**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 Трикош 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 метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

# Завдання

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

* обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
* зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
* після цього параметр фіксується і змінюються інші параметри;
* далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
* зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

|  |  |
| --- | --- |
| **№** | **Задача** |
| 1 | **Задача про рюкзак** (місткість P=500, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб сумарна вага не перевищувала задану, а сумарна цінність була максимальною.  Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика. |
| 2 | **Задача комівояжера** (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.  **Розглядається симетричний, асиметричний та змішаний варіанти.**  В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.  У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.  У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху.  Застосування:   * доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); * доставка води; * моніторинг об'єктів; * поповнення банкоматів готівкою; * збір співробітників для доставки вахтовим методом. |
| 3 | **Розфарбовування графа** (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.  Застосування:   * розкладу для освітніх установ; * розкладу в спорті; * планування зустрічей, зборів, інтерв'ю; * розклади транспорту, в тому числі - авіатранспорту; * розкладу для комунальних служб; |
| 4 | **Задача вершинного покриття** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Вершинне покриття для неорієнтованого графа G = (V, E) - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.  Задача вершинного покриттяполягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).  На вході: Граф G = (V, E).  Результат: множина C ⊆ V - найменше вершинне покриття графа G.    Застосування:   * розміщення пунктів обслуговування; * призначення екіпажів на транспорт; * проектування інтегральних схем і конвеєрних ліній. |
| 5 | **Задача про кліку** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.  Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k, тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).  Застосування:   * біоінформатика; * електротехніка; |
| 6 | **Задача про найкоротший шлях** (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.  Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший. |

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

|  |  |
| --- | --- |
| **№** | **Алгоритми і досліджувані параметри** |
| 1 | **Генетичний алгоритм:**   * оператор схрещування (мінімум 3); * мутація (мінімум 2); * оператор локального покращення (мінімум 2). |
| 2 | **Мурашиний алгоритм**:   * α; * β; * ρ; * Lmin; * кількість мурах М і їх типи (елітні, тощо…); * маршрути з однієї чи різних вершин. |
| 3 | **Бджолиний алгоритм:**   * кількість ділянок; * кількість бджіл (фуражирів і розвідників). |

Таблиця 2.3 – Варіанти задач і алгоритмів

|  |  |
| --- | --- |
| **№** | **Задачі і алгоритми** |
| 1 | Задача про рюкзак + Генетичний алгоритм |
| 2 | Задача про рюкзак + Бджолиний алгоритм |
| 3 | Задача комівояжера (асиметрична мережа) + Генетичний алгоритм |
| 4 | Задача комівояжера (симетрична мережа) + Генетичний алгоритм |
| 5 | Задача комівояжера (змішана мережа) + Генетичний алгоритм |
| 6 | Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм |
| 7 | Задача комівояжера (симетрична мережа) + Мурашиний алгоритм |
| 8 | Задача комівояжера (змішана мережа) + Мурашиний алгоритм |
| 9 | Задача вершинного покриття + Генетичний алгоритм |
| 10 | Задача вершинного покриття + Бджолиний алгоритм |
| 11 | Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм |
| 12 | Задача комівояжера (симетрична мережа) + Бджолиний алгоритм |
| 13 | Задача комівояжера (змішана мережа) + Бджолиний алгоритм |
| 14 | Розфарбовування графа + Генетичний алгоритм |
| 15 | Розфарбовування графа + Бджолиний алгоритм |
| 16 | Задача про кліку (задача розпізнавання) + Генетичний алгоритм |
| 17 | Задача про кліку (задача розпізнавання) + Бджолиний алгоритм |
| 18 | Задача про кліку (обчислювальна задача) + Генетичний алгоритм |
| 19 | Задача про кліку (обчислювальна задача) + Бджолиний алгоритм |
| 20 | Задача про найкоротший шлях + Генетичний алгоритм |
| 21 | Задача про найкоротший шлях + Мурашиний алгоритм |
| 22 | Задача про найкоротший шлях + Бджолиний алгоритм |
| 23 | Задача про рюкзак + Генетичний алгоритм |
| 24 | Задача про рюкзак + Бджолиний алгоритм |
| 25 | Задача комівояжера (асиметрична мережа) + Генетичний алгоритм |
| 26 | Задача комівояжера (симетрична мережа) + Генетичний алгоритм |
| 27 | Задача комівояжера (змішана мережа) + Генетичний алгоритм |
| 28 | Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм |
| 29 | Задача комівояжера (симетрична мережа) + Мурашиний алгоритм |
| 30 | Задача комівояжера (змішана мережа) + Мурашиний алгоритм |

