LAPORAN

PRAKTIKUM KECERDASAN BUATAN

Oleh :

Nama	**:**	**Rizqillah**
NIM	**:**	**1957301020**
Kelas	**:**	**TI 2C**
Dosen Pembimbing	**:**	**Muhammad Arhami, S.Si, M.Kom**

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI DAN KOMPUTER

POLITEKNIK NEGERI LHOKSEUMAWE

2021

LEMBAR PENGESAHAN

No. Praktikum	:	02/P.KB/2C/TI/2021
Judul Praktikum	:	Simulated Annealing
Tanggal Praktikum	:	29 Maret 2021
Tanggal Pengumpulan Laporan	:	05 April 2021
Nama Praktikan / NIM	:	Rizqillah / 1957301020
Kelas	:	TI 2C
Nilai	:	

Buketrata, 05 April 2021

Dosen Pengampu,

Muhammad Arhami, S.Si, M.Kom
NIP. 19741029 200003 1 001

DAFTAR ISI

LEMBAR PENGESAHAN i
DAFTAR ISI ii
DAFTAR GAMBAR iii
BAB I 1
1.1 Latar Belakang 1
1.2 Tujuan Praktikum 2
1.3 Manfaat Praktikum 2
BAB II 3
2.1 Simulated Annealing (SA) 3
BAB III 6
3.1 Alat Dan Bahan 6
3.2 Rangkaian Percobaan dan Hasil Percobaan 6
3.2.1 Traveling Salesman Problem 6
BAB IV 18
4.1 Simulated Annealing 18
4.2 Analisis 18
4.2.1 Program Java : 18
4.2.2 Program Python : 24
BAB V 32
4.3 Kesimpulan 32
4.4 Saran 32
DAFTAR PUSTAKA 33

DAFTAR GAMBAR

Gambar 1. Program Java Class City.java 7
Gambar 2. Program Java Class TourManager.java 7
Gambar 3. Program Java Class Tour.java 9
Gambar 4. Program Java Class Main SimulatedAnnealing.java 11
Gambar 5. Hasil Program Java Simulated Annealing 12
Gambar 6. Program Python tps_utils.py 12
Gambar 7. Program Python animated_visualizer.py 13
Gambar 8. Program Python simulated_annealing.py 15
Gambar 9. Program Python nodes_generator.py 16
Gambar 10. Program Python tsp.py 16
Gambar 11. Hasil Program Python Simulated Annealing 16
Gambar 12. Peta Perjalanan hasil program Python 17
Gambar 13. Grafik hasil Program Python 17

BAB I
PENDAHULUAN

1.1 Latar Belakang

Solusi optimal untuk masalah pengoptimalan tertentu bisa menjadi tugas yang sangat sulit untuk dapat ditemukan dengan baik, seringkali secara praktis hal tersebut tidak mungkin. Ini terjadi karena masalah yang besar dan kompleks, sehingga perlu mencari melalui sejumlah besar kemungkinan-kemungkinan solusi untuk menemukan solusi yang optimal, bahkan dengan kekuatan komputasi modern, masih banyak solusi yang mungkin untuk dipertimbangkan, karena dalam kasus tertentu tidak dapat secara realistis berharap dapat menemukan solusi yang optimal dan masuk akal dalam jangka waktu tertentu, sehingga harus puas dengan sesuatu yang didapatkan dari hasil pencarian solusi tersebut. (Lee Jacobson, 2013)

Salah satu contoh masalah pengoptimalan yang biasanya memiliki banyak kemungkinan solusi adalah masalah TSP (Travelling Salesman Problem). Untuk menemukan solusi masalah TSP, perlu menggunakan algoritma yang dapat menemukan solusi yang cukup baik dalam waktu yang wajar. (Lee Jacobson, 2013)

Simulated Annealing adalah teknik optimal numerik dengan prinsip thermo-dynamic. Annealing adalah proses di mana material solid dilebur dan didinginkan secara perlahan-lahan dengan mengurangi temperatur. Partikel dari material berusaha menyusun dirinya selama proses pendinginan.

Simulated Annealing adalah metode probabilistik yang diusulkan oleh Kirkpatrick, Gelett dan Vecchi (1983) dan Cerny (1985) telah menemukan minimum global dari fungsi biaya yang mungkin memiliki beberapa minimum lokal. Ia bekerja dengan meniru proses fisik di mana padatan secara perlahan didinginkan sehingga ketika pada akhirnya strukturnya “dibekukan”, ini terjadi pada konfigurasi energi minimum. Mereka membatasi diri pada kasus fungsi biaya yang ditentukan pada himpunan terbatas. Perluasan Simulated Annealing untuk kasus fungsi yang didefinisikan pada set continue juga telah diperkenalkan dalam literatur (misalnya, Geman dan Hwang, 1986; Gidas, 1985; Holley, Kusuoka dan Stroock, 1989; Jeng dan Woods, 1990; Kushner, 1985).

Simulated Annealing dikembangkan pada tahun 1983. Abramson(1991) mengemukakan bahwa aplikasi Simulated Annealing pada masalah penjadwalan relatif berbanding lurus. Atom-atom diganti dengan elemen. Elemen adalah kombinasi tertentu dari dosen, matakuliah, ruang dan kelas.

1.2 Tujuan Praktikum

Adapun tujuan dari penelitian ini di antaranya yaitu :

1. Mampu mengimplementasikan suatu sistem dengan algorithm - simulated annealing (SA).
2. Menganalisis kinerja algoritma SA. Perbandingan dilakukan dengan membandingkan percobaan-percobaan yang menggunakan parameter optimal untuk masing-masing algoritma.

1.3 Manfaat Praktikum

Manfaat yang diharapkan setelah mengikuti praktikum ini yaitu :

1. Dapat mengimplementasi metode algoritma Simulated Annealing dalam kehidupan sehari-hari dan dalam pembuatan aplikasi yang membutuhkan metode seperti metode dalam algoritma Simulated Annealing ini.
2. Mengetahui kegunaan dari algoritma Simulated Annealing dengan jelas, dan dapat menerapkan algoritma tersebut.
3. Memperoleh pengetahuan lebih banyak mengenai metode algoritma Simulated Annealing tersebut.
4. Mengetahui lebih jelas mengenai metode pencarian solusi menggunakan algoritma Simulated Annealing.

BAB II
TINJAUAN PUSTAKA

2.1 Simulated Annealing (SA)

(Freitas,2014) mengatakan bahwa Algoritma adalah jantung ilmu komputer atau informatika. Banyak cabang ilmu komputer yang diacu dalam terminologi algoritma. Namun, jangan beranggapan algoritma selalu identik dengan ilmu komputer saja. Dalam kehidupan sehari-haripun banyak terdapat terdapat proses yang dinyatakan dalam suatu algoritma. Cara-cara membuat membuat kue atau masakan yang dinyatakan dalam suatu resep juga dapat disebut sebagai algoritma. Pada setiap resep selalu ada urutan langkah-langkah membuat masakan. Bila langkah-langkahnya tidak logis, tidak dapat dihasilkan masakan yang diinginkan.

