DAFTAR PUSTAKA

Andriansyah, A., Novatama, R., & Sentia, P. D. (2020). Algoritma Simulated Annealing untuk Menentukan Rute Kendaraan Heterogen (Studi Kasus). *Jurnal Teknologi Informasi dan Ilmu Komputer*, *7*(5), 933-942.

Budi Utomo, P., & Setiafindari, W. (2021). Optimasi Penjadwalan Produksi Menggunakan Metode Simulated Annealing di Industri XYZ.

Cahyadi, F., Ong, J. O., & Kosasih, J. S. (2011). Perancangan algoritma simulated annealing untuk rute kendaraan yang mempertimbangkan backhaul, rute majemuk, dan time window. *Jurnal Telematika*, *7*(1).

Cahyaningsih, W. K., Sari, E. R., & Hernawati, K. (2015). Penyelesaian capacitated vehicle routing problem (cvrp) menggunakan algoritma sweep untuk optimasi rute distribusi surat kabar kedaulatan rakyat. In *Seminar Nasional Matematika dan Pendidikan Matematika UNY* (pp. 1-8).

Christian, S. J. (2011). Analisis Sistem Pengangkutan Sampah Kota Makassar Dengan Metode Penyelesaian Vehicle Routing Problem (VRP)(Studi Kasus: Kecamatan Mamajang). *Skripsi. Program Studi Teknik Industri. Jurusan Mesin. Fakultas Teknik. Universitas Hasanuddin. Makassar*.

Dermawan, D., Lahming, L., & S. Mandra, M. A. (2018). Kajian Strategi Pengelolaan Sampah. UNM Environmental Journals, 1(3), 88. https://doi.org/10.26858/uej.v1i3.8074

DIA, R. T. (2014). A Course on Meta-Heuristic Search Methods for Combinatorial Optimization Problems.

Ensiklopedia. Metaheuristik. Diakses pada 12 Mei 2024, dari Metaheuristik (stekom.ac.id)

Husen, M., Masudin, I., & Utama, D. M. (2015). Penjadwalan job shop statik dengan metode simulated annealing untuk meminimasi waktu makespan.

Spektrum Industri, *13*(2), 115.

Indonesia, S. N., & Nasional, B. S. (2002). Tata cara teknik operasional pengelolaan sampah perkotaan. *Badan Standarisasi Nasional, Jakarta*.

Indonesia. Undang-Undang Nomor 18 Tahun 2008 tentang Pengelolaan Sampah. Sekretariat Negara. Jakarta.

Indonesia. Peraturan Pemerintah Nomor 81 Tahun 2012 tentang Penyelenggaraan Pengelolaan Sampah. Sekretariat Negara. Jakarta.

Kemenkeu.go.id. (2022, 30 Maret). Pengelolaan Sampah Rumah Tangga dan Sampah Sejenis Sampah Rumah Tangga. Diakses pada 15 Februari 2024, dari Pengelolaan Sampah di Indonesia (kemenkeu.go.id).

Kbbi.go.id. Diakses pada 5 Mei 2024 2024, dari Hasil Pencarian - KBBI VI Daring (kemdikbud.go.id) dan Hasil Pencarian - KBBI VI Daring (kemdikbud.go.id) .

Komala, P. S., Aziz, R., & Ramadhani, F. (2012). Analisis Produktivitas Sistem Transportasi Sampah Kota Padang. *Dampak*, *9*(2), 73-86.

Lukman, M., & Hasibuan, M. (2021). Penerapan Metode Simulated Annealing Untuk Penjadwalan Perkuliahan. *Computer Technology and Information Systems*, *5*(2), 25-38.

Mahyudin, R. P. (2017). Kajian permasalahan pengelolaan sampah dan dampak lingkungan di TPA (Tempat Pemrosesan Akhir). *Jukung (Jurnal Teknik Lingkungan)*, *3*(1).

Muhaddad, R. A., & Santosa, B. (2014). ,. Pengembangan algoritma simulated annealing untuk penyelesaian permasalahan alokasi pada closed loop supply chain (CLSC). *Institut Technology Sepuluh Nopember*.

Notoadmodjo, S. (2003). Prinsip-prinsip dasar ilmu kesehatan masyarakat. *Jakarta: Rineka Cipta*.

