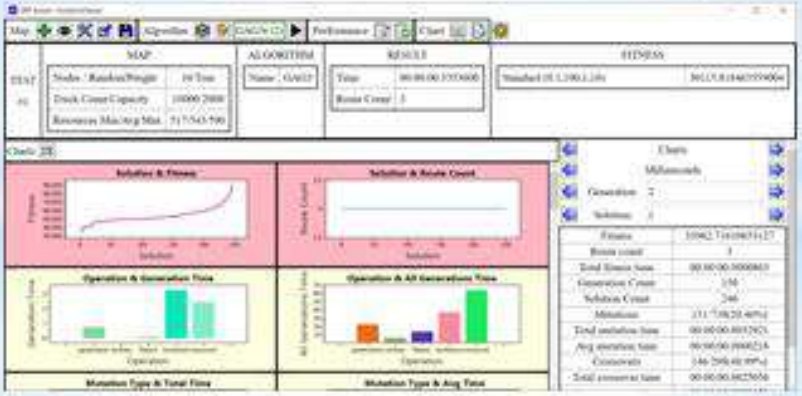
18

Використані технології та засоби розробки

19

Інтерфейс тестової програми





20

Висновки

В роботі досліджено гібридні підходи для рішення ЗМТ, засновані на мурашиних алгоритмах з різними типами мурах, та на поєднанні ГА та класичних алгоритмів. Найбільшу точність та швидкодію, у порівнянні з іншими підходами, забезпечує гібридизація ГА та модифікації жадібного алгоритму. Мурашині алгоритми потребують в декілька разів більше часу для вирішення ЗМТ, ніж ГА, що робить недоцільним їх використання.

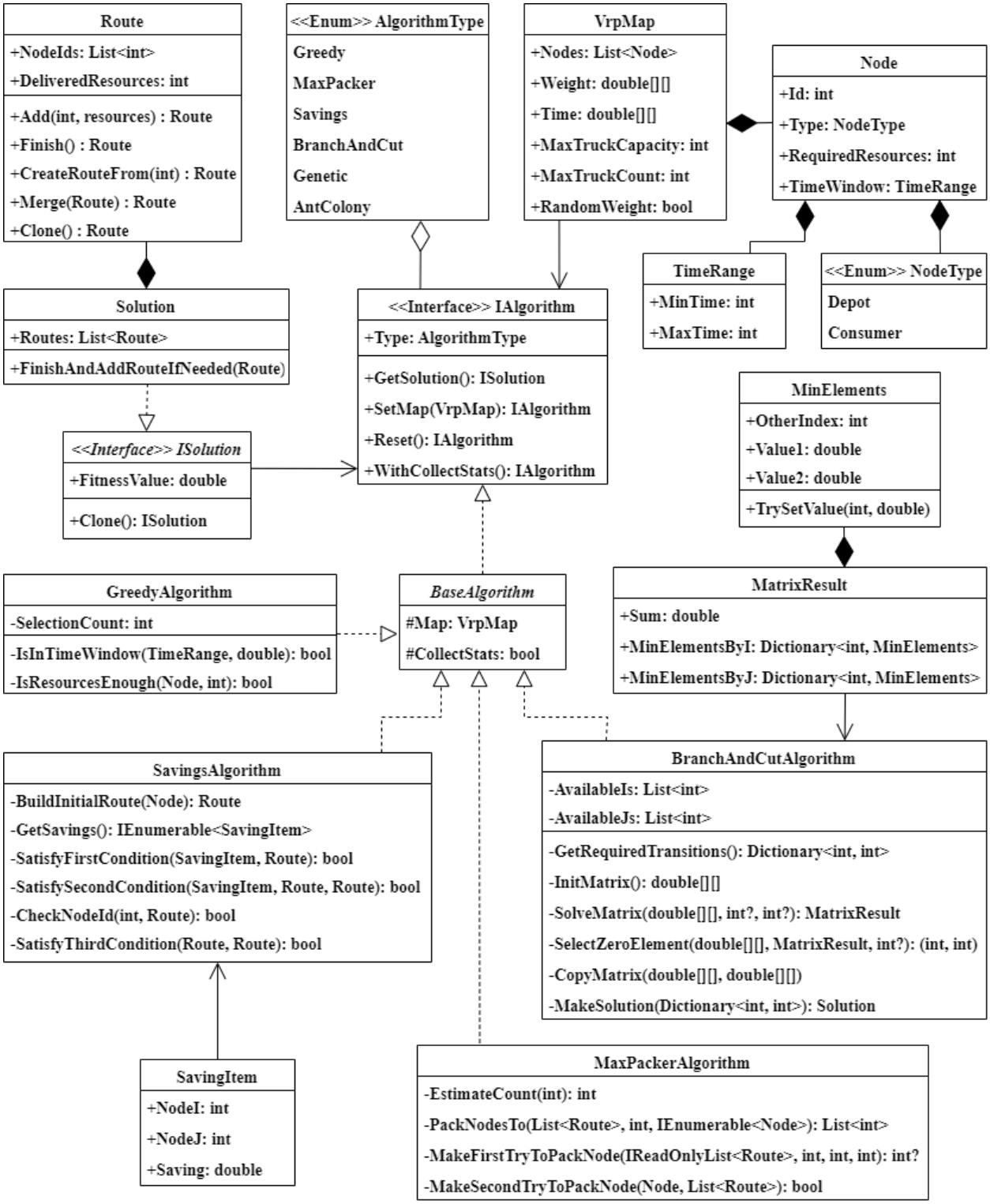
Для тестування та налагодження алгоритмів реалізовано програмний застосунок з графічним інтерфейсом користувача, що дозволяє генерувати ЗМТ з заданими параметрами, проводити запуск наборів алгоритмів для вирішення сформованих наборів задач та переглядати отримані результати.

