**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 5 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Боровков Іван*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc52291748)

[2 Завдання 4](#_Toc52291749)

[3 Виконання 10](#_Toc52291750)

[3.1 Покроковий алгоритм 10](#_Toc52291751)

[3.2 Програмна реалізація алгоритму 10](#_Toc52291752)

[3.2.1 Вихідний код 10](#_Toc52291753)

[3.2.2 Приклади роботи 10](#_Toc52291754)

[3.3 Тестування алгоритму 11](#_Toc52291755)

[Висновок 12](#_Toc52291756)

[Критерії оцінювання 13](#_Toc52291757)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи розробки метаеврестичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

# Виконання

## Покроковий алгоритм

1. Генеруємо квітки.

1.1. Генеруємо n випадкових різних варіантів заповнення рюкзаку, де n – кількість квіток.

2. Відправляємо скаутів.

2.1. Випадковим чином обираємо m різних квіток, де m – к-сть скаутів.

2.2. Сортуємо обрані квітки по їх привабливості (по початковому значенню заповненості рюкзака)

3. Відправляємо робітників.

3.1. Доки не скінчаться робітники :

3.1.1. Обираємо найпривабливішу квітку та визначаємо, скільки робітників можна відправити на цю ділянку (k).

3.1.2. Відправляємо k робітників на цю ділянку. Кожен з робітників видаляє найменш привабливу річ з рюкзака, після чого намагається додати у звільнений обсяг інші речі (в першу чергу — найбільш привабливі), доки рюкзак не переповниться.

3.1.3. Змінюємо стару квітку в масиві на щойно оброблену бджілками.

4. Знаходимо нову найкращу квітку в масиві квіток.

5. Повторюємо пункти 2, 3 та 4, поки не досягнемо найкращого результату.

## Програмна реалізація алгоритму

### Вихідний код

namespace BL;  
  
public class Beehive  
{  
 private int ScoutBeesNum = 4;  
 private int WorkerBeesNum = 30;  
 private int FlowersNum = 10;  
 private PossibleEntities \_possibleEntities = new PossibleEntities();  
 private PossibleEntities \_sortedEntities = new PossibleEntities();  
   
 private List<Backpack> \_allFlowers = new List<Backpack>();  
  
 public Beehive()  
 {  
 \_sortedEntities.SortByСost();  
 }  
 public Beehive(int scoutBeesNum, int workerBeesNum, int flowersNum)  
 {  
 ScoutBeesNum = scoutBeesNum;  
 WorkerBeesNum = workerBeesNum;  
 FlowersNum = flowersNum;  
 \_sortedEntities.SortByСost();  
 }  
  
 public void StartBeehive(int iterations)  
 {  
 GenerateFlowers();  
 for (int i = 0; i < iterations; i++)  
 {  
 Backpack[] scoutedFlowers = SendScouts();  
 SendWorkers(scoutedFlowers);  
 }  
 }  
  
 private void GenerateFlowers()  
 {  
 for (int i = 0; i < FlowersNum; i++)  
 {  
 Backpack backpack = new Backpack();  
 backpack.GenerateRandomBackpack();  
 \_allFlowers.Add(backpack);  
 }  
 }  
  
 private Backpack[] SendScouts()  
 {  
 Backpack[] scoutedFlowers = GetFlowersToScout();  
 EvaluateFlowers(ref scoutedFlowers);  
 return scoutedFlowers;  
 }  
  
 private Backpack[] GetFlowersToScout()  
 {  
 var rnd = new Random();  
 var randomNumbers = Enumerable.Range(0, FlowersNum).OrderBy(x => rnd.Next()).Take(ScoutBeesNum).ToList();  
  
 Backpack[] scoutedFlowers = new Backpack[ScoutBeesNum];  
 for (int i = 0; i < ScoutBeesNum; i++)  
 {  
 scoutedFlowers[i] = \_allFlowers[randomNumbers[i]];  
 }  
 randomNumbers.Sort((x, y) => y.CompareTo(x));  
 for (int i = 0; i < ScoutBeesNum; i++)  
 {  
 \_allFlowers.RemoveAt(randomNumbers[i]);  
 }  
   
 return scoutedFlowers;  
 }  
  
 private void EvaluateFlowers(ref Backpack[] scoutedFlowers)  
 {  
 Array.Sort(scoutedFlowers);  
 }  
  