Prosedur penggunaan simbol-simbol pada diagram alir harus sesuai dengan maksud dan tujuan alir yang ingin dibuat. Simbol pada diagram alir diantarannya adalah Terminal, persiapan, pengolahan/proses, keputusan, input/output dan garis. Simbol terminal berbentuk persegi panjang dengan sisi yang berbentuk agak oval, berfungsi untuk menunjukan awal dan akhir dari program. Simbol persiapan berfungsi untuk memberikan nilai awal pada suatu variabel. Simbol pengolahan/proses untuk pengolahan arithmatika dan pemindahan data. Simbol keputusan untuk mewakili operasi perbandingan logika. Dan symbol input/output digunakan untuk menyatakan proses input/baca atau output/tulis. (Chen 1993)

Pengertian Annealing menurut Dowsland (1993) adalah bahan padat yang dipanaskan melampaui titik cairnya dan kemudian didinginkan kembali menjadi keadaan padat. Kirkpatrick dkk (1983) menyarankan menggunakan SA (Simulated Annealing) untuk optimisasi, yang diterapkan pada desain VLSI (Very Large Scale Integration) dan TSP (Travelling Salesman Problem).

Simulated Annealing (SA) merupakan suatu pendekatan algoritma untuk memecahkan masalah optimasi kombinatorial. Simulated Annealing (SA) dapat dipandang sebagai versi yang disempurnakan dari metode perbaikan iteratif (yang berulang) dimana solusi awal ditingkatkan berulang-kali dengan membuat perubahan kecil hingga ditemukan solusi yang lebih baik. Simulated Annealing (SA) mengacak prosedur pencarian lokal dan dalam beberapa kasus memungkinkan untuk melakukan perubahan solusi yang memperburuk. Ini merupakan upaya untuk mengurangi kemungkinan terjebak dalam solusi optimal lokal.

Algoritma Simulated Annealing (SA) merupakan suatu pendekatan yang efisien untuk memecahkan masalah kombinatorial yang sulit. Algoritma SA mengeluarkan minimum lokal dengan menggunakan bilangan acak dalam pemilihan perpindahan. Pada setiap iterasi dari algoritma Simulated Annealing, perpindahan dipilih secara acak dan perubahan biaya dihitung untuk perpindahan.

Mari kita lihat cara kerja Simulated Annealing, dan mengapa ini bagus dalam menemukan solusi khususnya untuk TSP. Algoritma Simulated Annealing pada awalnya terinspirasi dari proses annealing pada pengerjaan logam. Simulated Annealing melibatkan pemanasan dan pendinginan material untuk mengubah sifat fisiknya karena perubahan struktur internalnya. Saat logam mendingin, struktur barunya menjadi tetap, akibatnya menyebabkan logam mempertahankan sifat yang baru diperolehnya. Simulated Annealing menyimpan variabel suhu untuk mensimulasikan proses pemanasan ini. Awalnya ditetapkan tinggi dan kemudian dibiarkan 'mendingin' secara perlahan saat algoritma berjalan. Meskipun variabel suhu ini tinggi, algoritma akan diizinkan, dengan lebih banyak frekuensi, untuk menerima solusi yang lebih buruk daripada solusi saat ini. Hal ini memberi peluang kemampuan algoritma untuk keluar dari optimal lokal apa pun yang ditemukannya sendiri di awal eksekusi. (Lee Jacobson, 2013)

(Lee Jacobson, 2013) mengatakan local optimum merupakan nilai optimal yang dapat dicapai oleh sebuah algoritma berada dalam rentang nilai tertentu yang telah dibatasi. Contoh algoritma yang menerapkan konsep local optimum adalah Neural Network. Hasil dari metode local optimum biasanya bergantung dari nilai minimal dan maximal.

Ketika suhu berkurang, begitu juga peluang untuk menerima solusi yang lebih buruk, oleh karena itu memungkinkan algoritma untuk secara bertahap fokus pada area ruang pencarian di mana diharapkan, solusi yang mendekati optimal dapat ditemukan. Proses 'pendinginan' bertahap inilah yang membuat algoritma SA tersimulasi sangat efektif dalam menemukan solusi yang mendekati optimal saat menangani masalah besar yang berisi banyak optimal lokal. Sifat dari masalah TSP menjadikannya contoh yang mudah dipahami.

Global optimum adalah nilai optimal yang didapat oleh sebuah algoritma merupakan nilai optimal dari keseluruhan rentang input data. Proses pencarian global optimum pun melibatkan keseluruhan data yang akan diproses atau dicari. Contoh algoritmanya adalah SVM (Support Vector Machine).

Ada beberapa parameter yang mengatur algoritma SA, yaitu :

1. Temperatur awal (Initial Temperature)
2. *Move*
3. Penghentian pengerjaan (Termination Condition)
4. Jadwal pendinginan - Coolingrate

Algoritma dari metode Simulated Annealing yaitu :

1. Mulai dari keadaan awal. Lakukan pengujian. Jika merupakan keadaan tujuan, maka berhenti. Jika tidak, jadikan keadaan sekarang sebagai keadaan awal.
2. Mulailah BEST-SO-FAR untuk keadaan sekarang.
3. Mulailah T sesuai Annealing Schedule.
4. Ulangi hingga mendapatkan solusi atau tak ada lagi operator baru yang akan diaplikasikan pada keadaan sekarang.
   a. Pilih operator yang belum digunakan, dan gunakan untuk menghasilkan keadaan yang baru.
   b. Evaluasi keadaan baru tersebut.
      E = (nilai keadaan sekarang) – (nilai keadaan baru)
      - Jika keadaan baru tersebut merupakan tujuan maka berhenti.
      - Jika bukan merupakan tujuan, tetapi lebih baik dari keadaan sekarang, maka jadikan sebagai keadaan sekarang. Gunakan juga BEST-SO-FAR untuk keadaan baru ini.
      - Jika ternyata lebih jelek dari keadaan sekarang maka jadikan keadaan sekarang dengan probabilitas p” seperti yang didefinisikan diatas. Langkah ini biasanya diimplementasikan dengan meminta pembangkit angka acak untuk menghasilkan angka dalam range [0,1]. Jika angka tersebut lebih kecil dari p” maka move diterima. Jika berlaku sebaliknya, tak terjadi apapun.
   c. Periksa T seperlunya sesuai Annealing Schedule.
5. BEST-SO-FAR menjadi jawabannya.

BAB III
LANGKAH-LANGKAH PRAKTIKUM

3.1 Alat Dan Bahan

a. PC atau Laptop
b. Aplikasi VisualStudio Code, Python, NetBeans IDE, Java Jdk/Jre
c. Ekstensi Python di Software VisualStudio Code

3.2 Rangkaian Percobaan dan Hasil Percobaan

Adapun rangkaian percobaan yang telah dilakukan adalah sebagai berikut :

3.2.1 Traveling Salesman Problem

Dengan Bahasa Pemrograman :

a. Java

Program :

City.java

```
/*
* City.java
* Models a city
 */
public class City {

    int x;
    int y;

    // Constructs a randomly placed city
    public City() {
        this.x = (int) (Math.random() * 200);
        this.y = (int) (Math.random() * 200);
    }

    // Constructs a city at chosen x, y location
    public City(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // Gets city's x coordinate
    public int getX() {
        return this.x;
    }

    // Gets city's y coordinate
    public int getY() {
```

```
        return this.y;
    }

    // Gets the distance to given city
    public double distanceTo(City city) {
        int xDistance = Math.abs(getX() - city.getX());
        int yDistance = Math.abs(getY() - city.getY());
        double distance = Math.sqrt((xDistance * xDistance) +
(yDistance * yDistance));

        return distance;
    }

    @Override
    public String toString() {
        return getX() + ", " + getY();
    }
}
```