Permana, E. R., Midyanti, D. M., & Hidayati, R. (2020). Optimasi pencarian rute terpendek distribusi barang menggunakan metode simulated annealing (Studi kasus: PD Bumi Jaya Indah Kota Pontianak). *Coding Jurnal Komputer dan Aplikasi*, *8*(3), 9-18.

Purwendah, E. K., & Periani, A. (2022). Kewajiban Masyarakat Dalam Pemeliharaan Kelestarian Lingkungan Hidup Melalui Pengelolaan Sampah Berbasis Masyarakat. *Jurnal Pacta Sunt Servanda*, *3*(2), 163-171.

Samana, E., Prihandono, B., & Noviani, E. (2015). Aplikasi simulated annealing untuk menyelesaikan travelling salesman problem. *Bimaster: Buletin Ilmiah Matematika, Statistika dan Terapannya*, *4*(01).

Saputra, R., & Pujotomo, D. (2019). Penyelesaian Vehicle Routing Problem Dengan Karakteristik Time Windows Dan Multiple Trips Menggunakan Metode Saving Matrix (Studi Kasus: PT. Coca Cola Bottling Indonesia-Wilayah Medan. *Industrial Engineering Online Journal*, *7*(4).

Siringoringo, H. (2005). Seri teknik riset operasional pemrograman linear. *Yogyakarta: Graha Ilmu*, 1-233.

Supriyadi, S., Mawardi, K., & Nalhadi, A. (2017, December). Minimasi Biaya Dalam Penentuan Rute Distribusi Produk Minuman Menggunakan Metode Savings Matrix. In *Prosiding Seminar Nasional Riset Terapan| SENASSET* (pp. 1-8).

Tampuyak, S., Anwar, C., & Sangadji, M. N. (2016). Analisis proyeksi pertumbuhan penduduk dan kebutuhan fasilitas persampahan di Kota Palu 2015-2025. *E Jurnal Katalogis*, *4*(4), 94-104.

Yuliastuti, G. E., Prabiantissa, C. N., & Rizki, A. M. (2021). Implementasi Simulated Annealing untuk Penentuan Rute pada Jaringan. *MATICS: Jurnal Ilmu Komputer dan Teknologi Informasi (Journal of Computer Science and Information Technology)*, *13*(2), 42-46.

Yunus, A. I., Sinaga, J., Herliana, E., Syaiful, A. Z., Satriawan, D., Sari, D., ... & Sari, N. P. (2022). Pengelolaan Sampah Organik dan Anorganik. *PTGlobal Eksekutif Teknologi, Sumatera Barat*.