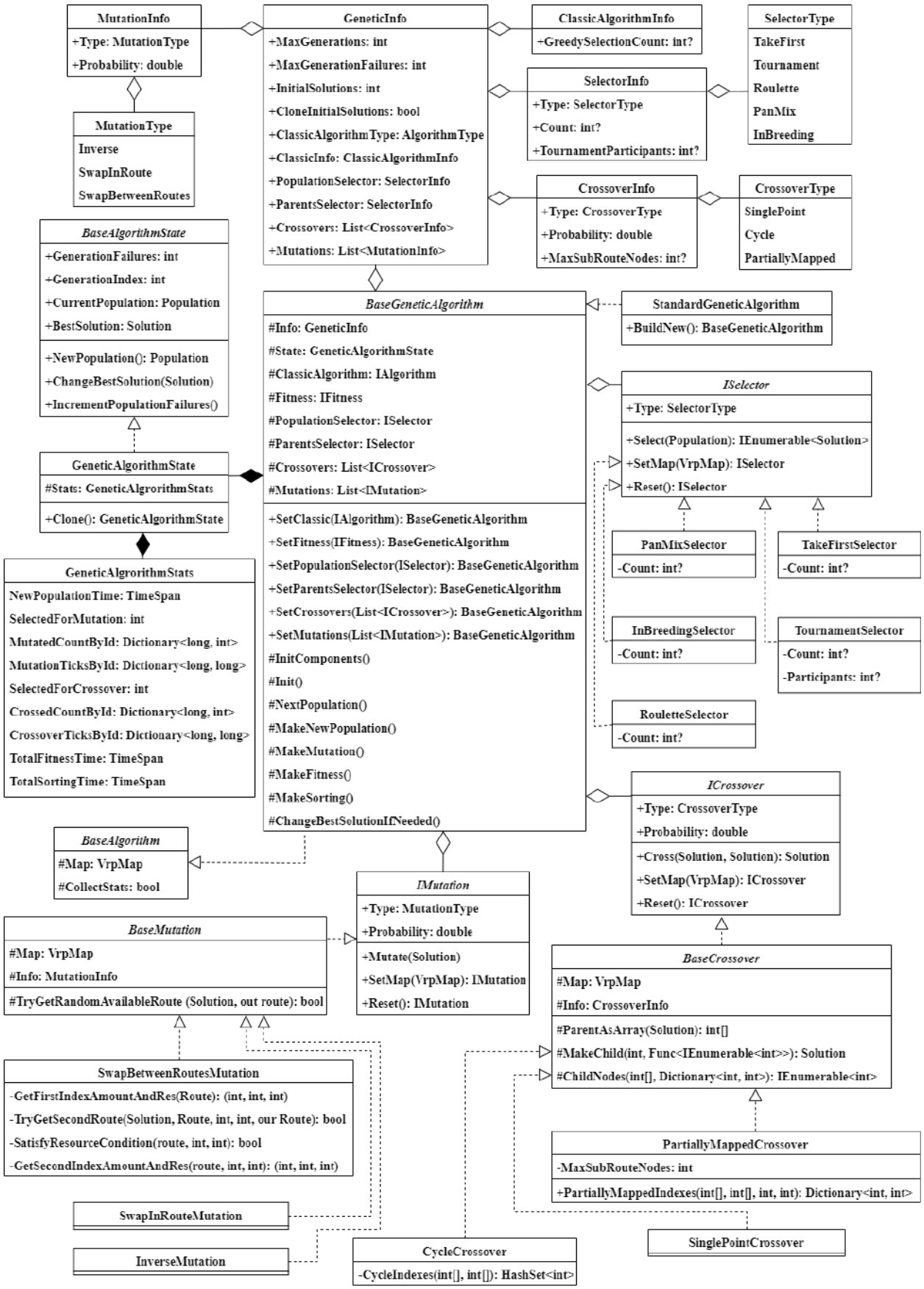
Результати роботи були опубліковані в журналі «Системи управління, навігації та зв'язку» та представлені в рамках дев'ятої міжнародної науково-технічної конференції «Проблеми інформатизації».