Gambar 1. Program Java Class City.java

TourManager.java

```
import java.util.ArrayList;

public class TourManager {

    // Holds our cities
    private static ArrayList destinationCities = new
ArrayList<City>();

    // Adds a destination city
    public static void addCity(City city) {
        destinationCities.add(city);
    }

    // Get a city
    public static City getCity(int index) {
        return (City) destinationCities.get(index);
    }

    // Get the number of destination cities
    public static int numberOfCities() {
        return destinationCities.size();
    }
}
```

Gambar 2. Program Java Class TourManager.java

Tour.java

```
import java.util.ArrayList;
import java.util.Collections;

public class Tour {

    // Holds our tour of cities
    private ArrayList tour = new ArrayList<City>();
    // Cache
    private int distance = 0;

    // Constructs a blank tour
    public Tour() {
        for (int i = 0; i < TourManager.numberOfCities(); i++) {
            tour.add(null);
        }
    }

    // Constructs a tour from another tour
    public Tour(ArrayList tour) {
        this.tour = (ArrayList) tour.clone();
    }

    // Returns tour information
    public ArrayList getTour() {
        return tour;
    }

    // Creates a random individual
    public void generateIndividual() {
        // Loop through all our destination cities and add them
to our tour
        for (int cityIndex = 0; cityIndex <
TourManager.numberOfCities(); cityIndex++) {
            setCity(cityIndex, TourManager.getCity(cityIndex));
        }
        // Randomly reorder the tour
        Collections.shuffle(tour);
    }

    // Gets a city from the tour
    public City getCity(int tourPosition) {
        return (City) tour.get(tourPosition);
    }

    // Sets a city in a certain position within a tour
    public void setCity(int tourPosition, City city) {
        tour.set(tourPosition, city);
```

```
        // If the tours been altered we need to reset the fitness
and distance
        distance = 0;
    }

    // Gets the total distance of the tour
    public int getDistance() {
        if (distance == 0) {
            int tourDistance = 0;
            // Loop through our tour's cities
            for (int cityIndex = 0; cityIndex < tourSize();
cityIndex++) {
                // Get city we're traveling from
                City fromCity = getCity(cityIndex);
                // City we're traveling to
                City destinationCity;
                // Check we're not on our tour's last city, if we
are set our
                // tour's final destination city to our starting
city
                if (cityIndex + 1 < tourSize()) {
                    destinationCity = getCity(cityIndex + 1);
                } else {
                    destinationCity = getCity(0);
                }
                // Get the distance between the two cities
                tourDistance +=
fromCity.distanceTo(destinationCity);
            }
            distance = tourDistance;
        }
        return distance;
    }

    // Get number of cities on our tour
    public int tourSize() {
        return tour.size();
    }

    @Override
    public String toString() {
        String geneString = "|";
        for (int i = 0; i < tourSize(); i++) {
            geneString += getCity(i) + "|";
        }
        return geneString;
    }
}
```

Gambar 3. Program Java Class Tour.java

SimulatedAnnealing.java

```
public class SimulatedAnnealing {
    public static double acceptanceProbability(int energy, int
newEnergy, double temperature) {
        if (newEnergy < energy) {
            return 1.0;
        }
        return Math.exp((energy - newEnergy) / temperature);
    }

    public static void main(String[] args) {
        City city = new City(60, 200);
        TourManager.addCity(city);
        City city2 = new City(180, 200);
        TourManager.addCity(city2);
        City city3 = new City(80, 180);
        TourManager.addCity(city3);
        City city4 = new City(140, 180);
        TourManager.addCity(city4);
        City city5 = new City(20, 160);
        TourManager.addCity(city5);
        City city6 = new City(100, 160);
        TourManager.addCity(city6);
        City city7 = new City(200, 160);
        TourManager.addCity(city7);
        City city8 = new City(140, 140);
        TourManager.addCity(city8);
        City city9 = new City(40, 120);
        TourManager.addCity(city9);
        City city10 = new City(100, 120);
        TourManager.addCity(city10);
        City city11 = new City(180, 100);
        TourManager.addCity(city11);
        City city12 = new City(60, 80);
        TourManager.addCity(city12);
        City city13 = new City(120, 80);
        TourManager.addCity(city13);
        City city14 = new City(180, 60);
        TourManager.addCity(city14);
        City city15 = new City(20, 40);
        TourManager.addCity(city15);
        City city16 = new City(100, 40);
        TourManager.addCity(city16);
        City city17 = new City(200, 40);
        TourManager.addCity(city17);
        City city18 = new City(20, 20);
        TourManager.addCity(city18);
        City city19 = new City(60, 20);
        TourManager.addCity(city19);
```

```
        City city20 = new City(160, 20);
        TourManager.addCity(city20);
        double temp = 10000;

        double coolingRate = 0.003;

        Tour currentSolution = new Tour();
        currentSolution.generateIndividual();

        System.out.println("Initial solution distance: " +
currentSolution.getDistance());

        Tour best = new Tour(currentSolution.getTour());

        while (temp > 1) {
            Tour newSolution = new
Tour(currentSolution.getTour());

            int tourPos1 = (int) (newSolution.tourSize() *
Math.random());
            int tourPos2 = (int) (newSolution.tourSize() *
Math.random());

            City citySwap1 = newSolution.getCity(tourPos1);
            City citySwap2 = newSolution.getCity(tourPos2);
            newSolution.setCity(tourPos2, citySwap1);
            newSolution.setCity(tourPos1, citySwap2);
            int currentEnergy = currentSolution.getDistance();
            int neighbourEnergy = newSolution.getDistance();


            if (acceptanceProbability(currentEnergy,
neighbourEnergy, temp) > Math.random()) {
                currentSolution = new
Tour(newSolution.getTour());
            }
            if (currentSolution.getDistance() <
best.getDistance()) {
                best = new Tour(currentSolution.getTour());
            }
            temp *= 1 - coolingRate;
        }

        System.out.println("Final solution distance: " +
best.getDistance());
        System.out.println("Tour: " + best);
    }
}
```

Gambar 4. Program Java Class Main SimulatedAnnealing.java

Hasil :

```
Initial solution distance: 1992
Final solution distance: 890
Tour:  |100,  120|40,  120|20,  160|60,  200|80,  180|100,  160|140,
140|140,  180|180,  200|200,  160|180,  100|180,  60|200,  40|160,
20|100, 40|60, 20|20, 20|20, 40|60, 80|120, 80|
```

Gambar 5. Hasil Program Java Simulated Annealing

b. Python

Program :

tsp_utils.py

```
import math
import random
import numpy as np

def vectorToDistMatrix(coords):
    return np.sqrt((np.square(coords[:, np.newaxis] -
 coords).sum(axis=2)))
def nearestNeighbourSolution(dist_matrix):
    node = random.randrange(len(dist_matrix))
    result = [node]
    nodes_to_visit = list(range(len(dist_matrix)))
    nodes_to_visit.remove(node)


    while nodes_to_visit:
        nearest_node = min([(dist_matrix[node][j], j) for j in no
des_to_visit], key=lambda x: x[0])
        node = nearest_node[1]
        nodes_to_visit.remove(node)
        result.append(node)

    return result
```

Gambar 6. Program Python tps_utils.py

Animated_visualizer.py

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np

def animateTSP(history, points):
    key_frames_mult = len(history) // 1500
    fig, ax = plt.subplots()
```