LAMPIRAN

Lampiran 1. Kode Python Algoritma Simulated Annealing Truk 1

```
#TRUK 1
import random
import math
import pandas as pd
from tabulate import tabulate



# Definisi matriks jarak
matriks_jarak = {
    "0": {"1": 2.3, "2": 2.2, "3": 2.9, "4": 3.1, "5": 3.1, "6": 2.4, "7": 3.0, "x": 11.2},
    "1": {"0": 2.3, "2": 0.65, "3": 1.2, "4": 1.9, "5": 1.8, "6": 0.16, "7": 0.6, "x": 13.5},
    "2": {"0": 2.2, "1": 0.65, "3": 0.7, "4": 1.3, "5": 1.3, "6": 0.55, "7": 1, "x": 13.1},
    "3": {"0": 2.9, "1": 1.2, "2": 0.7, "4": 1, "5": 0.6, "6": 0.7, "7": 0.35, "x": 12.7},
    "4": {"0": 3.1, "1": 1.9, "2": 1.3, "3": 1, "5": 0.13, "6": 1.8, "7": 1.4, "x": 13.2},
    "5": {"0": 3.1, "1": 1.8, "2": 1.3, "3": 0.6, "4": 0.13, "6": 1.6, "7": 1, "x": 11.4},
    "6": {"0": 2.4, "1": 0.16, "2": 0.55, "3": 0.7, "4": 1.8, "5": 1.6, "7": 0.4, "x": 13.3},
    "7": {"0": 3, "1": 0.6, "2": 1, "3": 0.35, "4": 1.4, "5": 1, "6": 0.4, "x": 13.8},
    "x": {"0": 11.32, "1": 13.5, "2": 13.1, "3": 12.7, "4": 13.2, "5": 11.4, "6": 13.3, "7": 13.8},
}

volume_sampah = {
    "1": 480,
    "2": 1760,
    "3": 240,
    "4": 2400,
    "5": 1120,
    "6": 2500,
    "7": 520
}

# Definisi kapasitas maksimum truk dan jarak
kapasitas_maksimum_truk = 6000
jarak_maksimum = 30

def hitung_jarak(solusi):
    total_jarak = 0
    for i in range(len(solusi) - 1):
```

```
        titik_awal = solusi[i]
        titik_tujuan = solusi[i + 1]
        total_jarak += matriks_jarak[titik_awal][titik_tujuan]
    return total_jarak

def generate_tetangga(solusi):
    tetangga = solusi.copy()


    # Pastikan semua titik ada dalam solusi tetangga
    for node in matriks_jarak.keys():
        if node not in tetangga:
            idx = random.randint(1, len(tetangga) - 2)  # Hindari menghapus titik awal, x, dan akhir
            tetangga.insert(idx, node)

    # Jika solusi tetangga melebihi kapasitas, kurangi titik sampai sesuai
    while hitung_volume_sampah(tetangga) > kapasitas_maksimum_truk and hitung_jarak(tetangga) > jarak_maksimum:
        idx = random.randint(1, len(tetangga) - 2)  # Hindari menghapus titik awal atau akhir
        del tetangga[idx]

    # Menghitung total jarak baru
    total_jarak = hitung_jarak(tetangga)

    return tetangga, total_jarak

def hitung_volume_sampah(solusi):
    total_volume = sum(volume_sampah[sampah] for sampah in solusi if sampah in volume_sampah)
    return total_volume

def simulated_annealing(solusi_awal, matriks_jarak, volume_sampah, kapasitas_maksimum_truk, suhu_awal, laju_penurunan):
    solusi_sekarang = solusi_awal
    jarak_sekarang = hitung_jarak(solusi_sekarang)
    volume_sekarang = hitung_volume_sampah(solusi_sekarang)

    suhu = suhu_awal
    iterasi = 1
    tabel = [["Iterasi", "Suhu", "Solusi Sekarang", "Jarak Sekarang", "Volume Sekarang"]]
```

```
    # Menambahkan solusi awal ke dalam tabel sebelum memulai iterasi
    tabel.append([iterasi, f"{suhu:.2f}", solusi_sekarang, f"{jarak_sekarang:.2f}", f"{volume_sekarang:.2f}"])

    while suhu > 0.01:  # Kondisi terminasi
        for iterasi_dalam in range(5):  # Iterasi dalam dengan 5 iterasi
            tetangga, jarak_tetangga = generate_tetangga(solusi_sekarang)
            volume_tetangga = hitung_volume_sampah(tetangga)
            delta_f = jarak_tetangga - jarak_sekarang

            # Pastikan titik awal dan akhir adalah '0'
            if tetangga[0] == '0' and tetangga[-1] == '0' and tetangga[-2] == 'x':
                if volume_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <= jarak_maksimum and delta_f <= 0:
                    solusi_sekarang = tetangga
                    jarak_sekarang = jarak_tetangga
                    volume_sekarang = volume_tetangga

