**Министерство образования и науки Украины**

**Национальный технический университет Украины "Киевский политехнический институт имени Игоря Сикорского"**

**Факультет информатики и вычислительной техники**

**Кафедра автоматизированных систем обработки**

**информации и управления**

**ОТЧЕТ**

по лабораторной работе № 5 по дисциплине

«Проектирование и анализ вычислительных алгоритмов»

„ **Проектирование и анализ алгоритмов поиска** ”

**Выполнил**

(шифр, фамилия, имя, отчество)

*ІП-61, Кушка Михаил*

**Проверил**

(фамилия, имя, отчество )

*Головченко М.Н.*

Киев 2018

Содержание

[1 Цель лабораторной работы 3](#_Toc516431825)

[2 Задание 4](#_Toc516431826)

[3 Выполнение 5](#_Toc516431827)

[3.1 Программная реализация алгоритма 5](#_Toc516431828)

[3.1.1 Исходный код 5](#_Toc516431829)

[3.1.2 Примеры работы 5](#_Toc516431830)

[3.2 Испытания алгоритма 7](#_Toc516431831)

[3.2.1 Значения целевой функции с ростом итераций 7](#_Toc516431832)

[3.2.2 Графики зависимости решения от числа итераций 7](#_Toc516431833)

[Выводы 9](#_Toc516431834)

# Цель лабораторной работы

Цель работы – изучить основные подходы к формализации метаэвристических алгоритмов и решению типовых задач с их помощью.

# Задание

Согласно варианту, разработать алгоритм решения задачи и выполнить его программную реализацию на любом языке программирования.

Задача, алгоритм и его параметры заданы в таблице 2.1.

Зафиксировать качество полученного решения (значение целевой функции) после каждых 20 итераций до 1000 и построить график зависимости качества решения от числа итераций.

Сделать обобщенный вывод по лабораторной работе.

Таблица 2.1 – Варианты алгоритмов

|  |  |
| --- | --- |
| **№** | **Задача и алгоритм** |
| 12 | Задача раскраски графа (100 вершин, степень вершины не более 20, но не менее 1), пчелиный алгоритм (начальные решения найти жадным алгоритмом, число пчел 30 из них 2 разведчики) |

# Выполнение

## Программная реализация алгоритма

### Исходный код

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

class InputFile:

"""Create random input file for the graph"""

def \_\_init\_\_(self, n=100, min=1, max=20, file='inputs/input2.txt'):

"""

Init n = number of vertexes in the graph, min = minimum power of the

vertex, max = maximum power of the vertex, file = name of the output

file

"""

self.n = n

self.min = min

self.max = max

self.file = file

def create(self):

"""Create input file"""

# Generate random list of correspondence using setted limitations

list = []

for i in range(self.n):

num\_of\_edjes = np.random.randint(1, self.max+1)

all\_vertexes = np.arange(self.n)

np.random.shuffle(all\_vertexes)

neighbors = all\_vertexes[:num\_of\_edjes]

for neighbor in neighbors:

list.append([i, neighbor])

# Save matrix to the file

with open(self.file, 'w') as f:

f.write(str(self.n) + "\n")

for key, value in list:

f.write(str(key) + " " + str(value) + "\n")

class Algorithms:

"""Greedy + Bees algorithms to coloring a graph"""

def \_\_init\_\_(self, file="inputs/input2.txt"):

"""Read data to build the graph from file"""

with open(file, 'r') as f:

lst = f.readlines()

n = int(lst[0])

lst = lst[1:]

lst = [list(map(int, elem.split())) for elem in lst]

self.n = n

self.lst = lst

def getNeighbors(self, vertex):

"""Get list of neighbors for the vertex"""

result = []

for key, val in self.lst:

if key == vertex:

result.append(val)

return result

def allNeigborsNotInSameColor(self, vertex, vertex\_colors, current\_color):

"""Check are all neighbors has colors not the same as the vertex"""

neighbors = self.getNeighbors(vertex)

neighbors\_not\_in\_same\_color = [

vertex\_colors[neighbor] != current\_color

for neighbor in neighbors

]

if sum(neighbors\_not\_in\_same\_color) == \

len(neighbors\_not\_in\_same\_color):

return True

return False

def maxPowerVertex(self):

"""Find vertex in the graph with max number of edjes"""

elements = [row[0] for row in self.lst]

max\_val = max(set(elements), key=elements.count)

return max\_val

def getAvailableColor(

self, neighbors,

vertex\_colors,

num\_colors,

old\_color

):