 private void SendWorkers(Backpack[] scoutedFlowers)  
 {  
 int sentWorkers = 0, flowerIndex = 0;  
 while (WorkerBeesNum > sentWorkers && ScoutBeesNum > flowerIndex)  
 {  
 int workersNeeds = CountWorkersSentOnFlower(sentWorkers, scoutedFlowers[flowerIndex]);  
 Backpack newFlower = SendWorkerOnFlower(workersNeeds, scoutedFlowers[flowerIndex]);  
 \_allFlowers.Add(newFlower);  
 sentWorkers += workersNeeds;  
  
 flowerIndex++;  
 }  
  
 for (int i = flowerIndex; i < ScoutBeesNum; i++)  
 {  
 \_allFlowers.Add(scoutedFlowers[flowerIndex]);  
 }  
 }  
  
 private int CountWorkersSentOnFlower(int alreadySentWorkers, Backpack flower)  
 {  
 int sentWorkers;  
 if (alreadySentWorkers + flower.NumOfEntities <= WorkerBeesNum)  
 {  
 sentWorkers = flower.NumOfEntities;  
 }  
 else  
 {  
 sentWorkers = WorkerBeesNum - alreadySentWorkers;  
 }  
  
 return sentWorkers;  
 }  
  
 private Backpack SendWorkerOnFlower(int numOfWorkers, Backpack flower)  
 {  
 Backpack bestFlower = flower;  
 int checkedEntity = 0;  
 for (int i = 0; i < numOfWorkers; i++)  
 {  
 Backpack processedFlower = (Backpack) flower.Clone();  
 while (!processedFlower.IsEntityAdded(checkedEntity))  
 {  
 checkedEntity++;  
 }  
  
 processedFlower.RemoveEntity(checkedEntity);  
 processedFlower = TryToUpgrade(processedFlower);  
  
 if (processedFlower.Size > bestFlower.Size)  
 {  
 bestFlower = processedFlower;  
 }  
 }  
  
 return bestFlower;  
 }  
  
 private Backpack TryToUpgrade(Backpack flower)  
 {  
 for (int i = 15 - 1; i >= 0; i--)  
 {  
 if (flower.IsPossibleToAdd(i))  
 {  
 flower.AddEntity(i);  
 }  
 }  
  
 return flower;  
 }  
  
 public Backpack GetBestSolution()  
 {  
 return \_allFlowers.Max();  
 }  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –

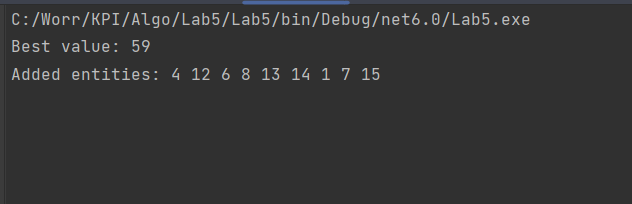
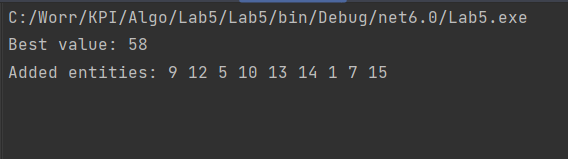


Рисунок 3.2 –



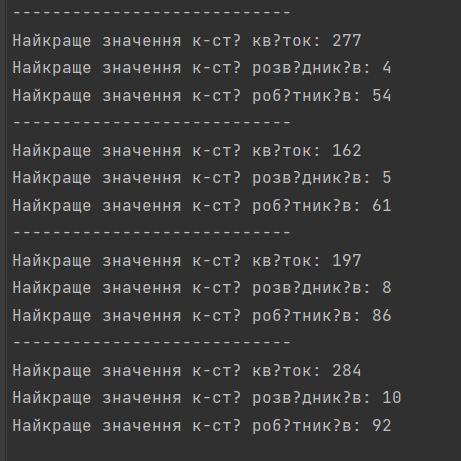
## Тестування алгоритму

Визначення найкращих параметрів для бджолиного алгоритму проведемо наступним чином.

Спочатку визначимо оптимальне значення для параметру к-сті квіток, потім, зафіксувавши цей параметр, обчислимо найкраще значення для к-сті скаутів (пам’ятаємо, що к-сть скаутів не повинна перевищувати к-сть квіток), після чого обчислимо значення к-сті робочих бджіл. Оскільки значення параметрів можуть бути залежними один від одного, то повторимо цей алгоритм декілька разів.