                elif volume_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <= jarak_maksimum and random.random() < math.exp(-(delta_f) / suhu):
                    solusi_sekarang = tetangga
                    jarak_sekarang = jarak_tetangga
                    volume_sekarang = volume_tetangga

                elif volume_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <= jarak_maksimum:
                    solusi_sekarang = solusi_awal
                    jarak_sekarang = hitung_jarak(solusi_sekarang)
                    volume_sekarang = hitung_volume_sampah(solusi_sekarang)

            iterasi += 1  # Increment counter iterasi

            # Tambahkan data ke dalam tabel
            tabel.append([iterasi, f"{suhu:.2f}", solusi_sekarang, f"{jarak_sekarang:.2f}", f"{volume_sekarang:.2f}"])

        suhu *= laju_penurunan # Update suhu

    return tabel, solusi_sekarang, jarak_sekarang, volume_sekarang
```

```
# Penggunaan contoh
suhu_awal = 100
laju_penurunan = 0.98

solusi_awal1 = ["0", "1", "6", "2", "5", "x", "0"]
solusi_awal2 = ["0", "1", "7", "4", "3", "5", "x", "0"]
original_seed = random.getstate()
random.seed(2)
tabel1, solusi_sekarang1, jarak_sekarang1, volume_sekarang1 = simulated_annealing(solusi_awal1, matriks_jarak, volume_sampah, kapasitas_maksimum_truk, suhu_awal, laju_penurunan)

solusi_terbaik1 = min(tabel1[1:], key=lambda x: (float(x[3]), -float(x[4])))
random.setstate(original_seed)
print("Solusi optimal Rute 1:")
for info_iter in tabel1[1:]:
    if info_iter[2] == solusi_terbaik1[2]:
        iterasi = info_iter[0]
        suhu = float(info_iter[1])
        solusi = info_iter[2]
        jarak = float(info_iter[3])
        volume = float(info_iter[4])
        print(f"Solusi optimal 1 terdapat pada iterasi {iterasi}, suhu {suhu}:")
        print(f"Rute: {solusi}, Volume: {volume}, Jarak: {jarak} km")
        # Menampilkan titik yang belum dikunjungi
        titik_belum_dikunjungi1 = [node for node in matriks_jarak.keys() if node not in solusi]
        print("Titik yang belum dikunjungi:", titik_belum_dikunjungi1)
        break

print()
original_seed = random.getstate()
random.seed(3)
tabel2, solusi_sekarang2, jarak_sekarang2, volume_sekarang2 = simulated_annealing(solusi_awal2, matriks_jarak, volume_sampah, kapasitas_maksimum_truk, suhu_awal, laju_penurunan)
# Ambil solusi terbaik dari setiap iterasi untuk rute 2 yang mencakup titik-titik yang belum dikunjungi oleh rute 1 dan titik-titik yang telah dikunjungi oleh rute 1
tabel2_difilter = [info for info in tabel2[1:] if set(info[2]) >= set(titik_belum_dikunjungi1)]

# Filter solusi yang memenuhi batasan kapasitas truk dan batasan jarak maksimum yang dapat ditempuh
```

```
solusi_fisibel2 = []
for info_iter in tabel2_difilter:
    solusi = info_iter[2]
    if (hitung_volume_sampah(solusi) <= kapasitas_maksimum_truk) and
(hitung_jarak(solusi) <= jarak_maksimum):
        solusi_fisibel2.append(info_iter)

# Ambil solusi terbaik yang memenuhi semua syarat
solusi_terbaik2 = min(solusi_fisibel2, key=lambda x: (float(x[3]), -float(x[4])))
random.setstate(original_seed)
print("Solusi optimal Rute 2:")
for info_iter in solusi_fisibel2:
    if info_iter == solusi_terbaik2:
        iterasi = info_iter[0]
        suhu = float(info_iter[1])
        solusi = info_iter[2]
        jarak = float(info_iter[3])
        volume = float(info_iter[4])
        print(f"Solusi optimal 2 terdapat pada iterasi {iterasi}, suhu {suhu}:")
        print(f"Rute: {solusi}, Volume: {volume}, Jarak: {jarak} km")
        titik_belum_dikunjungi2 = [node for node in matriks_jarak.keys() if node not in
solusi]
        print("Titik yang belum dikunjungi:", titik_belum_dikunjungi2)

        break
print()
```