"""Get first available color which no neighbor has"""

available\_colors = [color for color in range(num\_colors)]

for neighbor in neighbors:

color = vertex\_colors[neighbor]

if color in available\_colors:

available\_colors.remove(color)

if old\_color in available\_colors:

available\_colors.remove(old\_color)

if (len(available\_colors) != 0):

return available\_colors[0]

return -1

def tryToReduceNumOfColors(self, vertex, vertex\_colors, num\_colors):

"""

Try to reduce number of colors for the every neighbor of the current

vertex

"""

neighbors = self.getNeighbors(vertex)

for neighbor in neighbors:

temp\_colors = vertex\_colors.copy()

# Swap color with the neighbor

temp\_colors[vertex], temp\_colors[neighbor] = \

temp\_colors[neighbor], temp\_colors[vertex]

# Check is swap is legal

if self.allNeigborsNotInSameColor(

neighbor, temp\_colors, temp\_colors[neighbor]

):

# Try to reduce number of colors

new\_color = self.getAvailableColor(

self.getNeighbors(neighbor),

temp\_colors,

num\_colors,

vertex\_colors[neighbor]

)

# If any alternative color is ok => repaint

if new\_color != -1:

temp\_colors[neighbor] = new\_color

vertex\_colors = temp\_colors.copy()

return vertex\_colors

def removeDuplicateEdjes(self, lst):

"""Removes one of duplicate edjes such as 1-3, 3-1"""

new\_list = []

for key, val in lst:

if [key, val] and [val, key] not in new\_list:

new\_list.append([key, val])

return new\_list

#takes input from the file and creates a undirected graph

def createGraph(self):

G = nx.Graph()

lst = self.removeDuplicateEdjes(self.lst)

for row in lst:

G.add\_edge(row[0], row[1])

return G

#draws the graph and displays the weights on the edges

def drawGraph(self, G,col\_val, vertex\_colors):

pos = nx.spring\_layout(G)

colors = ['red', 'blue', 'yellow', 'purple', 'orange', 'black']

values = [colors[vertex\_colors[node]] for node in G.nodes()]

nx.draw(G, pos, with\_labels = True, node\_color = values, edge\_color = 'black' ,width = 1, alpha = 0.7)

def showGraph(self, vertex\_colors):

"""Show graph"""

print("Vertex colors:", vertex\_colors)

print("Number of colors:", len(set(vertex\_colors)))

# G = self.createGraph()

# col\_val = {}

# for i in range(7):

# col\_val[i] = 1

#

# self.drawGraph(G, col\_val, vertex\_colors)

# plt.show()

def greedy(self):

"""Apply Greedy algorithm"""

current\_color = 0

# List of colors for the every vertex. Note: -1 means no color

vertex\_colors = [-1 for \_ in range(self.n)]

while sum([val == -1 for val in vertex\_colors]) != 0:

for vertex in range(self.n):

if vertex\_colors[vertex] == -1:

neighbors = self.getNeighbors(vertex)

if self.allNeigborsNotInSameColor(

vertex, vertex\_colors, current\_color

):

vertex\_colors[vertex] = current\_color

current\_color += 1

return vertex\_colors, current\_color

def bees(self):

"""Apply Bees algorithm"""

# Apply Greedy algorithm

vertex\_colors, num\_colors = self.greedy()

# Draw graph coloring with Greedy algorithm

self.showGraph(vertex\_colors)

print(sum(vertex\_colors))

# Find a vertex with the maximum power (start vertex)

vertex = self.maxPowerVertex()

# List of the vertexes to process

next = [vertex]

counter = 0

parent = -1

RANDOMNESS = 9

mutation = RANDOMNESS

while len(next) < 1000:

vertex = next[counter]

neighbors = self.getNeighbors(vertex)

for neighbor in neighbors:

if neighbor != parent:

next.append(neighbor)

if mutation == 0:

next.append(np.random.randint(0, self.n+1))

mutation = RANDOMNESS

parent = vertex

counter += 1

mutation -= 1

# print(vertex\_colors)

# print(sum(vertex\_colors))

counter = 2

for vertex in next:

vertex\_colors = self.tryToReduceNumOfColors(

vertex,

vertex\_colors,

num\_colors

)

# print(vertex\_colors)

if (counter % 20 == 0):

print(sum(vertex\_colors))

counter += 1

# Draw result graph coloring with Bees algorithm

self.showGraph(vertex\_colors)

def main():

# fill = InputFile(file='inputs/big.txt')

# fill.create()

algorithms = Algorithms(file="inputs/big.txt")

algorithms.bees()

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

main()

### Примеры работы

На рисунках 3.1 и 3.2 показаны примеры работы программы.