# Виконання

## Покроковий генетичний алгоритм

Вхідні дані:

backpack\_capacity = 500 – місткість рюкзака

stuffs – 100 предметів з випадковою вагою та ціною

population – 100 осіб, по одному позитивному гену в кожному, що відповідає відповідному предмету

1. Для i від 0 до N (самостійно обране число) виконати:
   1. Визначаємо батьків (найкращий + випадковий):
      1. parent1 = особа в популяції, яка має найвище значення цільової функції (прототип рюкзака, який має найвищу ціну)
      2. parent2 = випадкова (але не найкраща) особа в популяції
   2. Обираємо оператор схрещування (crossing\_1, crossing\_2, crossing\_3) і виконуємо схрещування:
      1. Якщо обрано crossing\_1 (одноточкове схрещування), то виконуємо:

1.2.1.1) Обираємо випадкове число l з проміжку [0; n), де n – кількість предметів (кількість генів у хромосомі)

1.2.1.2) result = об’єднання генів parent1 з 0 по l (не включно) та генів parent2 з l по n (не включно)

1.2.1.3) Якщо вага особи result > backpack\_capacity, то схрещування невдале і result = None

* + 1. Якщо обрано crossing\_2 (двоточкове схрещування), то виконуємо:

1.2.2.1) Обираємо випадкове число l1 з проміжку [0; n), де n – кількість предметів (кількість генів у хромосомі)

1.2.2.2) Обираємо випадкове число l2 з проміжку [0; n), де n – кількість предметів (кількість генів у хромосомі)

1.2.2.3) Якщо l1 < l2, то:

1.2.2.3.1) result = об’єднання генів parent1 з 0 по l1 (не включно), генів parent2 з l1 по l2 (не включно) і генів parent1 з l2 по n (не включно)

1.2.2.4) інакше:

1.2.2.4.1) result = об’єднання генів parent1 з 0 по l2 (не включно), генів parent2 з l2 по l1 (не включно) і генів parent1 з l1 по n (не включно)

1.2.2.5) Якщо вага особи result > backpack\_capacity, то схрещування невдале і result = None

* + 1. Якщо обрано crossing\_3 (рівномірне схрещування), то виконуємо:

1.2.3.1) result = [] – пустий список

1.2.3.2) Для i від 0 до parent1.length (не включно):

1.2.3.2.1) Обираємо випадкове число choice з проміжку [0; 1].

1.2.3.2.2) Якщо choice == 1, то:

1.2.3.2.2.1) Додаємо ген parent1[i] до result

1.2.3.2.3) інакше:

1.2.3.2.2.1) Додаємо ген parent2[i] до result

1.2.3.3) Якщо вага особи result > backpack\_capacity, то схрещування невдале і result = None

* 1. Якщо result != None (схрещування вдале), то:

1.3.1) Обираємо оператор мутації (mutation\_1, mutation \_2) і виконуємо мутацію:

1.3.1.1) Якщо обрано mutation\_1 (ймовірнісна (10%) мутація), то:

1.3.1.1.1) res1 = result

1.3.1.1.2) Обираємо випадкове число numb з проміжку [1; 10]

1.3.1.1.3) Якщо numb == 1, то:

1.3.1.1.3.1) Обираємо випадкове число index з проміжку [0, result.length)

1.3.1.1.3.2) res1[index] = 1 – res1[index]

1.3.1.1.4) Якщо вага особи res1 > backpack\_capacity, то мутація невдала і res1 = None, інакше result = res1

1.3.1.2) Якщо обрано mutation\_2 (Мутація хромосоми з переміщенням генів), то:

1.3.1.2.1) res1 = result

1.3.1.2.2) Якщо у хромосомі res1 є хоча б по одному позитивному та негативному гену, то:

1.3.1.2.2.1) Обираємо з хромосоми res1 один позитивний ген та один негативний ген, і міняємо їх місцями.

1.3.1.2.2.2) Якщо вага особи res1 > backpack\_capacity, то мутація невдала і res1 = None, інакше result = res1

1.3.2) Обираємо оператор локального покращення (improvement\_1, improvement\_2) і виконуємо локальне покращення:

1.3.2.1) Якщо обрано improvement \_1 (додавання у рюкзак предмету з мінімальною вагою), то:

1.3.2.1.1) index = індекс предмету у stuffs, який не включений у рюкзак result і має серед невключених у рюкзак result предметів найменшу вагу

1.3.2.1.2) res2 = result

1.3.2.1.3) res2[index] = 1

1.3.2.1.4) Якщо вага особи res2 > backpack\_capacity, то покращення невдале і res2 = None, інакше result = res2

1.3.2.2) Якщо обрано improvement \_2 (додавання у рюкзак предмета з допустимою вагою і найбільшою ціною), то:

1.3.2.2.1) res2 = result

1.3.2.2.2) weight = вага рюкзака res2

1.3.2.2.3) stuffs\_indexes = [] – індекси предметів, які вміщаються у рюкзак res2

1.3.2.2.4) Для i від 0 до res2.lenght (не включно):

1.3.2.2.4.1) Якщо res2[i] == 0 (предмета немає в рюкзаку res2), то:

1.3.2.2.4.1.1) Якщо backpack\_capacity >= weight + stuffs[i].weight (вага предмету), то додаємо індекс i до stuffs\_indexes

1.3.2.2.5) Якщо stuffs\_indexes.lenght > 0, то:

1.3.2.2.5.1) max\_stuff\_index = індекс предметів з stuffs\_indexes, що має найбільшу ціну

1.3.2.2.5.2) res2[max\_stuff\_index] = 1

1.3.2.2.6) result = res2

1.3.3) Додаємо рюкзак result до population

1.3.4) Видаляємо із population рюкзак, що має найнижче значення цільової функції (має найнижчу ціну)

1.3.5) Записуємо у backpack\_result (результат роботи алгоритму) рюкзак, що має найвище значення цільової функції (має найвищу ціну)

## Програмна реалізація алгоритму

### Вихідний код

import random

class Stuff:

"""Предмет"""

def \_\_init\_\_(self):

"""Конструктор"""

self.price = random.randint(2, 30)

self.weight = random.randint(1, 20)

return

def get\_stuff(self):

"""Отримання предмету як рядка"""

return "price = " + str(self.price) + ", weight = " + str(self.weight) + "; "

class Backpack:

"""Рюкзак"""

def \_\_init\_\_(self):

"""Конструктор"""

self.capacity = 500

self.number = 100

self.result = []

return

class Algorithm:

"""Генетичний алгоритм"""

def \_\_init\_\_(self):

"""Конструктор"""

self.backpack = Backpack()

self.stuffs = [Stuff() for i in range(self.backpack.number)]

self.population = []

self.set\_start\_population()

self.backpack.result = self.best\_chromosome()

self.numb = 0

return

def set\_start\_population(self):

"""Встановлення початкової популяції"""

for i in range(self.backpack.number):

self.population += [[0 for j in range(self.backpack.number)]]

self.population[i][i] = 1

return

def chromosome\_price(self, chromosome):

