PP/018/V/R2

LABORATORIUM
INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS AHMAD DAHLAN

PETUNJUK PRAKTIKUM

EDISI KURIKULUM OBE

TEKNIK OPTIMASI

Penyusun:
Ardiansyah, M.Cs

2022

HAK CIPTA

PETUNJUK PRAKTIKUM TEKNIK OPTIMASI

Diterbitkan oleh:
Program Studi Informatika
Fakultas Teknologi Industri
Universitas Ahmad Dahlan
Jalan Ring Road Selatan, Tamanan, Banguntapan, Bantul Yogyakarta 55166

Penulis	: Ardiansyah
Editor	: Laboratorium Informatika, Universitas Ahmad Dahlan
Desain sampul	: Laboratorium Informatika, Universitas Ahmad Dahlan
Tata letak	: Laboratorium Informatika, Universitas Ahmad Dahlan
Ukuran/Halaman	: 21 x 29,7 cm / 70 halaman

Didistribusikan oleh:

Laboratorium Informatika
Universitas Ahmad Dahlan
Jalan Ring Road Selatan, Tamanan, Banguntapan, Bantul Yogyakarta 55166
Indonesia

KATA PENGANTAR

Mempelajari algoritme optimasi berbasis metaheuristik tidak cukup hanya sekadar teori dan simulasi menggunakan *spreadsheet*. Perlu dilakukan pembelajaran lanjutan dengan mempraktikkannya secara langsung menggunakan bahasa pemrograman. Praktik seperti ini akan semakin memperdalam pemahaman mahasiswa terhadap teori dan penerapan berbagai algoritme yang telah dipelajari di kelas.

Petunjuk praktikum ini hadir sebagai satu materi kuliah yang menyatu dengan perkuliahan di kelas (teori). Oleh karena itu agar mudah menjalani praktikum ini, mahasiswa harus sering-sering membuka kembali materi teori yang telah diberikan di kelas. Begitu pula sebaliknya, agar memahari teori, mahasiswa perlu mempraktikkannya dengan membuat program optimasi menggunakan bahasa pemrograman.

Semoga dengan hadirnya petunjuk praktikum ini semakin meningkatkan pemahaman dan keterampilan mahasiswa terhadap algoritme optimasi berbasis metaheuristik

Lab. Relata UAD, 5 September 2022

Ardiansyah, M.Cs

DAFTAR PENYUSUN

Ardiansyah, M.Cs

	Ardiansyah adalah seorang dosen dan peneliti di Program Studi Informatika UAD sejak tahun 2004. Ia menempuh studi S1 Teknik Informatika UAD (1998-2003), S2 Ilmu Komputer UGM (2009-2010), dan sejak 2019 menempuh S3 di Departemen Teknik Elektro dan Teknologi Informasi (DTETI) UGM. Ardiansyah memiliki keahlian di bidang *software engineering* meliputi *software requirements*, *software effort estimation*, dan *software testing*. Bidang penelitian yang ditekuninya adalah fokus mengenai *search based software engineering* (SBSE), yaitu penerapan algoritme optimasi metaheuristik untuk pemecahan masalah-masalah *software engineering* seperti *next release problem* (NRP), *effort estimation*, *test case prioritization* (TCP), dan lain sebagainya. Web: http://ardiansyah.tif.uad.ac.id Email: ardiansyah@tif.uad.ac.id

HALAMAN REVISI

Yang bertanda tangan di bawah ini:

Nama : Ardiansyah, M.Cs

NIP/NIY : 60030476

Jabatan : Teknik Optimasi

Dengan ini menyatakan pelaksanaan Revisi Petunjuk Praktikum **Teknik Optimasi** untuk Program Studi Informatika telah dilaksanakan dengan penjelasan sebagai berikut:

No	Keterangan Revisi	Tanggal Revisi	Nomor Modul
1	Mengubah seluruh isi modul sebelumnya	5 September 2022	PP/018/V/R1

Yogyakarta, 5 September 2022

Penyusun

Ardiansyah, M.Cs

NIY. 60030476

HALAMAN PERNYATAAN

Yang bertanda tangan di bawah ini:

Nama : Lisna Zahrotun, S.T., M.Cs.