```
    line, = plt.plot([], [], lw=2)

    def init():
        x = [points[i][0] for i in history[0]]
        y = [points[i][1] for i in history[0]]
        plt.plot(x, y, 'co')

        extra_x = (max(x) - min(x)) * 0.05
        extra_y = (max(y) - min(y)) * 0.05
        ax.set_xlim(min(x) - extra_x, max(x) + extra_x)
        ax.set_ylim(min(y) - extra_y, max(y) + extra_y)
        line.set_data([], [])
        return line,

    def update(frame):
        x = [points[i, 0] for i in history[frame] + [history[fram
e][0]]]
        y = [points[i, 1] for i in history[frame] + [history[fram
e][0]]]
        line.set_data(x, y)
        return line
    ani = FuncAnimation(fig, update, frames=range(0, len(history)
, key_frames_mult),
                        init_func=init, interval=3, repeat=False)

    plt.show()
```

Gambar 7. Program Python animated_visualizer.py

Simulated_annealing.py

```
import math
import random
import matplotlib.pyplot as plt
import tsp_utils
import animated_visualizer


class SimulatedAnnealing:
    def __init__(self, coords, temp, alpha, stopping_temp, stoppi
ng_iter):
        self.coords = coords
        self.sample_size = len(coords)
        self.temp = temp
        self.alpha = alpha
        self.stopping_temp = stopping_temp
```

```
        self.stopping_iter = stopping_iter
        self.iteration = 1

        self.dist_matrix = tsp_utils.vectorToDistMatrix(coords)
        self.curr_solution = tsp_utils.nearestNeighbourSolution(s
elf.dist_matrix)
        self.best_solution = self.curr_solution

        self.solution_history = [self.curr_solution]

        self.curr_weight = self.weight(self.curr_solution)
        self.initial_weight = self.curr_weight
        self.min_weight = self.curr_weight

        self.weight_list = [self.curr_weight]

        print('Intial weight: ', self.curr_weight)

    def weight(self, sol):
        return sum([self.dist_matrix[i, j] for i, j in zip(sol, s
ol[1:] + [sol[0]])])

    def acceptance_probability(self, candidate_weight):
        return math.exp(-abs(candidate_weight -
 self.curr_weight) / self.temp)

    def accept(self, candidate):
        candidate_weight = self.weight(candidate)
        if candidate_weight < self.curr_weight:
            self.curr_weight = candidate_weight
            self.curr_solution = candidate
            if candidate_weight < self.min_weight:
                self.min_weight = candidate_weight
                self.best_solution = candidate

        else:
            if random.random() < self.acceptance_probability(cand
idate_weight):
                self.curr_weight = candidate_weight
                self.curr_solution = candidate

    def anneal(self):
        while self.temp >= self.stopping_temp and self.iteration
< self.stopping_iter:
            candidate = list(self.curr_solution)
```

```
            l = random.randint(2, self.sample_size - 1)
            i = random.randint(0, self.sample_size - l)

            candidate[i: (i + l)] = reversed(candidate[i: (i + l)
])

            self.accept(candidate)
            self.temp *= self.alpha
            self.iteration += 1
            self.weight_list.append(self.curr_weight)
            self.solution_history.append(self.curr_solution)

        print('Minimum weight: ', self.min_weight)
        print('Improvement: ',
              round((self.initial_weight -
 self.min_weight) / (self.initial_weight), 4) * 100, '%')

    def animateSolutions(self):
        animated_visualizer.animateTSP(self.solution_history, sel
f.coords)

    def plotLearning(self):
        plt.plot([i for i in range(len(self.weight_list))], self.
weight_list)
        line_init = plt.axhline(y=self.initial_weight, color='r',
 linestyle='--')
        line_min = plt.axhline(y=self.min_weight, color='g', line
style='--')
        plt.legend([line_init, line_min], ['Initial weight', 'Opt
imized weight'])
        plt.ylabel('Weight')
        plt.xlabel('Iteration')
        plt.show()
```

Gambar 8. Program Python simulated_annealing.py

Nodes_generator.py

```
import random
import numpy as np

class NodeGenerator:
    def __init__(self, width, height, nodesNumber):
        self.width = width
        self.height = height
        self.nodesNumber = nodesNumber
```

```
    def generate(self):
        xs = np.random.randint(self.width, size=self.nodesNumber)
        ys = np.random.randint(self.height, size=self.nodesNumber
)

        return np.column_stack((xs, ys))
```

Gambar 9. Program Python nodes_generator.py

Tsp.py

```
from nodes_generator import NodeGenerator
from simulated_annealing import SimulatedAnnealing

def main():
    temp = 1000
    stopping_temp = 0.00000001
    alpha = 0.9995
    stopping_iter = 10000000
    size_width = 200
    size_height = 200
    population_size = 70

    nodes = NodeGenerator(size_width, size_height, population_siz
e).generate()

    sa = SimulatedAnnealing(nodes, temp, alpha, stopping_temp, st
opping_iter)

    sa.anneal()
    sa.animateSolutions()
    sa.plotLearning()

if __name__ == "__main__":
    main()
```

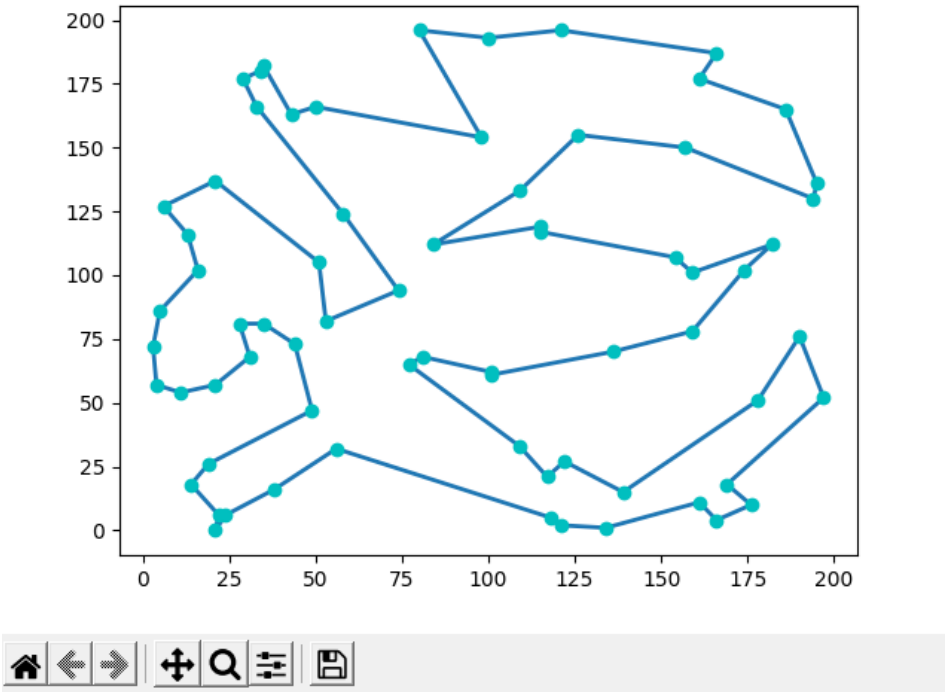
Gambar 10. Program Python tsp.py

Hasil :