"""Ціна хромосоми (особи)"""

price = 0

for i in range(len(chromosome)):

price += chromosome[i] \* self.stuffs[i].price

return price

def chromosome\_weight(self, chromosome):

"""Вага хромосоми (особи)"""

weight = 0

for i in range(len(chromosome)):

weight += chromosome[i] \* self.stuffs[i].weight

return weight

def best\_chromosome(self):

"""Найкраща хромосома (особа) в популяції"""

best = self.population[0]

index = 0

for i in range(1, len(self.population)):

if self.chromosome\_price(best) < self.chromosome\_price(self.population[i]):

best = self.population[i]

index = i

return index

def worst\_chromosome(self):

"""Найгірша хромосома (особа) в популяції"""

worst = self.population[0]

index = 0

for i in range(1, len(self.population)):

if self.chromosome\_price(worst) > self.chromosome\_price(self.population[i]):

worst = self.population[i]

index = i

return index

def choose\_parents(self):

"""Вибір батьків"""

index = self.best\_chromosome()

temp = self.population.copy()

parent1 = temp.pop(index)

parent2 = random.choice(temp)

return parent1, parent2

def crossing\_1(self, parent1, parent2):

"""Одноточокове схрещування хромосом"""

l = random.randint(0, len(parent1) - 1)

result = parent1[:l] + parent2[l:]

if self.chromosome\_weight(result) > self.backpack.capacity:

result = None

return result

def crossing\_2(self, parent1, parent2):

"""Двоточкове схрещування хромосом"""

l1 = random.randint(0, len(parent1) - 1)

l2 = random.randint(0, len(parent1) - 1)

result = []

if l1 < l2:

result = parent1[:l1] + parent2[l1:l2] + parent1[l2:]

else:

result = parent1[:l2] + parent2[l2:l1] + parent1[l1:]

if self.chromosome\_weight(result) > self.backpack.capacity:

result = None

return result

def crossing\_3(self, parent1, parent2):

"""Рівномірне схрещування хромосом"""

result = []

for i in range(len(parent1)):

choice = random.randint(0, 1)

if choice:

result += [parent1[i]]

else:

result += [parent2[i]]

if self.chromosome\_weight(result) > self.backpack.capacity:

result = None

return result

def mutation\_1(self, chromosome):

"""Ймовірнісна мутація хромосоми"""

result = chromosome.copy()

numb = random.randint(1, 10)

if numb == 1:

index = random.choice(range(len(result)))

result[index] = 1 - result[index]

self.numb += 1

if self.chromosome\_weight(result) > self.backpack.capacity:

result = chromosome.copy()

return result

def mutation\_2(self, chromosome):

"""Мутація хромосоми з переміщенням генів"""

result = chromosome.copy()

zeroes = []

ones = []

for i in range(len(result)):

if result[i]:

ones += [i]

else:

zeroes += [i]

if len(zeroes) > 0 and len(ones) > 0:

gene1 = random.choice(zeroes)

gene2 = random.choice(ones)

result[gene1], result[gene2] = result[gene2], result[gene1]

if self.chromosome\_weight(result) > self.backpack.capacity:

result = chromosome.copy()

return result

def get\_min\_stuff(self, chromosome):

"""Отримання предмету з мінімальною вагою"""

index = 0

while chromosome[index]:

index += 1

min\_stuff = self.stuffs[index]

for i in range(index + 1, len(self.stuffs)):

if not chromosome[i]:

if min\_stuff.weight > self.stuffs[i].weight:

min\_stuff = self.stuffs[i]

index = i

return index

def improvement\_1(self, chromosome):

"""Локальне покращення (додавання у рюкзак предмету з мінімальною вагою)"""

index = self.get\_min\_stuff(chromosome)

result = chromosome.copy()

result[index] = 1

if self.chromosome\_weight(result) > self.backpack.capacity:

result = chromosome.copy()

return result

def get\_max\_stuff\_price(self, stuffs\_indexes):