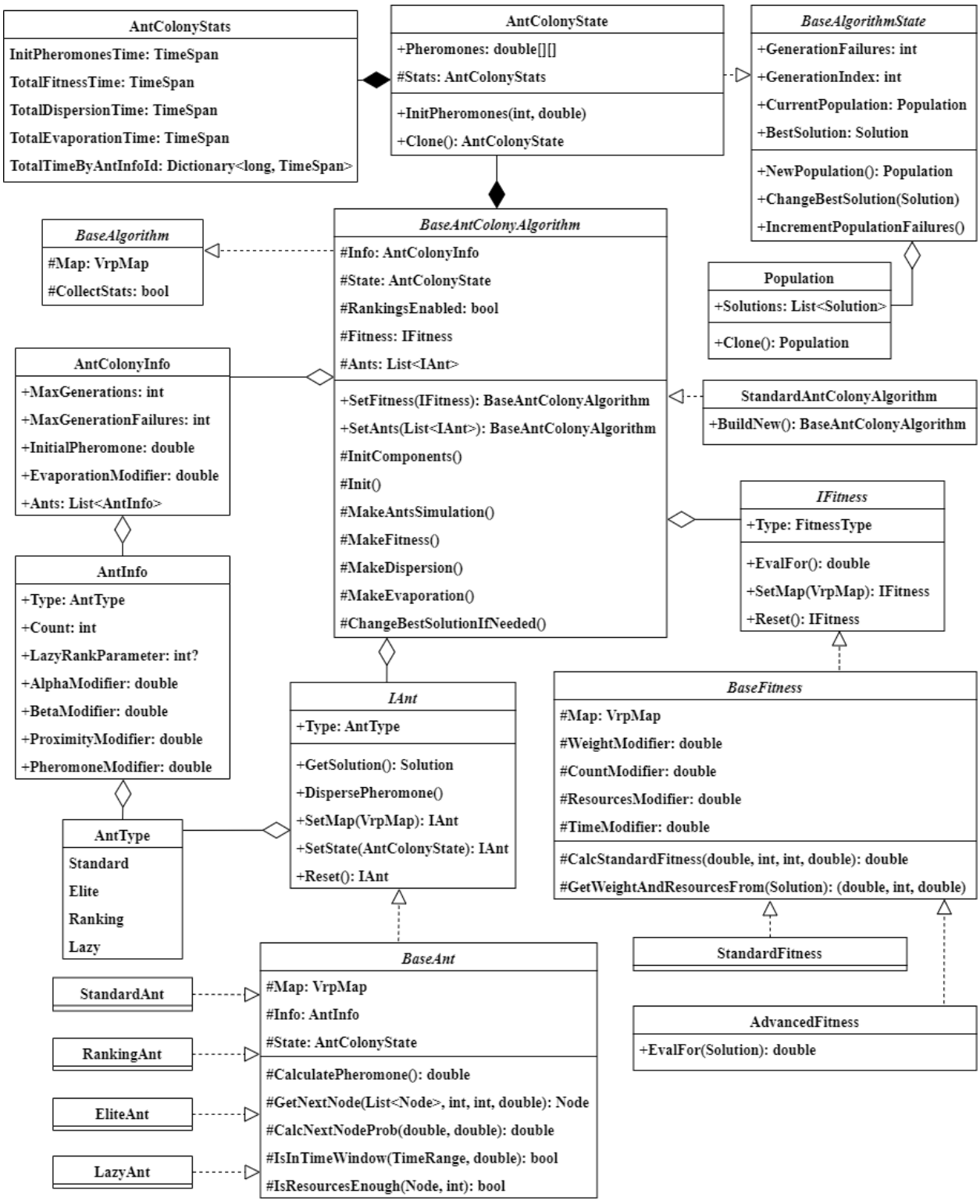
21

ДОДАТОК Б

UML-діаграми класів розроблених алгоритмів




MutationInfo
+Type: MutationType
+Probability: double
MutationType
Inverse
SwapInRoute
SwapBetweenRoutes
GeneticInfo
+MaxGenerations: int
+MaxGenerationFailures: int
+InitialSolutions: int
+CloneInitialSolutions: bool
+ClassicAlgorithmType: AlgorithmType
+ClassicInfo: ClassicAlgorithmInfo
+PopulationSelector: SelectorInfo
+ParentsSelector: SelectorInfo
+Crossovers: List<CrossoverInfo>
+Mutations: List<MutationInfo>
ClassicAlgorithmInfo
+GreedySelectionCount: int?
SelectorInfo
+Type: SelectorType
+Count: int?
+TournamentParticipants: int?
SelectorType
TakeFirst
Tournament
Roulette
PanMix
InBreeding
CrossoverInfo
+Type: CrossoverType
+Probability: double
+MaxSubRouteNodes: int?
CrossoverType
SinglePoint
Cycle
PartiallyMapped
BaseAlgorithmState
+GenerationFailures: int
+GenerationIndex: int
+CurrentPopulation: Population
+BestSolution: Solution
+NewPopulation(): Population
+ChangeBestSolution(Solution)
+IncrementPopulationFailures()
GeneticAlgorithmState
#Stats: GeneticAlgrorithmStats
+Clone(): GeneticAlgorithmState
GeneticAlgrorithmStats
NewPopulationTime: TimeSpan
SelectedForMutation: int
MutatedCountById: Dictionary<long, int>
MutationTicksById: Dictionary<long, long>
SelectedForCrossover: int
CrossedCountById: Dictionary<long, int>
CrossoverTicksById: Dictionary<long, long>
TotalFitnessTime: TimeSpan
TotalSortingTime: TimeSpan
BaseGeneticAlgorithm
#Info: GeneticInfo
#State: GeneticAlgorithmState
#ClassicAlgorithm: IAlgorithm
#Fitness: IFitness
#PopulationSelector: ISelector
#ParentsSelector: ISelector
#Crossovers: List<ICrossover>
#Mutations: List<IMutation>
+SetClassic(IAlgorithm): BaseGeneticAlgorithm
+SetFitness(IFitness): BaseGeneticAlgorithm
+SetPopulationSelector(ISelector): BaseGeneticAlgorithm
+SetParentsSelector(ISelector): BaseGeneticAlgorithm
+SetCrossovers(List<ICrossover>): BaseGeneticAlgorithm
+SetMutations(List<IMutation>): BaseGeneticAlgorithm
#InitComponents()
#Init()
#NextPopulation()
#MakeNewPopulation()
#MakeMutation()
#MakeFitness()
#MakeSorting()
#ChangeBestSolutionIfNeeded()
StandardGeneticAlgorithm
+BuildNew(): BaseGeneticAlgorithm
ISelector
+Type: SelectorType
+Select(Population): IEnumerable<Solution>
+SetMap(VrpMap): ISelector
+Reset(): ISelector
PanMixSelector
-Count: int?
TakeFirstSelector
-Count: int?
InBreedingSelector
-Count: int?
TournamentSelector
-Count: int?
-Participants: int?
RouletteSelector
-Count: int?
ICrossover
+Type: CrossoverType
+Probability: double
+Cross(Solution, Solution): Solution
+SetMap(VrpMap): ICrossover
+Reset(): ICrossover
BaseAlgorithm
#Map: VrpMap
#CollectStats: bool
IMutation
+Type: MutationType
+Probability: double
+Mutate(Solution)
+SetMap(VrpMap): IMutation
+Reset(): IMutation
BaseMutation
#Map: VrpMap
#Info: MutationInfo