```
Intial weight:  1555.819688383805
Minimum weight:  1532.6667585886662
Improvement:  1.49 %
```

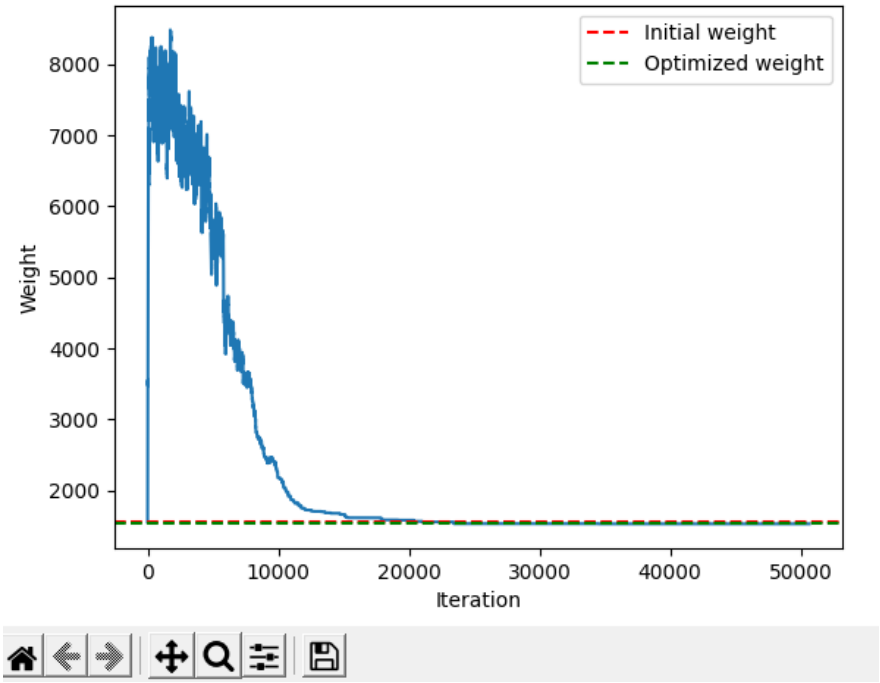
Gambar 11. Hasil Program Python Simulated Annealing

Grafik Perjalanan :



Gambar 12. Peta Perjalanan hasil program Python

Grafik Maximum & Minimum :



Gambar 13. Grafik hasil Program Python

BAB IV
PEMBAHASAN DAN ANALISA HASIL

4.1 Simulated Annealing

Simulated Annealing adalah algoritma pengoptimalan penelusuran global stokastik. Ini berarti menggunakan keacakan sebagai bagian dari proses pencarian. Hal ini membuat algoritma sesuai untuk fungsi objektif non-linier di mana algoritma penelusuran lokal lainnya tidak beroperasi dengan baik.

4.2 Analisis

4.2.1 Program Java :

```
City city = new City(60, 200);
TourManager.addCity(city);
```

➔ Membuat objek dari class city, dan mengisi nilai parameter dari method construct class city.

➔ Memanggil method addCity dengan mengisi parameter dari objek city yang ada di class TourManager

```
public class City {
  int x;
  int y;
```

➔ Deklarasi variabel global class City dengan nama x dan y bertipe int

```
public City() {
  this.x = (int) (Math.random() * 200);
  this.y = (int) (Math.random() * 200);
}
```

➔ Membuat method construct dengan nilai random class City

```
public City(int x, int y) {
  this.x = x;
  this.y = y;
}
```

➔ Membuat method construct yang akan menerima nilai parameter variabel x dan y local. Dan nilai tersebut akan diisi kedalam variabel x dan y global

```
public class TourManager {
 private static ArrayList destinationCities = new ArrayList<City>();
public static void addCity(City city) {
 destinationCities.add(city);
}
```

- Membuat class TourManager dan objek dari ArrayList dengan nama destinationCities yang diisi dengan nilai ArrayList dari variabel City
- Kemudian membuat method addCity yang menerima parameter variabel city dengan tipe data non-primitif(tipe data class).
- Memanggil method add dari class ArrayList dan mengisi nilai dari variabel city local di parameter melalui objek destinationCities. Ini berfungsi menambahkan nilai city kedalam list.

```
double temp = 10000;
double coolingRate = 0.003;
```

- Membuat variabel temp dengan nilai 10000 bertipe data double
- Membuat variabel coolingRate dengan nilai 0.003 bertipe data double

```
Tour currentSolution = new Tour();
currentSolution.generateIndividual();
System.out.println("Initial solution distance: " + currentSolution.g
etDistance());
```

- Membuat objek dari class Tour bernama currentSolution dengan isian dari method construct Tour
- Memanggil method generateIndividual dengan objek currentSolution
- Mencetak nilai hasil pemanggilan method getDistance dengan objek currentSolution

```
public class Tour {
  private ArrayList tour = new ArrayList<City>();
  private int distance = 0;
```

- Membuat class Tour. Membuat variabel tour dari ArrayList dengan isinya nilai objek ArrayList dan diisi nilai dari City
- Membuat variabel distance dengan nilai 0 dengan tipe data int

```
public Tour() {
  for (int i = 0; i < TourManager.numberOfCities(); i++) {
      tour.add(null);
  }
}
```

- Membuat method construct class Tour
- Melakukan perulangan selama nilai i=0 lebih kecil dari nilai kembalian method numberOfCities yang dipanggil melalui class TourManager.
- Melakukan penambahan nilai pada objek tour = null.

```
public static int numberOfCities() {
  return destinationCities.size();
}
```

➔ Membuat method numberOfCities yang bertipe int dengan mengembalikan nilai size melalui objek destinationCities yang bertipe ArrayList.

```
public void generateIndividual() {
  for (int cityIndex = 0; cityIndex < TourManager.numberOfCities();
cityIndex++) {
        setCity(cityIndex, TourManager.getCity(cityIndex));
  }
  Collections.shuffle(tour);
}
```

➔ Membuat method generateIndividual. Kemudian melakukan perulangan selama nilai cityIndex=0 lebih kecil dari nilai kembalian method numberOfCities yang dipanggil melalui class TourManager

➔ Kemudian memanggil method setCity yang diisi dengan nilai dari cityIndex, dan nilai kembalian method getCity yang diisi parameter cityIndex melalui class TourManager.

➔ Kemudian memanggil nilai random dari shuffle dengan isian parameter variabel tour menggunakan Collections.

```
public static City getCity(int index) {
  return (City) destinationCities.get(index);
}
```

➔ Membuat method yang mengembalikan nilai dari method get dengan isian parameter index. Method tersebut berasal dari class ArrayList.

```
public int getDistance() {
  if (distance == 0) {
      int tourDistance = 0;
```

➔ Membuat method getDistance. Kemudian mengecek apakah distance sama dengan 0. Jika iya, maka variabel tourDistance diisi dengan 0.

```
for (int cityIndex = 0; cityIndex < tourSize(); cityIndex++) {
  City fromCity = getCity(cityIndex);
  City destinationCity;
```

➔ Mengecek apakah nilai cityIndex=0 lebih kecil dari method tourSize. Jika iya maka akan mengisi nilai ke variabel fromCity dengan nilai dari method getCity yang diisi parameter cityIndex. Dan membuat variabel destinationCity.

```
if (cityIndex + 1 < tourSize()) {
  destinationCity = getCity(cityIndex + 1);
} else {
  destinationCity = getCity(0);
}
```

- Mengecek apakah nilai cityIndex+1 lebih kecil dari method tourSize. Jika iya maka nilai variabel destinationCity diisi dengan nilai method getCity dengan parameter cityIndex+1.
- Jika tidak maka nilai destinationCity diisi dengan nilai getCity dengan parameter 0.

```
            tourDistance += fromCity.distanceTo(destinationCity);
        }
        distance = tourDistance;
    }
    return distance;
}
```

- Melakukan penambahan terhadap nilai tourDistance dengan penambahan dari nilai kembalian method distanceTo yang diisi parameter nilai destinationCity.
- Mengatur nilai distance = tourDistance. Kemudian mengembalikan nilai distance.

```
public int tourSize() {
  return tour.size();
}
```

- Membuat method tourSize dengan mengembalikan panjang dari objek ArrayList dengan nama tour.