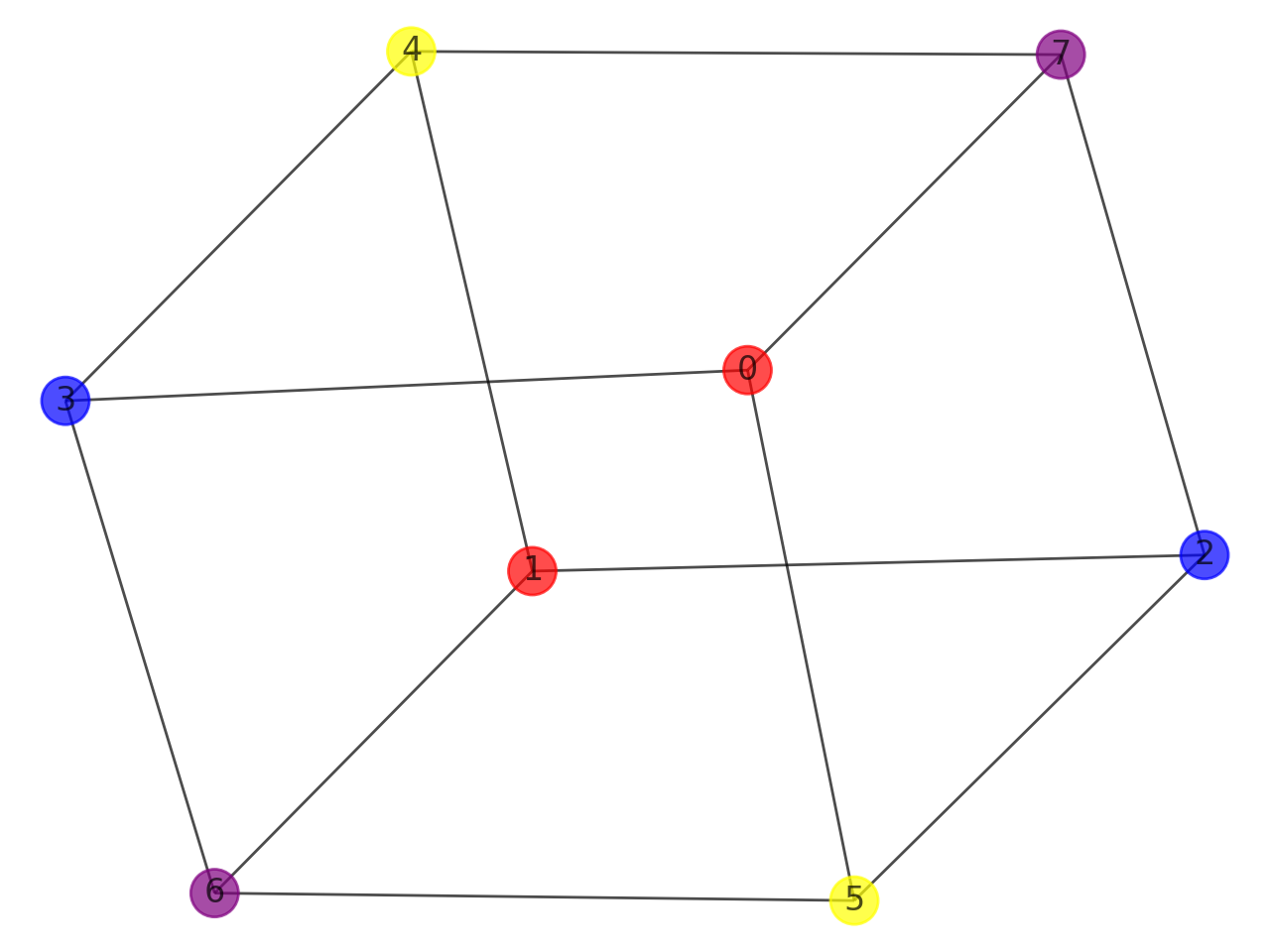


Рисунок 3.1 – начальный граф, разукрашенный Жадным алгоритмом

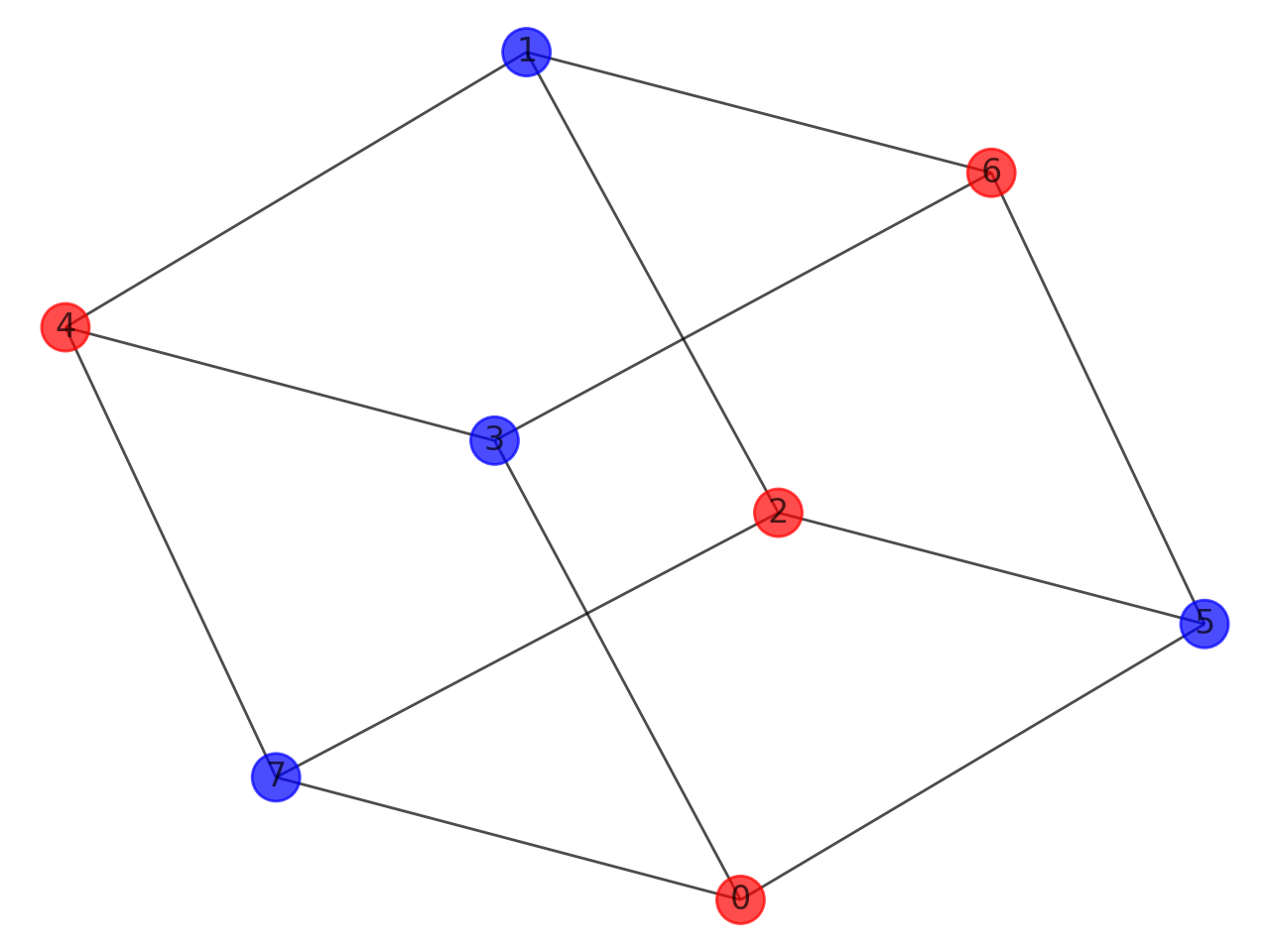


Рисунок 3.2 – Граф после работы Алгоритма пчелиного роя

## Испытания алгоритма

### Значения целевой функции с ростом итераций

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Value | Iteration | Value | Iteration | Value |
| 0 | 179 | 340 | 158 | 680 | 146 |
| 20 | 208 | 360 | 160 | 700 | 150 |
| 40 | 192 | 380 | 163 | 720 | 146 |
| 60 | 188 | 400 | 164 | 740 | 159 |
| 80 | 180 | 420 | 149 | 760 | 155 |
| 100 | 181 | 440 | 169 | 780 | 156 |
| 120 | 179 | 460 | 152 | 800 | 156 |
| 140 | 172 | 480 | 160 | 820 | 157 |
| 160 | 175 | 500 | 152 | 840 | 150 |
| 180 | 165 | 520 | 155 | 860 | 152 |
| 200 | 183 | 540 | 151 | 880 | 153 |
| 220 | 184 | 560 | 147 | 900 | 152 |
| 240 | 181 | 580 | 140 | 920 | 156 |
| 260 | 172 | 600 | 150 | 940 | 148 |
| 280 | 162 | 620 | 140 | 960 | 158 |
| 300 | 167 | 640 | 144 | 980 | 152 |
| 320 | 171 | 660 | 141 | 1000 | 150 |

### Графики зависимости решения от числа итераций

На рисунке 3.3 показан график отображающий качество полученного решения.