#TryGetRandomAvailableRoute (Solution, out route): bool
BaseCrossover
#Map: VrpMap
#Info: CrossoverInfo
#ParentAsArray(Solution): int[]
#MakeChild(int, Func<IEnumerable<int>>): Solution
#ChildNodes(int[], Dictionary<int, int>): IEnumerable<int>
SwapBetweenRoutesMutation
-GetFirstIndexAmountAndRes(Route): (int, int, int)
-TryGetSecondRoute(Solution, Route, int, int, our Route): bool
-SatisfyResourceCondition(route, int, int): bool
-GetSecondIndexAmountAndRes(route, int, int): (int, int, int)
PartiallyMappedCrossover
-MaxSubRouteNodes: int
+PartiallyMappedIndexes(int[], int[], int, int): Dictionary<int, int>
SwapInRouteMutation
InverseMutation
CycleCrossover
-CycleIndexes(int[], int[]): HashSet<int>
SinglePointCrossover


AntColonyStats
InitPheromonesTime: TimeSpan
TotalFitnessTime: TimeSpan
TotalDispersionTime: TimeSpan
TotalEvaporationTime: TimeSpan
TotalTimeByAntInfoId: Dictionary<long, TimeSpan>
AntColonyState
+Pheromones: double[][]
#Stats: AntColonyStats
+InitPheromones(int, double)
+Clone(): AntColonyState
BaseAlgorithmState
+GenerationFailures: int
+GenerationIndex: int
+CurrentPopulation: Population
+BestSolution: Solution
+NewPopulation(): Population
+ChangeBestSolution(Solution)
+IncrementPopulationFailures()
BaseAlgorithm
#Map: VrpMap
#CollectStats: bool
BaseAntColonyAlgorithm
#Info: AntColonyInfo
#State: AntColonyState
#RankingsEnabled: bool
#Fitness: IFitness
#Ants: List<IAnt>
+SetFitness(IFitness): BaseAntColonyAlgorithm
+SetAnts(List<IAnt>): BaseAntColonyAlgorithm
#InitComponents()
#Init()
#MakeAntsSimulation()
#MakeFitness()
#MakeDispersion()
#MakeEvaporation()
#ChangeBestSolutionIfNeeded()
Population
+Solutions: List<Solution>
+Clone(): Population
AntColonyInfo
+MaxGenerations: int
+MaxGenerationFailures: int
+InitialPheromone: double
+EvaporationModifier: double
+Ants: List<AntInfo>
StandardAntColonyAlgorithm
+BuildNew(): BaseAntColonyAlgorithm
IFitness
+Type: FitnessType
+EvalFor(): double
+SetMap(VrpMap): IFitness
+Reset(): IFitness
AntInfo
+Type: AntType
+Count: int
+LazyRankParameter: int?
+AlphaModifier: double
+BetaModifier: double
+ProximityModifier: double
+PheromoneModifier: double
IAnt
+Type: AntType
+GetSolution(): Solution
+DispersePheromone()
+SetMap(VrpMap): IAnt
+SetState(AntColonyState): IAnt
+Reset(): IAnt
BaseFitness
#Map: VrpMap
#WeightModifier: double
#CountModifier: double
#ResourcesModifier: double
#TimeModifier: double
#CalcStandardFitness(double, int, int, double): double
#GetWeightAndResourcesFrom(Solution): (double, int, double)
AntType
Standard
Elite
Ranking
Lazy
StandardFitness
AdvancedFitness
BaseAnt
#Map: VrpMap
#Info: AntInfo
#State: AntColonyState
#CalculatePheromone(): double
#GetNextNode(List<Node>, int, int, double): Node
#CalcNextNodeProb(double, double): double
#IsInTimeWindow(TimeRange, double): bool
#IsResourcesEnough(Node, int): bool
StandardAnt
RankingAnt
EliteAnt
LazyAnt