```
public City getCity(int tourPosition) {
  return (City) tour.get(tourPosition);
}
```

- Membuat method yang menerima parameter variabel tourPosition. Dan mengembalikan nilai hasil method get dari objek tour dengan mengisi parameter dari tourPosition.

```
public double distanceTo(City city) {
  int xDistance = Math.abs(getX() - city.getX());
  int yDistance = Math.abs(getY() - city.getY());
  double distance = Math.sqrt((xDistance * xDistance) + (yDistance *
 yDistance));
```

- Membuat methdo distanceTo yang menerima parameter variabel city.

➔ Mengambil hasil kembalian math.abs dari variabel x dan y yang diisi kedalam variabel yDistance dan xDistance. Math.abs berfungsi sebagai pengubah nilai menjadi absolut.

➔ Mengambil hasil nilai kuadrat dari math.sqrt (xDistance*xDistance) + (yDistance*yDistance) kedalam variabel distance bertipe data double.

```
return distance;
```

➔ Mengembalikan hasil dari distance ke si pemanggil method

```
Tour best = new Tour(currentSolution.getTour());
```

➔ Membuat variabel dari class Tour dengan nama best dan mengisi nilai kedalam method construct dengan nilai hasil dari method getTour yang dipanggil melalui objek currentSolution.

```
public ArrayList getTour() {
  return tour;
}
```

➔ Membuat method getTour di class Tour yang mengembalikan nilai dari tour.

```
public Tour(ArrayList tour) {
  this.tour = (ArrayList) tour.clone();
}
```

➔ Membuat method construct dari class Tour dengan menerima parameter variabel tour.

➔ Dan nilai membuat variabel tour global menerima nilai dari nilai tour local yang di clone.

```
while (temp > 1) {
  Tour newSolution = new Tour(currentSolution.getTour());
```

➔ Melakukan perulangan selama nilai temp lebih besar dari 1. Atau selama sistemnya menjadi dingin

➔ Membuat objek dari class Tour dengan nama newSolution

```
int tourPos1 = (int) (newSolution.tourSize() * Math.random());
int tourPos2 = (int) (newSolution.tourSize() * Math.random());
```

➔ Membuat variabel tourPos1 dan 2 bertipe int yang diisi dengan nilai hasil perkalian nilai tourSize * nilai random.

```
City citySwap1 = newSolution.getCity(tourPos1);
City citySwap2 = newSolution.getCity(tourPos2);
```

➔ Membuat objek dari class City yang diisi dengan nilai method getCity dengan parameter tourPos1 dan 2 yang diisi kedalam objek citySwap1 dan 2.

```
newSolution.setCity(tourPos2, citySwap1);
newSolution.setCity(tourPos1, citySwap2);
```

➔ Memanggil method setCity melalui objek newSolution dan mengisi nilai parameter dari tourPos2 dan citySwap1 dan memanggil method setCity lagi dengan nilai parameter tourPos1 dan citySwap2

```
int currentEnergy = currentSolution.getDistance();
int neighbourEnergy = newSolution.getDistance();
```

➔ Mengisi nilai variabel currentEnergy dengan hasil kembalian method getDistance melalui objek currentSolution

➔ Mengisi nilai variabel neighbourEnergy dengan hasil kembalian method getDistance melalui objek newSolution.

```
if (acceptanceProbability(currentEnergy, neighbourEnergy, temp) > Ma
th.random()) {
  currentSolution = new Tour(newSolution.getTour());
}
```

➔ Melakukan pengecekan apakah nilai kembalian method acceptanceProbability yang diisi dengan nilai parameter currentEnergy, neighbourEnergy dan temp lebih besar dari nilai random.

➔ Jika iya maka akan mengisi nilai pada variabel currentSolution dengan nilai class Tour pada method getTour melalui objek newSolution

```
public static double acceptanceProbability(int energy, int newEnergy
, double temperature) {
  if (newEnergy < energy) {
      return 1.0;
  }
  return Math.exp((energy - newEnergy) / temperature);
}
```

➔ Membuat method acceptanceProbability yang menerima parameter energy, newEnergy dan temperature

➔ Mengecek apakah nilai newEnergy lebih besar dari energy, jika iya maka akan me-return nilai 1.0

➔ Dan jika tidak maka akan mereturn hasil math.exp dari (energy – newEnergy)/temperature. Math.exp berfungsi untuk menghitung hasil dari e^x dimana x adalah argumen yang diberikan. e merupakan logaritma natural dengan nilai 2.718.

```
if (currentSolution.getDistance() < best.getDistance()) {
  best = new Tour(currentSolution.getTour());
}
```

➔ Mengecek apakah nilai kembalian method getDistance melalui objek currentSolution lebih kecil dari nilai kembalian method getDistance melalui objek best.
➔ Jika iya maka akan mengisi nilai objek best dengan nilai kembalian method getTour melalui objek currentSolution.

```
temp *= 1 - coolingRate;
```
➔ Mengatur nilai variabel temp menjadi temp = temp * 1 – coolingRate

```
System.out.println("Final solution distance: " +best.getDistance());
System.out.println("Tour: " + best);
```
➔ Mencetak nilai kembalian method getDistance melalui objek best.
➔ Dan mencetak nilai dari objek best.

4.2.2 Program Python :

```
from nodes_generator import NodeGenerator
from simulated_annealing import SimulatedAnnealing
```
➔ Mengimport file node_generator sebagai NodeGenerator. Dan file simulated_annealing sebagai SimulatedAnnealing

```
def main():
    temp = 1000
    stopping_temp = 0.00000001
    alpha = 0.9995
    stopping_iter = 10000000
```
➔ Membuat function main. Dan membuat beberapa variabel beserta nilainya.

```
    size_width = 200
    size_height = 200
```
➔ Membuat variabel size_width dan size_height dengan nilai 200.

```
    population_size = 70
```
➔ Membuat variabel population_size dengan nilai 70.

```
nodes = NodeGenerator(size_width, size_height, population_size).gene
rate()
```
➔ Membuat objek dari NodeGenerator dengan mengisi nilai construct dari variabel size_width, size_height dan population_size. Kemudian memanggil method generate.

```
import random
import numpy as np
```
➔ Memanggil library random dan library numpy sebagai np.

```
class NodeGenerator:
    def __init__(self, width, height, nodesNumber):
        self.width = width
        self.height = height
        self.nodesNumber = nodesNumber
```
➔ membuat class NodeGenerator dan method construct yang menerima variabel self, width, height dan nodesNumber

➔ mengisi nilai variabel global dengan variabel local dari method construct

```
    def generate(self):
        xs = np.random.randint(self.width, size=self.nodesNumber)
        ys = np.random.randint(self.height, size=self.nodesNumber)

        return np.column_stack((xs, ys))
```
➔ membuat method generate yang menerima nilai self(classnya sendiri)

➔ mengisi nilai random dari parameter width, size dan nodesNumber ke variabel xs dan ys

➔ kemudian me-return nilai numpy dari method colum_stack dengan parameter xs dan ys

```
sa = SimulatedAnnealing(nodes, temp, alpha, stopping_temp, stopping_
iter)
sa.anneal()
```
➔ Membuat objek dari SimulatedAnnealing dengan mengisi nilai parameter method construct dengan variabel nodes, temp, alpha, stopping_temp dan stopping_iter

➔ Memanggil method anneal dari objek sa.