NIK/NIY : 60150773

Jabatan : Kepala Laboratorium Informatika

Menerangkan dengan sesungguhnya bahwa Petunjuk Praktikum ini telah direview dan akan digunakan untuk pelaksanaan praktikum di Semester Gasal Tahun Akademik 2020/2021 di Laboratorium Praktikum Informatika, Program Studi Informatika, Fakultas Teknologi Industri, Universitas Ahmad Dahlan.

Yogyakarta, 5 September 2022

Mengetahui, Ketua Kelompok Keilmuan Rekayasa Perangkat Lunak dan Data (RELATA)	Kepala Laboratorium Informatika
Guntur Maulana Zamroni, B.Sc., M.Kom.	**Lisna Zahrotun, S.T., M.Cs.**
NIY. 60181172	NIY. 60150773

VISI DAN MISI PRODI INFORMATIKA

VISI

Menjadi Program Studi Informatika yang diakui secara internasional dan unggul dalam bidang Informatika serta berbasis nilai-nilai Islam.

MISI

1. Menjalankan pendidikan sesuai dengan kompetensi bidang Informatika yang diakui nasional dan internasional
2. Meningkatkan penelitian dosen dan mahasiswa dalam bidang Informatika yang kreatif, inovatif dan tepat guna.
3. Meningkatkan kuantitas dan kualitas publikasi ilmiah tingkat nasional dan internasional
4. Melaksanakan dan meningkatkan kegiatan pengabdian masyarakat oleh dosen dan mahasiswa dalam bidang Informatika.
5. Menyelenggarakan aktivitas yang mendukung pengembangan program studi dengan melibatkan dosen dan mahasiswa.
6. Menyelenggarakan kerja sama dengan lembaga tingkat nasional dan internasional.
7. Menciptakan kehidupan Islami di lingkungan program studi.

TATA TERTIB LABORATORIUM INFORMATIKA

DOSEN/KOORDINATOR PRAKTIKUM

1. Dosen harus hadir saat praktikum minimal 15 menit di awal kegiatan praktikum untuk mengisi materi dan menandatangani presensi kehadiran praktikum.
2. Dosen membuat modul praktikum, soal seleksi asisten, pre-test, post-test, dan responsi dengan berkoordinasi dengan asisten dan pengampu mata praktikum.
3. Dosen berkoordinasi dengan koordinator asisten praktikum untuk evaluasi praktikum setiap minggu.
4. Dosen menandatangani surat kontrak asisten praktikum dan koordinator asisten praktikum.
5. Dosen yang tidak hadir pada slot praktikum tertentu tanpa pemberitahuan selama 2 minggu berturut-turut mendapat teguran dari Kepala Laboratorium, apabila masih berlanjut 2 minggu berikutnya maka Kepala Laboratorium berhak mengganti koordinator praktikum pada slot tersebut.