ДОДАТОК В

Вихідний код програмного забезпечення

Лістинг 1 – Файл IAlgorithm.cs

```
using VRP.Model.Algorithms.Solutions;
namespace VRP.Model.Algorithms.Base
{
public interface IAlgorithm<in TMap, out TSolution>
where TMap : class
where TSolution : ISolution<TSolution>
{
AlgorithmType Type { get; }
TSolution GetSolution();
IAlgorithm<TMap, TSolution> SetMap(TMap map);
IAlgorithm<TMap, TSolution> Reset();
IAlgorithm<TMap, TSolution> WithCollectStats();
}
}
```

Лістинг 2 – Файл BaseAlgorithm.cs

```
using VRP.Model.Algorithms.Solutions;
namespace VRP.Model.Algorithms.Base
{
public abstract class BaseAlgorithm<TMap, TSolution>  :
IAlgorithm<TMap, TSolution>
where TMap : class
where TSolution : ISolution<TSolution>
{
public abstract AlgorithmType Type { get; }
protected TMap Map = null!;
protected bool CollectStats;
public virtual IAlgorithm<TMap, TSolution> SetMap(TMap map)
{
Map = map;
return this;
}
public virtual IAlgorithm<TMap, TSolution> Reset()
{
Map = null!;
CollectStats = false;
return this;
}
public virtual IAlgorithm <TMap, TSolution> WithCollectStats()
```

```
{
CollectStats = true;
return this;
}
public abstract TSolution GetSolution();
}
}
```

Лістинг 3 – Файл GreedyAlgorithm.cs

```
using System.Linq;
using VRP.Common;
using VRP.Model.Algorithms.Base;
using VRP.Model.Algorithms.Solutions;
using VRP.Model.Map;
namespace VRP.Model.Algorithms.Classic
{
public class GreedyAlgorithm : BaseAlgorithm<VrpMap, Solution>
{
public override AlgorithmType Type => AlgorithmType.Greedy;
private readonly int _selectionCount;
public GreedyAlgorithm(int selectionCount = 1) =>
_selectionCount = selectionCount;
public override Solution GetSolution()
{
var availableConsumers = Map.Nodes.Where(x =>
x.IsConsumer).ToList();
var solution = new Solution();
var route = Route.CreateRouteFrom(Map.DepotId);
var routeTime = 0d;
while (availableConsumers.Count > 0)
{
var availableCapacity = Map.MaxTruckCapacity -
route.DeliveredResources;
var prevNodeId = route.NodeIds.Last();
Node[] nextNodes = null;
var nextNodesWithTimeWindows = availableConsumers
.Where(x => IsResourcesEnough(x, availableCapacity))
.Where(x => IsInTimeWindow(x.TimeWindow, routeTime +
Map.TimeMatrix[prevNodeId][x.Id]))
.OrderBy(x => x.TimeWindow.MaxTime)
.ThenBy(x => Map.Weight[prevNodeId][x.Id])
.Take(_selectionCount)
.ToArray();
if (!nextNodesWithTimeWindows.IsNullOrEmpty())
nextNodes = nextNodesWithTimeWindows;
if (nextNodes == null)
{
var nextNodesWithoutTimeWindows = availableConsumers
.Where(x => x.TimeWindow == null)
.Where(x => IsResourcesEnough(x, availableCapacity))
```

```
.OrderBy(x => Map.Weight[prevNodeId][x.Id])
.Take(_selectionCount)
.ToArray();                    if
(!nextNodesWithoutTimeWindows.IsNullOrEmpty())
nextNodes = nextNodesWithoutTimeWindows;
nextNodes ??= availableConsumers
.Where(x => IsResourcesEnough(x, availableCapacity))
.OrderBy(x => Map.Weight[prevNodeId][x.Id])
.Take(_selectionCount)
.ToArray();
if (nextNodes.Length == 0)
{
solution.FinishAndAddRouteIfNeeded(route);
route = Route.CreateRouteFrom(Map.DepotId);
routeTime = 0;
}
else
{
var nextIndex = ConcurrentRandom.Next(0, nextNodes.Length);
var nextNode = nextNodes[nextIndex];
availableConsumers.Remove(nextNode);
route.Add(nextNode.Id, nextNode.RequiredResources ?? 0);
routeTime += Map.TimeMatrix[prevNodeId][nextNode.Id];
}
}
solution.FinishAndAddRouteIfNeeded(route);
return solution;
}
private static bool IsInTimeWindow(TimeRange window, double
routeTime) =>
window != null && window.MinTime < routeTime && window.MaxTime >
routeTime;
private static bool IsResourcesEnough(Node node, int
availableCapacity) =>
node.RequiredResources <= availableCapacity;
}
}
```