```
class SimulatedAnnealing:
    def __init__(self, coords, temp, alpha, stopping_temp, stopping_
iter):
        self.coords = coords
        self.sample_size = len(coords)
        self.temp = temp
        self.alpha = alpha
        self.stopping_temp = stopping_temp
        self.stopping_iter = stopping_iter
        self.iteration = 1
```
➔ Membuat class SimulatedAnnealing dan method construct dengan menerima parameter self, coords, temp, alpha, stopping_temp, dan stopping_iter.

- Membuat nilai variabel global menjadi nilai dari variabel local.
- Membuat variabel global iteration dengan nilai 1.

```
        self.dist_matrix = tsp_utils.vectorToDistMatrix(coords)
        self.curr_solution = tsp_utils.nearestNeighbourSolution(self
.dist_matrix)
        self.best_solution = self.curr_solution
        self.solution_history = [self.curr_solution]
```

- Mengisi nilai dari pemanggilan method vectorToDistMatrix yang diisi dengan parameter coord dan diisi kedalam variabel global dist_matrix
- Mengisi nilai dari kembalian method nearestNeighbourSolution yang diisi dengan parameter global dist_matrix kedalam variabel global curr_solution
- Membuat variabel global best_solution menjadi nilai dari variabel global curr_solution
- Membuat variabel global solution_history menjadi nilai dari variabel global curr_solution

```
        self.curr_weight = self.weight(self.curr_solution)
        self.initial_weight = self.curr_weight
        self.min_weight = self.curr_weight
        self.weight_list = [self.curr_weight]
        print('Intial weight: ', self.curr_weight)
```

- Membuat variabel global curr_weight menjadi nilai global dari weight yang diisi nilai parameter curr_solution
- Membuat variabel global initial_weight menjadi bernilai variabel global curr_weight
- Dan variabel min_weight menjadi curr_weight. Dan variabel weight_list menjadi nilai curr_weight
- Dan mencetak nilai curr_weight kelayar sebagai nilai inisial weight

```
def anneal(self):
        while self.temp >= self.stopping_temp and self.iteration < s
elf.stopping_iter:
            candidate = list(self.curr_solution)
            l = random.randint(2, self.sample_size - 1)
            i = random.randint(0, self.sample_size - l)
```

- Membuat method anneal. Dan melakukan perulangan selama nilai temp lebih besar sama dengan nilai stopping_temp dan nilai iteration lebih besar dari stopping_iter
- Mengatur nilai candidate menjadi nilai ketika melakukan list variabel curr_solution
- Mengatur nilai l dan i dari nilai random dengan parameter 2 dan 0 dan sample_size – 1.

```
            candidate[i: (i + l)] = reversed(candidate[i: (i + l)])
            self.accept(candidate)
            self.temp *= self.alpha
            self.iteration += 1
            self.weight_list.append(self.curr_weight)
            self.solution_history.append(self.curr_solution)

        print('Minimum weight: ', self.min_weight)
        print('Improvement: ',
              round((self.initial_weight -
 self.min_weight) / (self.initial_weight), 4) * 100, '%')
```

- ➔ Mengatur nilai candidate di index i : i+1 menjadi nilai reversed dari nilainya.
- ➔ Memanggil method accept yang diisi parameter variabel candidate
- ➔ Mengatur nilai temp = temp * alpha
- ➔ Mengatur nilai iteration = iteration + 1
- ➔ Memanggil append yang diisi parameter curr_weight melalui weight_list
- ➔ Memanggil append yang diisi parameter curr_solution melalui solution_history
- ➔ Mencetak nilai dari min_weight
- ➔ Dan mencetak nilai yang telah di round dari ((initial_weigh – min_weight) / (initial_weight) dan 4) * 100

```
import math
import random
import numpy as np


def vectorToDistMatrix(coords):
    return np.sqrt((np.square(coords[:, np.newaxis] -
 coords).sum(axis=2)))
```

- ➔ Memanggil library math, random dan numpy dengan inisialisasi sebagai np
- ➔ Membuat method vectorToDistMatrix yang menerima parameter coords
- ➔ Dan mengembalikan hasil kuadrat dari nilai numpy square pada coords dan numpy newaxis – coords. Dan penjumlahan nilai axis=2

```
def nearestNeighbourSolution(dist_matrix):
    node = random.randrange(len(dist_matrix))
    result = [node]

    nodes_to_visit = list(range(len(dist_matrix)))
    nodes_to_visit.remove(node)
```

- ➔ Membuat method nearestNeighbourSolution yang menerima parameter matrix

- Membuat variabel node diisi dengan nilai random dari range nilai matrix
- Mengisi variabel result dengan nilai node
- Mengisi nilai nodes_to_visit menjadi list dari panjang nilai matrix
- Menghapus nilai variabel nodes_to_visit pada parameter node

```
    while nodes_to_visit:
        nearest_node = min([(dist_matrix[node][j], j) for j in nodes
_to_visit], key=lambda x: x[0])
        node = nearest_node[1]
        nodes_to_visit.remove(node)
        result.append(node)

    return result
```

- Melakukan perulangan selama nilai nodes_to_visit true.
- Mengisi nilai nearest_node dengan nilai minimal dist_matrix pada node dan j. Dan melakukan perulangan selama nilai j didalam nodes_to_visit dan nilai key=lambda x dan x pada index 0.
- Mengisi nilai node dengan nearest_node index 1
- Menghapus nilai nodes_to_visit pada index node
- Memanggil append dengan parameter node dari result
- Me-return nilai result

```
sa.animateSolutions()
```

- Memanggil method animateSolutions melalui objek sa dari class Simulated_annealing

```
def animateSolutions(self):
  animated_visualizer.animateTSP(self.solution_history, self.coords)
```

- Membuat method animateSolutions dengan parameter classnya sendiri.
- Memanggil method animateTSP dengan mengisi nilai parameter dari solution_history dan coords pada class animated_visualizer.

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np
def animateTSP(history, points):
    key_frames_mult = len(history) // 1500
```

- Memanggil library pylot pada package matplotlib sebagai plt. Dan library animation pada package matplotlib sebagai FuncAnimation. Dan memanggil library numpy sebagai np
- Membuat method animateTSP yang menerima parameter history dan points
- Mengatur variabel key_frames_mult dengan nilai len panjang nilai history melakukan operator bitwise dengan nilai 1500

```
    fig, ax = plt.subplots()
    line, = plt.plot([], [], lw=2)
```

➔ Membuat nilai fig, ax menjadi nilai dari method subplots dari library matplotlib

➔ Membuat nilai line dari method plot library matplotlib dan beberapa matrix

```
    def init():
        x = [points[i][0] for i in history[0]]
        y = [points[i][1] for i in history[0]]
        plt.plot(x, y, 'co')
```

➔ Membuat method init dengan menginisialisasi nilai dari node dot x dan y.

```
        extra_x = (max(x) - min(x)) * 0.05
        extra_y = (max(y) - min(y)) * 0.05
        ax.set_xlim(min(x) - extra_x, max(x) + extra_x)
        ax.set_ylim(min(y) - extra_y, max(y) + extra_y)

        line.set_data([], [])
        return line,
```

➔ Mengatur nilai extra_x dan extra_y dengan nilai max(y atau x) – min(y atau x) * 0.05

➔ Mengatur nilai set_xlim dan set_ylim pada ax menjadi min(y atau x) – extra_x d atau extra_y. Dan max(y atau x) + extra_x atau extra_y

➔ Mengatur set_data pada line. Dan me-return nilai line

```
sa.plotLearning()
```