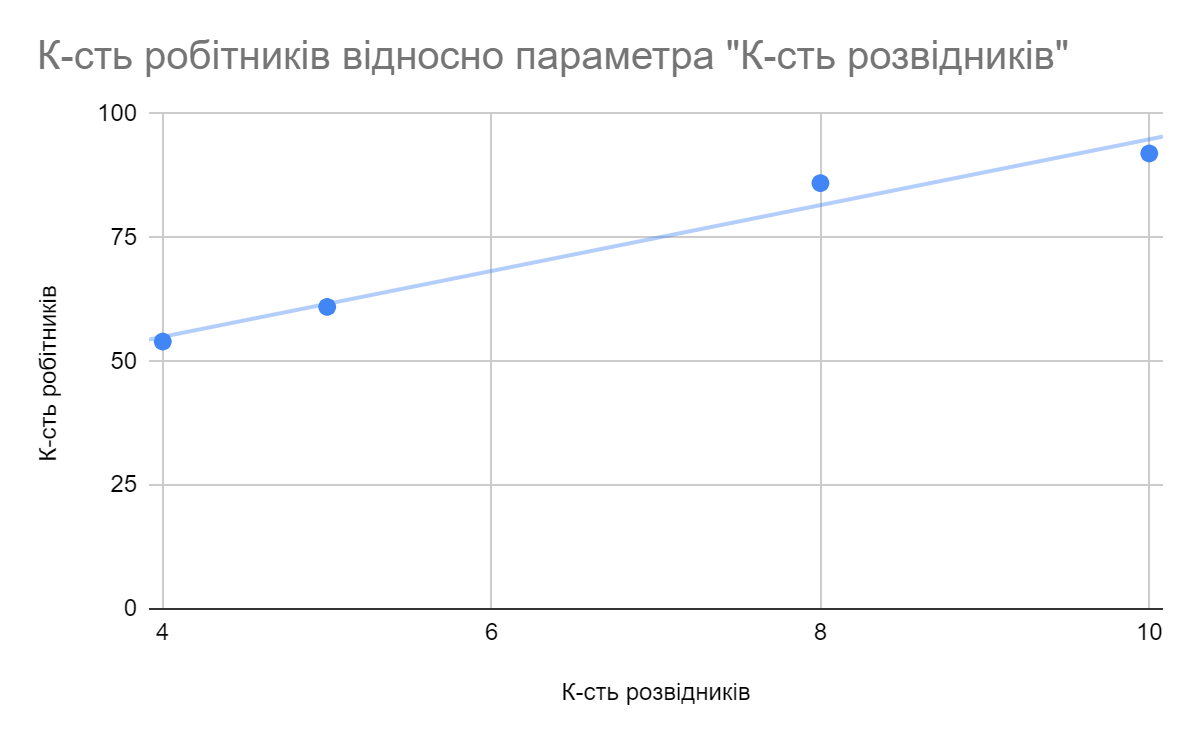
Оскільки бджолиний алгоритм є частково випадковим, то дослідимо значення параметрів на дистанції в 100 пошуків. Найкращим вважатимемо те значення параметру, яке в загальному покаже кращий результат.

Для 15 різних унікальних об’єктів, які можна додати в рюкзак отримали наступні параметри.



Як бачимо, параметр к-сті квіток не залежить від інших параметрів і не є надто важливим для обрахунку результатів. Це значення просто має бути достатньо великим, щоб мати простір для вибору ділянок, щоб не уткнутись в локальне рішення. Більше того, цей параметр мало впливає на часові показники алгоритму (збільшується лише час генерації початкового поля). В той-же час можемо спостерігати певну залежність значень к-сті робітників від к-сті розвідників.

|  |  |
| --- | --- |
| К-сть розвідників | К-сть робітників |
| 4 | 54 |
| 5 | 61 |
| 8 | 86 |
| 10 | 92 |



З тенденції росту видно, що залежність к-сті робітників від к-сті розвідників буде лінійною. Також, варто розуміти, що збільшення к-сті розвідників та робітників прямо впливає на час виконання алгоритму, однак покращує якість рішення.

Висновок

В рамках даної лабораторної роботи в ході даної роботи я дослідив залежність якості та швидкості виконання бджолиного алгоритму від заданих параметрів, встановив залежність між к-стю розвідників та к-стю робітників та вплив цих параметр на якість отримуваних значень.

Критерії оцінювання

При здачі лабораторної роботи до 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* покроковий алгоритм – 15%;
* програмна реалізація алгоритму – 50%;
* тестування алгоритму– 30%;
* висновок – 5%.