Лістинг 4 – Файл MaxPackerAlgorithm.cs

```
using System.Collections.Generic;
using System.Linq;
using VRP.Model.Algorithms.Base;
using VRP.Model.Algorithms.Solutions;
using VRP.Model.Map;
namespace VRP.Model.Algorithms.Classic
{
public class MaxPackerAlgorithm : BaseAlgorithm<VrpMap,
Solution>
{
public override AlgorithmType Type => AlgorithmType.MaxPacker;
```

```
public override Solution GetSolution()
{
var totalRes = Map.NodesWithoutDepot.Sum(x =>
x.RequiredResources!.Value);
var estimatedCount = EstimateCount(totalRes);
var finalRoutes = new List<Route>(estimatedCount);
var notPacked = PackNodesTo(finalRoutes, estimatedCount,
Map.NodesWithoutDepot);
while (notPacked.Count > 0)
{
totalRes = notPacked.Select(x => Map.Nodes[x]).Sum(x =>
x.RequiredResources!.Value);
estimatedCount = EstimateCount(totalRes);
notPacked = PackNodesTo(finalRoutes, estimatedCount,
notPacked.Select(x => Map.Nodes[x]));
}
foreach (var route in finalRoutes)
route.Finish();
return new Solution(finalRoutes);
}
private int EstimateCount(int totalRes)
{
var lastRouteCount = totalRes % Map.MaxTruckCapacity != 0 ? 1 :
0;
return totalRes / Map.MaxTruckCapacity + lastRouteCount;
}
private List<int> PackNodesTo(List<Route> finalRoutes, int
routesCount, IEnumerable<Node> nodes)
{
var routes = new List<Route>(routesCount);
var notPackedOnFirstTry = new List<int>();
var routeIndex = 0;
foreach (var node in nodes.OrderByDescending(x =>
x.RequiredResources))
{
if (routeIndex >= routes.Count)
routes.Add(Route.CreateRouteFrom(Map.DepotId));
var res = node.RequiredResources!.Value;
var addedIndex = MakeFirstTryToPackNode(routes, routeIndex,
node.Id, res);
if (addedIndex == null)
notPackedOnFirstTry.Add(node.Id);
else if (addedIndex == routeIndex && ++routeIndex >=
routesCount)
routeIndex = 0;
}
finalRoutes.AddRange(routes);
var notPackedOnSecondTry = new List<int>();
foreach (var nodeId in notPackedOnFirstTry)
if (!MakeSecondTryToPackNode(Map.Nodes[nodeId], finalRoutes))
notPackedOnSecondTry.Add(nodeId);
return notPackedOnSecondTry;
}
```

```
private int? MakeFirstTryToPackNode(IReadOnlyList<Route> routes,
int routeIndex, int nodeId, int res)
{
for (var i = routeIndex; i < routes.Count; i++)
{
var route = routes[i];
if (route.DeliveredResources + res > Map.MaxTruckCapacity)
continue;
route.Add(nodeId, res);
return i;
}
return null;
}
private bool MakeSecondTryToPackNode(Node node, List<Route>
routes)
{
var neededRes = node.RequiredResources!.Value;
foreach (var route1 in routes)
foreach (var route2 in routes)
{
if (route1 == route2)
continue;
var res1 = Map.MaxTruckCapacity - route1.DeliveredResources;
var res2 = Map.MaxTruckCapacity - route2.DeliveredResources;
if (res1 + res2 < neededRes)
continue;
for (var i = 1; i < route1.NodeIds.Count; i++)
{
var node1 = Map.Nodes[route1.NodeIds[i]];
for (var j = 1; j < route2.NodeIds.Count; j++)
{
var node2 = Map.Nodes[route2.NodeIds[j]];
var newRes1 = (route1.DeliveredResources -
node1.RequiredResources + node2.RequiredResources)!.Value;
var newRes2 = (route2.DeliveredResources -
node2.RequiredResources + node1.RequiredResources)!.Value;
if (newRes1 > Map.MaxTruckCapacity || newRes2 >
Map.MaxTruckCapacity)
continue;
Route selectedRoute;
if (newRes1 + node.RequiredResources!.Value <
Map.MaxTruckCapacity)
selectedRoute = route1;
else if (newRes2 + node.RequiredResources!.Value <
Map.MaxTruckCapacity)
selectedRoute = route2;
else
continue;
route1.NodeIds[i] = node2.Id;
route2.NodeIds[j] = node1.Id;
route1.DeliveredResources = newRes1;
route2.DeliveredResources = newRes2;
selectedRoute.Add(node.Id, node.RequiredResources!.Value);
```

```
return true;
}
}
}
return false;
}
}
}
```