Lampiran 2. Kode Python Algoritma Simulated Annealing Truk 2

```
#TRUK 2
import random
import math
import pandas as pd
from tabulate import tabulate
random.seed(1764)

# Definisi matriks jarak
matriks_jarak =  {
    "0": {"1": 3.2, "2": 3.2, "3": 3.3, "4": 3.3, "5": 3.4, "6": 2.9, "7": 4.2, "8": 3.7, "x": 11.2},
    "1": {"0": 3.2, "2": 0.28, "3": 0.4, "4": 0.19, "5": 0.35, "6": 0.28, "7": 0.75, "8": 1.1, "x":
13.7},
    "2": {"0": 3.2, "1": 0.28, "3": 0.75, "4": 0.5, "5": 0.35, "6": 0.55, "7": 0.5, "8": 0.6, "x":
13.4},
    "3": {"0": 3.3, "1": 0.4, "2": 0.75, "4": 0.04, "5": 0.85, "6": 0.13, "7": 0.95, "8": 1.3, "x":
13.8},
    "4": {"0": 3.3, "1": 0.19, "2": 0.5, "3": 0.04, "5": 0.85, "6": 0.13, "7": 1, "8": 1.3, "x":
12.8},
    "5": {"0": 3.4, "1": 0.35, "2": 0.35, "3": 0.85, "4": 0.85, "6": 0.45, "7": 0.85, "8": 1, "x":
11.8},
    "6": {"0": 2.9, "1": 0.28, "2": 0.55, "3": 0.13, "4": 0.13, "5": 0.45, "7": 1.4, "8": 1, "x":
11.8},
    "7": {"0": 4.2, "1": 0.75, "2": 0.5, "3": 0.95, "4": 1, "5": 0.85, "6": 1.4, "8": 0.15, "x":
13.1},
    "8": {"0": 3.7, "1": 1.1, "2": 0.6, "3": 1.3, "4": 1.3, "5": 1, "6": 1.5, "7": 0.15, "x": 13},
    "x": {"0": 11.2, "1": 13.7, "2": 13.4, "3": 13.8, "4": 12.8, "5": 11.8, "6": 13.7, "7": 13.1,
"8": 13},
}

volume_sampah = {
    "1": 2400,
    "2": 1200,
    "3": 1500,
```

```
    "4": 620,
    "5": 740,
    "6": 600,
    "7": 680,
    "8": 1400
}

# Definisi kapasitas maksimum truk dan jarak
kapasitas_maksimum_truk = 6000
jarak_maksimum = 30

def hitung_jarak(solusi):
    total_jarak = 0
    for i in range(len(solusi) - 1):
        titik_awal = solusi[i]
        titik_tujuan = solusi[i + 1]
        total_jarak += matriks_jarak[titik_awal][titik_tujuan]
    return total_jarak

def generate_tetangga(solusi):
    tetangga = solusi.copy()
    # Pastikan semua titik ada dalam solusi tetangga
    for node in matriks_jarak.keys():
        if node not in tetangga:
            idx = random.randint(1, len(tetangga) - 2)  # Hindari menghapus titik awal, x,
dan akhir
            tetangga.insert(idx, node)

    # Jika solusi tetangga melebihi kapasitas, kurangi titik sampai sesuai
    while hitung_volume_sampah(tetangga) > kapasitas_maksimum_truk and
hitung_jarak(tetangga) > jarak_maksimum:
        idx = random.randint(1, len(tetangga) - 2)  # Hindari menghapus titik awal atau
akhir
        del tetangga[idx]

    # Menghitung total jarak baru
    total_jarak = hitung_jarak(tetangga)
```

```
    return tetangga, total_jarak

def hitung_volume_sampah(solusi):
    total_volume = sum(volume_sampah[sampah] for sampah in solusi if sampah in
volume_sampah)
    return total_volume

def simulated_annealing(solusi_awal, matriks_jarak, volume_sampah,
kapasitas_maksimum_truk, suhu_awal, laju_penurunan):
    solusi_sekarang = solusi_awal
    jarak_sekarang = hitung_jarak(solusi_sekarang)
    volume_sekarang = hitung_volume_sampah(solusi_sekarang)

    suhu = suhu_awal
    iterasi = 1
    tabel = [["Iterasi", "Suhu", "Solusi Sekarang", "Jarak Sekarang", "Volume
Sekarang"]]

    # Menambahkan solusi awal ke dalam tabel sebelum memulai iterasi
    tabel.append([iterasi, f"{suhu:.2f}", solusi_sekarang, f"{jarak_sekarang:.2f}",
f"{volume_sekarang:.2f}"])

    while suhu > 0.01:  # Kondisi terminasi
        for iterasi_dalam in range(5):  # Iterasi dalam dengan 10 iterasi
            tetangga, jarak_tetangga = generate_tetangga(solusi_sekarang)
            volume_tetangga = hitung_volume_sampah(tetangga)
            delta_f = jarak_tetangga - jarak_sekarang

            # Pastikan titik awal dan akhir adalah '0'
            if tetangga[0] == '0' and tetangga[-1] == '0' and tetangga[-2] == 'x':
                if volume_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <=
jarak_maksimum and delta_f <= 0:
                    solusi_sekarang = tetangga
                    jarak_sekarang = jarak_tetangga
                    volume_sekarang = volume_tetangga
```

```
            elif volume_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <=
jarak_maksimum and random.random() < math.exp(-(delta_f) / suhu):
                solusi_sekarang = tetangga
                jarak_sekarang = jarak_tetangga
                volume_sekarang = volume_tetangga

            elif volume_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <=
jarak_maksimum:
                solusi_sekarang = solusi_awal
                jarak_sekarang = hitung_jarak(solusi_sekarang)
                volume_sekarang = hitung_volume_sampah(solusi_sekarang)
        iterasi += 1  # Increment counter iterasi

        # Tambahkan data ke dalam tabel
        tabel.append([iterasi, f"{suhu:.2f}", solusi_sekarang, f"{jarak_sekarang:.2f}",
f"{volume_sekarang:.2f}"])

      suhu *= laju_penurunan # Update suhu
   return tabel, solusi_sekarang, jarak_sekarang, volume_sekarang

# Penggunaan contoh
suhu_awal = 100
laju_penurunan = 0.98
solusi_awal1= ["0", "1", "2", "4", "3", "x", "0"]
solusi_awal2 = ["0", "1", "5", "2", "3", "x", "0"]
solusi_awal3 = ["0", "1", "8", "7", "6", "3", "x", "0"]
original_seed = random.getstate()
random.seed(1764)
tabel1, solusi_sekarang1, jarak_sekarang1, volume_sekarang1 =
simulated_annealing(solusi_awal1, matriks_jarak, volume_sampah,
kapasitas_maksimum_truk, suhu_awal, laju_penurunan)
solusi_terbaik1 = min(tabel1[1:], key=lambda x: (float(x[3]),  -float(x[4])-100))
random.setstate(original_seed)
print("Solusi optimal Rute 1:")
for info_iter in tabel1[1:]:
   if info_iter[2] == solusi_terbaik1[2]:
      iterasi = info_iter[0]
```

```
        suhu = float(info_iter[1])
        solusi = info_iter[2]
        jarak = float(info_iter[3])
        print(f"Solusi optimal 1 terdapat pada iterasi {iterasi}, suhu {suhu}:")
        print(f"Rute: {solusi},  Jarak: {jarak} km")
        # Menampilkan titik yang belum dikunjungi
        titik_belum_dikunjungi1 = [node for node in matriks_jarak.keys() if node not in
solusi]
        print("Titik yang belum dikunjungi:", titik_belum_dikunjungi1)
        break
print()

original_seed = random.getstate()
random.seed(1612)
tabel2, solusi_sekarang2, jarak_sekarang2, volume_sekarang2 =
simulated_annealing(solusi_awal2, matriks_jarak, volume_sampah,
kapasitas_maksimum_truk, suhu_awal, laju_penurunan)
# Ambil solusi terbaik dari setiap iterasi untuk rute 2 yang mencakup titik-titik yang
belum dikunjungi oleh rute 1 dan titik-titik yang telah dikunjungi oleh rute 1
tabel2_difilter = [info for info in tabel2[1:] if set(info[2]) >= set(titik_belum_dikunjungi1)]

# Filter solusi yang memenuhi batasan kapasitas truk dan batasan jarak maksimum
yang dapat ditempuh
solusi_fisibel2 = []
for info_iter in tabel2_difilter:
    solusi = info_iter[2]
    if (hitung_volume_sampah(solusi) <= kapasitas_maksimum_truk) and
(hitung_jarak(solusi) <= jarak_maksimum):
        solusi_fisibel2.append(info_iter)

# Ambil solusi terbaik yang memenuhi semua syarat
solusi_terbaik2 = min(solusi_fisibel2, key=lambda x: (float(x[3]), -float(x[4])))
random.setstate(original_seed)
print("Solusi optimal Rute 2:")
for info_iter in solusi_fisibel2:
    if info_iter == solusi_terbaik2:
        iterasi = info_iter[0]
```

```
        suhu = float(info_iter[1])
        solusi = info_iter[2]
        jarak = float(info_iter[3])
        print(f"Solusi optimal 2 terdapat pada iterasi {iterasi}, suhu {suhu}:")
        print(f"Rute: {solusi},  Jarak: {jarak} km")
        titik_belum_dikunjungi2 = [node for node in matriks_jarak.keys() if node not in
solusi]
        print("Titik yang belum dikunjungi:", titik_belum_dikunjungi2)

        break
print()
```