"""Отримати предмет з максимальною ціною"""

max\_stuff\_index = 0

if len(stuffs\_indexes):

max\_stuff\_index = stuffs\_indexes[0]

max\_stuff\_price = self.stuffs[max\_stuff\_index].price

for i in stuffs\_indexes:

if self.stuffs[i].price > max\_stuff\_price:

max\_stuff\_price = self.stuffs[i].price

max\_stuff\_index = i

elif self.stuffs[i].price == max\_stuff\_price and self.stuffs[i].weight < self.stuffs[

max\_stuff\_index].weight:

max\_stuff\_price = self.stuffs[i].price

max\_stuff\_index = i

return max\_stuff\_index

def improvement\_2(self, chromosome):

"""Локальне покращення (додавання у рюкзак предмета з допустимою вагою і найбільшою ціною)"""

result = chromosome.copy()

weight = self.chromosome\_weight(result)

stuffs\_indexes = []

for i in range(len(result)):

if not result[i]:

if self.backpack.capacity >= weight + self.stuffs[i].weight:

stuffs\_indexes += [i]

if len(stuffs\_indexes):

max\_stuff\_index = self.get\_max\_stuff\_price(stuffs\_indexes)

result[max\_stuff\_index] = 1

return result

def start(self):

"""Запуск алгоритму"""

for i in range(500):

parent1, parent2 = self.choose\_parents()

temp = self.crossing\_3(parent1, parent2)

if temp:

temp = self.mutation\_1(temp)

temp = self.improvement\_2(temp)

self.population += [temp]

self.population.pop(self.worst\_chromosome())

self.backpack.result = self.population[self.best\_chromosome()]

return self.backpack.result

def main():

algorithm = Algorithm()

with open("stuffs.txt", "r") as file:

lines = file.readlines()

for i in range(len(lines)):

lines[i] = lines[i][:-1]

if len(lines[i]) > 0:

pos = 0

for j in range(len(lines[i])):

if lines[i][j] == " ":

pos = j

algorithm.stuffs[i].price, algorithm.stuffs[i].weight = int(lines[i][:pos]), int(lines[i][pos + 1:])

# with open("stuffs.txt", "w") as file:

# for i in algorithm.stuffs:

# line = str(i.price) + " " + str(i.weight) + "\n"

# file.write(line)

result = algorithm.start()

print("\n\nStuffs:")

for i in range(len(algorithm.stuffs)):

print(algorithm.stuffs[i].get\_stuff(), end="")

if i % 5 == 4:

print()

print("\nResult:")

numb = 0

for i in range(len(result)):

if result[i]:

print(algorithm.stuffs[i].get\_stuff(), end="")

if numb % 5 == 4:

print()

numb += 1

print("\nThe weight of stuffs = ", algorithm.chromosome\_weight(result), "\nThe price of stuffs = ", algorithm.chromosome\_price(result))

numb = 0

for j in result:

if j:

numb += 1

print("The number of stuffs:", numb)

return

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Приклади роботи

На рисунках 3.1-3.12 показані приклади роботи програми за 500 ітерацій.

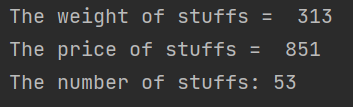


Рисунок 3.1 – Генетичний алгоритм з такими параметрами: схрещування – одноточкове, мутація – ймовірнісна, локальне покращення – додавання у рюкзак предмету з мінімальною вагою

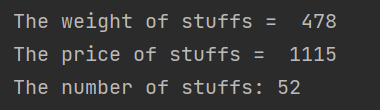


Рисунок 3.2 – Генетичний алгоритм з такими параметрами: схрещування – одноточкове, мутація – ймовірнісна, локальне покращення – додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