Лістинг 5 – Файл SavingsAlgorithm.cs

```
using System.Collections.Generic;
using System.Linq;
using VRP.Model.Algorithms.Base;
using VRP.Model.Algorithms.Solutions;
using VRP.Model.Map;
namespace VRP.Model.Algorithms.Classic
{
public class SavingItem
{
public readonly int NodeI;
public readonly int NodeJ;
public readonly double Saving;
public SavingItem(int nodeI, int nodeJ, double saving) =>
(NodeI, NodeJ, Saving) = (nodeI, nodeJ, saving);
}
public class SavingsAlgorithm : BaseAlgorithm<VrpMap, Solution>
{
public override AlgorithmType Type => AlgorithmType.Savings;
public override Solution GetSolution()
{
var routeByNodeId = Map.NodesWithoutDepot
.Select(BuildInitialRoute)
.ToDictionary(x => x.NodeIds.First(y => y != Map.DepotId), x =>
x);
foreach (var savingItem in GetSavings().OrderByDescending(x =>
x.Saving))
{
var routeI = routeByNodeId[savingItem.NodeI];
var routeJ = routeByNodeId[savingItem.NodeJ];
if (!(SatisfySecondCondition(savingItem, routeI, routeJ)
&& SatisfyThirdCondition(routeI, routeJ)
&& SatisfyFirstCondition(savingItem, routeI)))
continue;
routeI.Merge(routeJ);
foreach (var nodeId in routeJ.NodeIds)
routeByNodeId[nodeId] = routeI;
}
return new Solution(routeByNodeId.Values.Distinct().ToList());
}
private Route BuildInitialRoute(Node node) =>
```

```
Route
.CreateRouteFrom(Map.DepotId)
.Add(node.Id, node.RequiredResources ?? 0)
.Finish();
private IEnumerable<SavingItem> GetSavings()
{
for (var i = 0; i < Map.NodeCount; i++)
for (var j = 0; j < Map.NodeCount; j++)
{
if (i == j || i == 0 || j == 0)
continue;
var saving = Map.Weight[i][Map.DepotId] +
Map.Weight[Map.DepotId][j] - Map.Weight[i][j];
yield return new SavingItem(i, j, saving);
}
}
private bool SatisfyFirstCondition(SavingItem item, Route
routeI) => !routeI.NodeIds.Contains(item.NodeJ);
private bool SatisfySecondCondition(SavingItem item, Route
routeI, Route routeJ) =>
CheckNodeId(item.NodeI, routeI) && CheckNodeId(item.NodeJ,
routeJ);
private bool CheckNodeId(int nodeId, Route route) =>
nodeId == route.NodeIds[1] || nodeId == route.NodeIds[^2];
private bool SatisfyThirdCondition(Route routeI, Route routeJ)
=> routeI.DeliveredResources + routeJ.DeliveredResources <=
Map!.MaxTruckCapacity;
}
}
```