Lampiran 3. Kode Python Algoritma Simulated Annealing pada Contoh Studi Kasus

```
import random
import math
import pandas as pd
from tabulate import tabulate
random.seed(9)

# Definisi matriks jarak
matriks_jarak = {
    "0": {"1": 3.76, "2": 2.46, "3": 2.97, "4": 5.99, "5": 3.18},
    "1": {"0": 3.76, "2": 3.26, "3": 2.4, "4": 6.38, "5": 4.57},
    "2": {"0": 2.46, "1": 1.26, "3": 0.13, "4": 6.19, "5": 3.46},
    "3": {"0": 2.97, "1": 0.5, "2": 0.1, "4": 6.45, "5": 3.17},
    "4": {"0": 5.99, "1": 4.38, "2": 3.19, "3": 6.45, "5": 5.45},
    "5": {"0": 3.18, "1": 3.57, "2": 2.46, "3": 5.17, "4": 5.45},
}
jumlah_order ={
    "1": 20,
    "2": 25,
    "3": 29,
    "4": 20,
    "5": 5
}

# Definisi kapasitas maksimum truk dan jarak
kapasitas_maksimum_truk = 60
jarak_maksimum = 17.3

def hitung_jarak(solusi):
    total_jarak = 0
    for i in range(len(solusi) - 1):
        titik_awal = solusi[i]
        titik_tujuan = solusi[i + 1]
        total_jarak += matriks_jarak[titik_awal][titik_tujuan]
    return total_jarak
def generate_tetangga(solusi):
```

```
    tetangga = solusi.copy()

    # Pastikan semua titik ada dalam solusi tetangga
    for node in matriks_jarak.keys():
        if node not in tetangga:
            idx = random.randint(1, len(tetangga) - 1)
            tetangga.insert(idx, node)

    # Jika solusi tetangga melebihi kapasitas, kurangi titik sampai sesuai
    while hitung_jumlah_order(tetangga) > kapasitas_maksimum_truk and
hitung_jarak(tetangga) > jarak_maksimum:
        idx = random.randint(1, len(tetangga) - 1)  # Hindari menghapus titik awal, akhir
        del tetangga[idx]

    # Menghitung total jarak baru
    total_jarak = hitung_jarak(tetangga)

    return tetangga, total_jarak

def hitung_jumlah_order(solusi):
    total_order = sum(jumlah_order[order] for order in solusi if order in jumlah_order)
    return total_order

def simulated_annealing(solusi_awal, matriks_jarak, jumlah_order,
kapasitas_maksimum_truk, suhu_awal, laju_penurunan):
    solusi_sekarang = solusi_awal
    jarak_sekarang = hitung_jarak(solusi_sekarang)
    jumlah_order_sekarang = hitung_jumlah_order(solusi_sekarang)

    suhu = suhu_awal
    iterasi = 1
    tabel = [["Iterasi", "Suhu", "Solusi Sekarang", "Jarak Sekarang", "Jumlah Order
Sekarang"]]

    # Menambahkan solusi awal ke dalam tabel sebelum memulai iterasi
    tabel.append([iterasi, f"{suhu:.2f}", solusi_sekarang, f"{jarak_sekarang:.2f}",
f"{jumlah_order_sekarang:.2f}"])
```

```
    while suhu > 0.01:  # Kondisi terminasi
        for iterasi_dalam in range(2):  # Iterasi dalam dengan 10 iterasi
            tetangga, jarak_tetangga = generate_tetangga(solusi_sekarang)
            jumlah_tetangga = hitung_jumlah_order(tetangga)
            delta_f = jarak_tetangga - jarak_sekarang

            # Pastikan titik awal dan akhir adalah '0'
            if tetangga[0] == '0' and tetangga[-1] == '0' :
                if jumlah_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <=
jarak_maksimum and delta_f <= 0:
                    solusi_sekarang = tetangga
                    jarak_sekarang = jarak_tetangga
                    jumlah_order_sekarang = jumlah_tetangga

                elif jumlah_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <=
jarak_maksimum and random.random() < math.exp(-(delta_f) / suhu):
                    solusi_sekarang = tetangga
                    jarak_sekarang = jarak_tetangga
                    jumlah_order_sekarang = jumlah_tetangga


                elif jumlah_tetangga <= kapasitas_maksimum_truk and jarak_tetangga <=
jarak_maksimum:
                    solusi_sekarang = solusi_awal
                    jarak_sekarang = hitung_jarak(solusi_sekarang)
                    jumlah_order_sekarang = hitung_jumlah_order(solusi_sekarang)