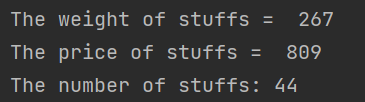


Рисунок 3.3 – Генетичний алгоритм з такими параметрами: схрещування – одноточкове, мутація – з переміщенням генів, локальне покращення – додавання у рюкзак предмету з мінімальною вагою

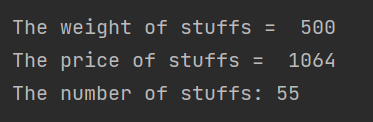


Рисунок 3.4 – Генетичний алгоритм з такими параметрами: схрещування – одноточкове, мутація – з переміщенням генів, локальне покращення – додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

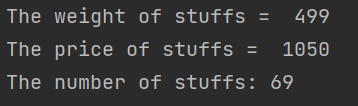


Рисунок 3.5 – Генетичний алгоритм з такими параметрами: схрещування – двоточкове, мутація – ймовірнісна, локальне покращення – додавання у рюкзак предмету з мінімальною вагою

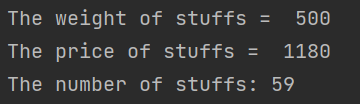


Рисунок 3.6 – Генетичний алгоритм з такими параметрами: схрещування – двоточкове, мутація – ймовірнісна, локальне покращення – додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

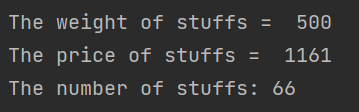


Рисунок 3.7 – Генетичний алгоритм з такими параметрами: схрещування – двоточкове, мутація – з переміщенням генів, локальне покращення – додавання у рюкзак предмету з мінімальною вагою

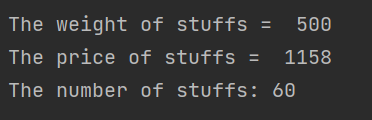


Рисунок 3.8 – Генетичний алгоритм з такими параметрами: схрещування – двоточкове, мутація – з переміщенням генів, локальне покращення – додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

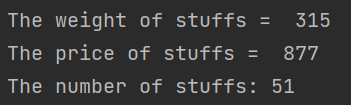


Рисунок 3.9 – Генетичний алгоритм з такими параметрами: схрещування – рівномірне, мутація – ймовірнісна, локальне покращення – додавання у рюкзак предмету з мінімальною вагою

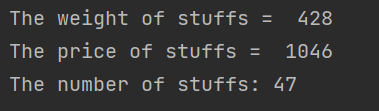


Рисунок 3.10 – Генетичний алгоритм з такими параметрами: схрещування – рівномірне, мутація – ймовірнісна, локальне покращення – додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

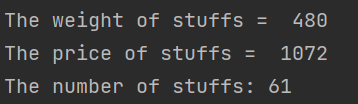


Рисунок 3.11 – Генетичний алгоритм з такими параметрами: схрещування – рівномірне, мутація – з переміщенням генів, локальне покращення – додавання у рюкзак предмету з мінімальною вагою

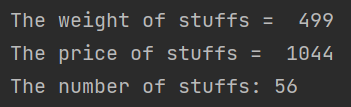


Рисунок 3.12 – Генетичний алгоритм з такими параметрами: схрещування – рівномірне, мутація – з переміщенням генів, локальне покращення – додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

## Тестування алгоритму

Критерій зупинки – 500 ітерацій.

Зафіксуємо параметри «оператор схрещування» на «одноточкове» і «мутація» на «ймовірнісна». Змінюємо тільки «оператор локального покращення». На рисунках 3.13 і 3.14 зображено результати цих випробувань:

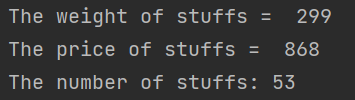


Рисунок 3.13 – Оператор локального покращення: додавання у рюкзак предмету з мінімальною вагою

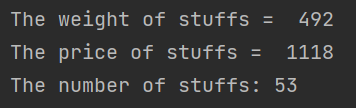


Рисунок 3.14 – Оператор локального покращення: додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

Як бачимо, «додавання у рюкзак предмета з допустимою вагою і найбільшою ціною» є ефективнішим за «додавання у рюкзак предмету з мінімальною вагою», тому далі фіксуємо його і змінюємо інші параметри.