Лістинг 6 – Файл BranchAndCutAlgorithm.cs

```
using System.Collections.Generic;
using System.Linq;
using VRP.Common;
using VRP.Model.Algorithms.Base;
using VRP.Model.Algorithms.Solutions;
using VRP.Model.Map;
namespace VRP.Model.Algorithms.Classic
{
public class BranchAndCutAlgorithm : BaseAlgorithm<VrpMap,
Solution>
{
private class MatrixResult
{
public double Sum;
public readonly Dictionary<int, MinElements> MinElementsByI;
public readonly Dictionary<int, MinElements> MinElementsByJ;
public MatrixResult(int countI, int countJ)
{
MinElementsByI = new Dictionary<int, MinElements>(countI);
```

```
MinElementsByJ = new Dictionary<int, MinElements>(countJ);
Sum = 0;
}
}
public override AlgorithmType Type =>
AlgorithmType.BranchAndCut;
private List<int> _availableIs = null!;
private List<int> _availableJs = null!;
public override Solution GetSolution()
{
var requiredTransitions = GetRequiredTransitions();
_availableIs = null!;
_availableJs = null!;
return MakeSolution(requiredTransitions);
}
private Dictionary<int, int> GetRequiredTransitions()
{
_availableIs = new List<int>(Map.NodeCount -
Map.DepotIds.Count);
_availableJs = new List<int>(Map.NodeCount -
Map.DepotIds.Count);
foreach (var node in Map.NodesWithoutDepot)
{
_availableIs.Add(node.Id);
_availableJs.Add(node.Id);
}
var leftMatrix = InitMatrix();
var rightMatrix = InitMatrix();
var requiredTransitions = new Dictionary<int,
int>(Map.NodeCount);
var (zeroI, zeroJ) = SelectZeroElement(leftMatrix,
SolveMatrix(leftMatrix));
var isLeft = true;
while (_availableIs.Count > 1)
{
if (isLeft)
CopyMatrix(rightMatrix, leftMatrix);
else
CopyMatrix(leftMatrix, rightMatrix);
leftMatrix[zeroJ][zeroI] = double.PositiveInfinity;
var leftResult = SolveMatrix(leftMatrix, zeroI, zeroJ);
rightMatrix[zeroI][zeroJ] = double.PositiveInfinity;
var rightResult = SolveMatrix(rightMatrix);
isLeft = leftResult.Sum <= rightResult.Sum;
if (isLeft)
requiredTransitions.Add(zeroI, zeroJ);
_availableIs.Remove(zeroI);
_availableJs.Remove(zeroJ);
(zeroI, zeroJ) = SelectZeroElement(leftMatrix, leftResult,
zeroI, zeroJ);
}
else
(zeroI, zeroJ) = SelectZeroElement(rightMatrix, rightResult);
```

```
}
foreach (var i in _availableIs)
foreach (var j in _availableJs)
{
if (requiredTransitions.ContainsKey(i))
continue;
requiredTransitions.Add(i, j);
}
return requiredTransitions;
}
private double[][] InitMatrix()
{
var matrix = new double[Map.NodeCount][];
foreach (var i in _availableIs)
{
matrix[i] = new double[Map.NodeCount];
foreach (var j in _availableJs)
matrix[i][j] = i != j ? Map.Weight[i][j] :
double.PositiveInfinity;
}
return matrix;
}
private MatrixResult SolveMatrix(double[][] matrix, int?
disabledI = null, int? disabledJ = null)
{
var result = new MatrixResult(_availableIs.Count,
_availableJs.Count);
#region Rows
foreach (var i in _availableIs)
{
if (disabledI == i)
continue;
var minElements = new MinElements
{
Value1 = double.PositiveInfinity,
Value2 = double.PositiveInfinity,
};
foreach (var j in _availableJs)
{
if (disabledJ == j)
continue;
minElements.TrySetValue(j, matrix[i][j]);
}
foreach (var j in _availableJs)
{
if (disabledJ == j)
continue;
matrix[i][j] -= minElements.Value1;
}
result.MinElementsByI.Add(i, minElements);
result.Sum += minElements.Value1;
}
#endregion
```

```
#region Columns
foreach (var j in _availableJs)
{
if (disabledJ == j)
continue;
var minElements = new MinElements
{
Value1 = double.PositiveInfinity,
Value2 = double.PositiveInfinity,
};
foreach (var i in _availableIs)
{
if (disabledI == i)
continue;
minElements.TrySetValue(i, matrix[i][j]);
}
foreach (var i in _availableIs)
{
if (disabledI == i)
continue;
matrix[i][j] -= minElements.Value1;
}
result.MinElementsByJ.Add(j, minElements);
result.Sum += minElements.Value1;
}
#endregion
return result;
}
private (int, int) SelectZeroElement(double[][] matrix,
MatrixResult result, int? disabledI = null,
int? disabledJ = null)
{
var maxSum = double.MinValue;
var zeroI = -1;
var zeroJ = -1;
foreach (var i in _availableIs)
{
if (disabledI == i)
continue;
foreach (var j in _availableJs)
{
if (disabledJ == j)
continue;
var value = matrix[i][j];
if (!value.IsZero())
continue;
var iMinElements = result.MinElementsByI[i];
var jMinElements = result.MinElementsByJ[j];
var localSum =(iMinElements.OtherIndex == j ?
iMinElements.Value2 : iMinElements.Value1) +
(jMinElements.OtherIndex == i ? jMinElements.Value2 :
jMinElements.Value1);
if (localSum < maxSum)
```

```
continue;
maxSum = localSum;
zeroI = i;
zeroJ = j;
}
}
return (zeroI, zeroJ);
}
private void CopyMatrix(double[][] matrixForFill, double[][]
matrixForRead)
{
foreach (var i in _availableIs)
foreach (var j in _availableJs)
matrixForFill[i][j] = matrixForRead[i][j];
}
private Solution MakeSolution(Dictionary<int, int>
requiredTransitions)
{
var count = Map.NodesWithoutDepot.Sum(x =>
x.RequiredResources)!.Value / Map.MaxTruckCapacity;
var solution = new Solution(count);
var transition = requiredTransitions.First();
var nextNodeId = transition.Key;
var route = Route.CreateRouteFrom(Map.DepotId);
route.Add(nextNodeId,
Map.Nodes[transition.Key].RequiredResources!.Value);
while (requiredTransitions.Count > 0){
var prevNodeId = nextNodeId;
nextNodeId = requiredTransitions[prevNodeId];
requiredTransitions.Remove(prevNodeId);
var resources = Map.Nodes[nextNodeId].RequiredResources!.Value;
if (nextNodeId == transition.Key)
{
requiredTransitions.Remove(nextNodeId);
solution.FinishAndAddRouteIfNeeded(route);
if (requiredTransitions.Count == 0)
break;
transition = requiredTransitions.First();
nextNodeId = transition.Key;
resources = Map.Nodes[nextNodeId].RequiredResources!.Value;
route = Route.CreateRouteFrom(Map.DepotId);
}
if (route.DeliveredResources + resources > Map.MaxTruckCapacity)
{
solution.FinishAndAddRouteIfNeeded(route);
route = Route.CreateRouteFrom(Map.DepotId);
}
route.Add(nextNodeId, resources);
}
return solution;
}
}
}
```