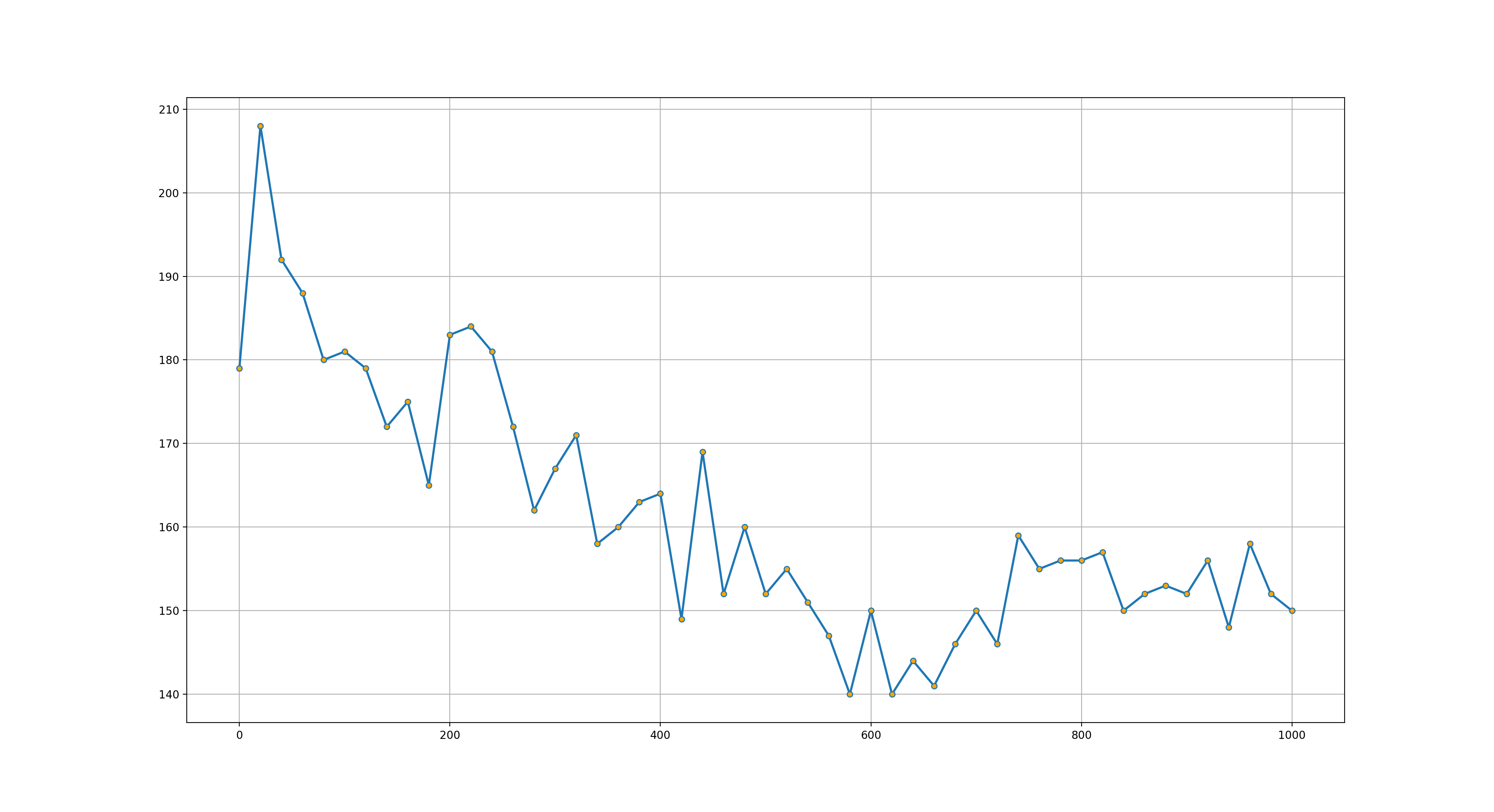


Рисунок 3.3 – Графики зависимости решения от числа итераций

Выводы

В рамках данной лабораторной работы я попытался реализовать Алгоритм пчелиной колонии (Artificial Bee Colony algorithm). Для небольших графов, где Жадный алгоритм ведет себе плохо, хорошо видно преимущество Пчелиного алгоритма, хотя работает он, конечно же, намного более медленно, чем Жадный.