➔ Memanggil method plotLearning dengan objek sa dari class Simulated_annealing

```
def plotLearning(self):
    plt.plot([i for i in range(len(self.weight_list))], self.weight_
list)
    line_init = plt.axhline(y=self.initial_weight, color='r', linest
yle='--')
    line_min = plt.axhline(y=self.min_weight, color='g', linestyle='
--')
    plt.legend([line_init, line_min], ['Initial weight', 'Optimized
weight'])
        plt.ylabel('Weight')
        plt.xlabel('Iteration')
        plt.show()
```

- Membuat method plotLearning dengan parameter class nya sendiri
- Mengatur nilai plot dengan perulangan for i dalam range panjang nilai weight_list dan dengan kondisi panjang nilai weight_list
- Mengatur nilai line_init dan line_min menjadi nilai axhline dari initial_weight dan min_weight dengan color dari red dan green. Dengan linestyle garis-garis
- Memanggil legend pada matplotlib dengan parameter line_ini dan line_min. Beserta dengan kata-kata Initial weight dan Optimized weight.
- Memanggil ylabel pada plt dengan nilai kata Weight. Dan xlabel dengan nilai Iteration
- Dan menampilkan library pylot dari matplotlib tersebut.

```
def weight(self, sol):
        return sum([self.dist_matrix[i, j] for i, j in zip(sol, sol[
1:] + [sol[0]])])
```

- Membuat method weight yang memiliki parameter self dan sol
- Mengembalikan nilai hasil penjumlah pada dist_matrix index i dan j
- Melakukan perulangan nilai i dan j dalam zip sol dan sol index 1 dan sol index 0

```
def acceptance_probability(self, candidate_weight):
   return math.exp(-abs(candidate_weight -
self.curr_weight) / self.temp)
```

- Membuat method acceptance_probability yang menerima parameter self dan candidate_weight
- Dan mengembalikan nilai hasil math.exp dari nilai absolute (candidate_weight - curr_weight) / temp

```
def accept(self, candidate):
        candidate_weight = self.weight(candidate)
        if candidate_weight < self.curr_weight:
```

- Membuat method accept yang menerima parameter self dan candidate
- Nilai candidate_weight diisi dengan nilai dari method weight yang diisi parameter candidate
- Mengecek apakah nilai candidate_weight lebih kecil dari nilai curr_weight

```
            self.curr_weight = candidate_weight
            self.curr_solution = candidate
            if candidate_weight < self.min_weight:
                self.min_weight = candidate_weight
                self.best_solution = candidate
```

- Membuat nilai curr_weight menjadi nilai dari candidate_weight
- Membuat nilai curr_solution menjadi nilai candidate
- Mengecek apakah nilai candidate_weight lebih kecil dari nilai min_weight
- Dan jika iya, maka mengisi nilai min_weight dengan nilai dari candidate_weight
- Membuat nilai best_solution menjadi nilai dari candidate

```
        else:
            if random.random() < self.acceptance_probability(candida
te_weight):
                self.curr_weight = candidate_weight
                self.curr_solution = candidate
```

- Jika nilai candidate_weight < curr_weight bernilai false. Maka akan mengecek apakah nilai random lebih kecil dari nilai kembalian method acceptance_probability yang diisi dengan parameter dari candidate_weight.
- Membuat nilai curr_weight menjadi nilai candidate_weight
- Membuat nilai curr_solution menjadi nilai candidate

BAB V
KESIMPULAN DAN SARAN

4.3 Kesimpulan

Pada praktikum yang telah dilakukan, dapat disimpulkan bahwa untuk melakukan pencarian jarak terdekat pada suatu aplikasi Travelling Salesman Problem juga dapat dilakukan melalui metode algoritma Simulated Annealing. Pada algoritma ini menggunakan metode ibarat sebuah alat yang awalnya dipanaskan, kemudian didinginkan hingga mendapat solusi yang diinginkan.

Simulated Annealing adalah teknik optimalisasi numerik dengan prinsip thermo-dynamic. Annealing adalah proses dimana material solid dilebur dan didinginkan secara perlahan-lahan dengan mengurangi temperatur.

Simulated Annealing adalah metode probabilistik yang diusulkan oleh Kirkpatrick, Gelett dan Vecchi (1983) dan Cerny (1985) yang telah menemukan minimum global dari fungsi biaya yang mungkin memiliki beberapa minimum lokal. Ia bekerja dengan meniru proses fisik di mana padatan secara perlahan didinginkan sehingga ketika pada akhirnya strukturnya “dibekukan”, ini terjadi pada konfigurasi energi minimum. Mereka membatasi diri pada kasus fungsi biaya yang ditentukan pada himpunan terbatas. Perluasan Simulated Annealing untuk kasus fungsi yang didefinisikan pada set continue juga telah diperkenalkan dalam literatur (misalnya, Geman dan Hwang, 1986; Gidas, 1985; Holley, Kusuoka dan Stroock, 1989; Jeng dan Woods, 1990; Kushner, 1985).

4.4 Saran

Saran untuk pengembangan lebih lanjut dari metode-metode algoritma pencarian solusi yang memuaskan dari metode algoritma Simulated Annealing lebih lanjut. Dan saran untuk pengembangan lebih lanjut menggunakan metode algoritma Simulated Annealing dalam kehidupan sehari-hari dan dalam pembuatan suatu aplikasi yang menggunakan pengimplementasian pencarian solusi yang diinginkan agar lebih banyak menggunakan algoritma Simulated Annealing.

DAFTAR PUSTAKA

[1] Shi-hua Zhan, Juan Lin, Ze-jun Zhang, dan Yi-wen Zhong. 10 Februari 2016. https://www.hindawi.com/journals/cin/2016/1712630/. College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China.

[2] Lee Jacobson. "Simulated Annealing for Beginner". 11 April 2013. https://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6.

[3] Cesar William Alvarenga. "How to Implement Simulated Annealing Algorithm in Python". 13 September 2020. https://medium.com/swlh/how-to-implement-simulated-annealing-algorithm-in-python-ab196c2f56a0.

[4] Dimitris Bertsimas & John N. Tsitsiklis. February 1993. https://www.researchgate.net/publication/38363197_Simulated_Annealing

[5] Chen, J. C. and Y. Z. Kou. 1993. Accumulation of Ammonia in the Haemolymph of Penaenus Monodon Exposed to Ambient Ammonia. Aquacultur.

[6] Freitas TAC, Widyastuti N, dan Iswahyudi C. 2014. Perancangan dan Pengembangan Sistem Informaasi pada Toko Bunga Amai di Dili Timor Leste Berbasis Web.

[7] Lilian Besson. Simulated Annealing in Python. 20 Juli 2017. https://perso.crans.org/besson/publis/notebooks/Simulated_annealing_in_Python.html. MIT Licences.

[8] GeeksForGeeks. "Simulated Annealing". 11 September 2019. https://www.geeksforgeeks.org/simulated-annealing/.

[9] Programmer Sought. "Simulated Annealing Algorithm". https://www.programmersought.com/article/42414556053/.

[10] Emmanuel Goosaert. "Simulated Annealing Applied to the Travelling Salesman Problem". 06 April 2010. https://codecapsule.com/2010/04/06/simulated-annealing-traveling-salesman/. CodeCapsule.

[11] Vinay Varma. "Simulated Annealing for Clustering Problem". 22 Oktober 2018. https://towardsdatascience.com/simulated-annealing-for-clustering-problems-part-1-3fa8994a3ebb.

[12] Arhami. M., 2006, Algoritma Simulasi Annealing, Tesis, Magister Ilmu Komputer UGM, Yogyakarta.

[13] Jason Brownlee. "Simulated Annealing from Scratch in Python". 19 Februari 2021. https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python.