            iterasi += 1  # Increment counter iterasi

            # Tambahkan data ke dalam tabel
            tabel.append([iterasi, f"{suhu:.2f}", solusi_sekarang, f"{jarak_sekarang:.2f}",
f"{jumlah_order_sekarang:.2f}"])

        suhu *= laju_penurunan # Update suhu

    return tabel, solusi_sekarang, jarak_sekarang,jumlah_order_sekarang
```

```
# Penggunaan contoh
suhu_awal = 100
laju_penurunan = 0.9


solusi_awal1= ["0", "1", "2","5", "0"]
solusi_awal2 = ["0", "3", "4",  "0"]

tabel1, solusi_sekarang1, jarak_sekarang1, jumlah_order_sekarang1 =
simulated_annealing(solusi_awal1, matriks_jarak, jumlah_order,
kapasitas_maksimum_truk, suhu_awal, laju_penurunan)
tabel2, solusi_sekarang2, jarak_sekarang2, jumlah_order_sekarang2 =
simulated_annealing(solusi_awal2, matriks_jarak, jumlah_order,
kapasitas_maksimum_truk, suhu_awal, laju_penurunan)


solusi_terbaik1 = min(tabel1[1:], key=lambda x:  -float(x[4])-100)
print("Solusi optimal Rute 1:")
for info_iter in tabel1[1:]:
    if info_iter[2] == solusi_terbaik1[2]:
        iterasi = info_iter[0]
        suhu = float(info_iter[1])
        solusi = info_iter[2]
        jarak = float(info_iter[3])
        order = float(info_iter[4])
        print(f"Solusi optimal 1 terdapat pada iterasi {iterasi}, suhu {suhu}:")
        print(f"Rute: {solusi}, Volume: {order}, Jarak: {jarak} km")
        # Menampilkan titik yang belum dikunjungi
        titik_belum_dikunjungi1 = [node for node in matriks_jarak.keys() if node not in
solusi]
        print("Titik yang belum dikunjungi:", titik_belum_dikunjungi1)
        break

print()
```

```
# Ambil solusi terbaik dari setiap iterasi untuk rute 2 yang mencakup titik-titik yang
belum dikunjungi oleh rute 1 dan titik-titik yang telah dikunjungi oleh rute 1
tabel2_difilter = [info for info in tabel2[1:] if set(info[2]) >= set(titik_belum_dikunjungi1)]

# Filter solusi yang memenuhi batasan kapasitas truk dan batasan jarak maksimum
yang dapat ditempuh
solusi_fisibel2 = []
for info_iter in tabel2_difilter:
    solusi = info_iter[2]
    if (hitung_jumlah_order(solusi) <= kapasitas_maksimum_truk) and
(hitung_jarak(solusi) <= jarak_maksimum):
        solusi_fisibel2.append(info_iter)

# Ambil solusi terbaik yang memenuhi semua syarat
solusi_terbaik2 = min(solusi_fisibel2, key=lambda x: (float(x[3]), -float(x[4])))
print("Solusi optimal Rute 2:")
for info_iter in solusi_fisibel2:
    if info_iter == solusi_terbaik2:
        iterasi = info_iter[0]
        suhu = float(info_iter[1])
        solusi = info_iter[2]
        jarak = float(info_iter[3])
        order = float(info_iter[4])
        print(f"Solusi optimal 2 terdapat pada iterasi {iterasi}, suhu {suhu}:")
        print(f"Rute: {solusi}, Volume: {order}, Jarak: {jarak} km")
        titik_belum_dikunjungi2 = [node for node in matriks_jarak.keys() if node not in
solusi]
        print("Titik yang belum dikunjungi:", titik_belum_dikunjungi2)

        break
print()
print()
print("Tabel Rute 1: ")
print(tabulate(tabel1, headers="firstrow", tablefmt="grid"))
print()
print("Tabel Rute 2: ")
print(tabulate(tabel2, headers="firstrow", tablefmt="grid"))
```

```
print()

# Konversi tabel ke dalam struktur DataFrame
df_rute1 = pd.DataFrame(tabel1[1:], columns=tabel1[0])
df_rute2 = pd.DataFrame(tabel2[1:], columns=tabel2[0])
# Modifikasi kolom 'Solusi Sekarang' untuk menampilkan rute dengan tanda strip (-)
df_rute1['Solusi Sekarang'] = df_rute1['Solusi Sekarang'].apply(lambda x: '-
'.join(str(elem) for elem in x))
df_rute2['Solusi Sekarang'] = df_rute2['Solusi Sekarang'].apply(lambda x: '-
'.join(str(elem) for elem in x))

# Simpan DataFrame ke dalam file Excel
with pd.ExcelWriter('output_tabel.xlsx') as writer:
    df_rute1.to_excel(writer, sheet_name='Rute_1', index=False)
    df_rute2.to_excel(writer, sheet_name='Rute_2', index=False)
```