Зафіксуємо параметри «оператор схрещування» на «одноточкове» і «оператор локального покращення» на «додавання у рюкзак предмета з допустимою вагою і найбільшою ціною». Змінюємо тільки «мутація». На рисунках 3.15 і 3.16 зображено результати цих випробувань:

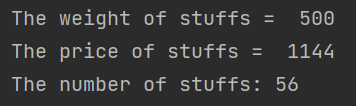


Рисунок 3.15 – Мутація: ймовірнісна

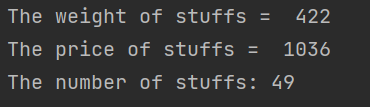


Рисунок 3.16 – Мутація: з переміщенням генів

Як бачимо, ймовірнісна мутація є ефективнішою за мутацію з переміщенням генів, тому далі фіксуємо її і змінюємо інші параметри.

Зафіксуємо параметри «мутація» на «ймовірнісна» і «оператор локального покращення» на «додавання у рюкзак предмета з допустимою вагою і найбільшою ціною». Змінюємо тільки «оператор схрещування». На рисунках 3.17, 3.18, 3.19 зображено результати цих випробувань:

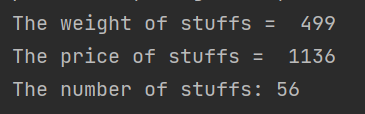


Рисунок 3.17 – Оператор схрещування: одноточкове

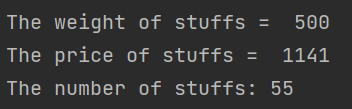


Рисунок 3.18 – Оператор схрещування: двоточкове

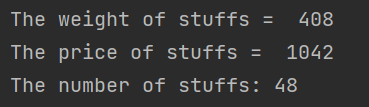


Рисунок 3.19 – Оператор схрещування: рівномірне

Як бачимо, двоточкове схрещування є ефективнішим за одноточкове і рівномірне, тому обираємо його.

Отже, найоптимальніші параметри такі для задачі про рюкзак такі:

* Схрещування – двоточкове (обираємо дві випадкові точки по довжині хромосоми)
* Мутація – ймовірнісна (змінюємо лише випадковий 1 ген)
* Локальне покращення - додавання у рюкзак предмета з допустимою вагою і найбільшою ціною

Висновок

В рамках даної лабораторної роботи я вивчив основні підходи розробки метаевристичних алгоритмів для типових прикладних задач, опрацював методологію підбору прийнятних параметрів алгоритму.

Я дослідив генетичний алгоритм на задачі про рюкзак. Для цього я використав такі параметри: схрещування (одноточкове (обираємо одну випадкову точку по довжині хромосоми), двоточкове (обираємо дві випадкові точки по довжині хромосоми), рівномірне (з ймовірністю 50% записуємо у нащадок ген одного з батьків), мутація (ймовірнісна (змінюємо лише випадковий 1 ген), з переміщенням генів (обираємо два випадкові гени зі значеннями 0 та 1 і міняємо їх місцями) та локальне покращення(додавання у рюкзак предмету з мінімальною вагою, додавання у рюкзак предмета з допустимою вагою і найбільшою ціною) і визначив, що найоптимальніші параметри для задачі про рюкзак такі: схрещування – двоточкове, мутація – ймовірнісна, локальне покращення - додавання у рюкзак предмета з допустимою вагою і найбільшою ціною.

Для алгоритму я використав такий вибір батьків – найкращий та випадковий.

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%.