PRAKTIKAN

1. Praktikan harus hadir 15 menit sebelum kegiatan praktikum dimulai, dan dispensasi terlambat 15 menit dengan alasan yang jelas (kecuali asisten menentukan lain dan patokan jam adalah jam yang ada di Laboratorium, terlambat lebih dari 15 menit tidak boleh masuk praktikum & dianggap Inhal).
2. Praktikan yang tidak mengikuti praktikum dengan alasan apapun, wajib mengikuti INHAL, maksimal 4 kali praktikum dan jika lebih dari 4 kali maka praktikum dianggap GAGAL.
3. Praktikan harus berpakaian rapi sesuai dengan ketentuan Universitas, sebagai berikut:
   a. Tidak boleh memakai Kaos Oblong, termasuk bila ditutupi Jaket/Jas Almamater (Laki-laki / Perempuan) dan Topi harus Dilepas.
   b. Tidak Boleh memakai Baju ketat, Jilbab Minim dan rambut harus tertutup jilbab secara sempurna, tidak boleh kelihatan di jidat maupun di punggung (khusus Perempuan).
   c. Tidak boleh memakai baju minim, saat duduk pun pinggang harus tertutup rapat (Laki-laki / Perempuan).
   d. Laki-laki tidak boleh memakai gelang, anting-anting ataupun aksesoris Perempuan.
4. Praktikan tidak boleh makan dan minum selama kegiatan praktikum berlangsung, harus menjaga kebersihan, keamanan dan ketertiban selama mengikuti kegiatan praktikum atau selama berada di dalam laboratorium (tidak boleh membuang sampah sembarangan baik kertas, potongan kertas, bungkus permen baik di lantai karpet maupun di dalam ruang CPU).
5. Praktikan dilarang meninggalkan kegiatan praktikum tanpa seizin Asisten atau Laboran.
6. Praktikan harus meletakkan sepatu dan tas pada rak/loker yang telah disediakan.
7. Selama praktikum dilarang NGENET/NGE-GAME, kecuali mata praktikum yang membutuhkan atau menggunakan fasilitas Internet.
8. Praktikan dilarang melepas kabel jaringan atau kabel power praktikum tanpa sepengetahuan laboran
9. Praktikan harus memiliki FILE Petunjuk praktikum dan digunakan pada saat praktikum dan harus siap sebelum praktikum berlangsung.
10. Praktikan dilarang melakukan kecurangan seperti mencontek atau menyalin pekerjaan praktikan yang lain saat praktikum berlangsung atau post-test yang menjadi tugas praktikum.
11. Praktikan dilarang mengubah setting software/hardware komputer baik menambah atau mengurangi tanpa permintaan asisten atau laboran dan melakukan sesuatu yang dapat merugikan laboratorium atau praktikum lain.

12. Asisten, Koordinator Praktikum, Kepala laboratorium dan Laboran mempunyai hak untuk menegur, memperingatkan bahkan meminta praktikan keluar ruang praktikum apabila dirasa anda mengganggu praktikan lain atau tidak melaksanakan kegiatan praktikum sebagaimana mestinya dan atau tidak mematuhi aturan lab yang berlaku.
13. Pelanggaran terhadap salah satu atau lebih dari aturan diatas maka Nilai praktikum pada pertemuan tersebut dianggap 0 (NOL) dengan status INHAL.

ASISTEN PRAKTIKUM

1. Asisten harus hadir 15 Menit sebelum praktikum dimulai (konfirmasi ke koordinator bila mengalami keterlambatan atau berhalangan hadir).
2. Asisten yang tidak bisa hadir WAJIB mencari pengganti, dan melaporkan kepada Koordinator Asisten.
3. Asisten harus berpakaian rapi sesuai dengan ketentuan Universitas, sebagai berikut:
   a. Tidak boleh memakai Kaos Oblong, termasuk bila ditutupi Jaket/Jas Almamater (Laki-laki / Perempuan) dan Topi harus Dilepas.
   b. Tidak Boleh memakai Baju ketat, Jilbab Minim dan rambut harus tertutup jilbab secara sempurna, tidak boleh kelihatan di jidat maupun di punggung (khusus Perempuan).
   c. Tidak boleh memakai baju minim, saat duduk pun pinggang harus tertutup rapat (Laki-laki / Perempuan).
   d. Laki-laki tidak boleh memakai gelang, anting-anting ataupun aksesoris Perempuan.
4. Asisten harus menjaga kebersihan, keamanan dan ketertiban selama mengikuti kegiatan praktikum atau selama berada di laboratorium, menegur atau mengingatkan jika ada praktikan yang tidak dapat menjaga kebersihan, ketertiban atau kesopanan.
5. Asisten harus dapat merapikan dan mengamankan presensi praktikum, Kartu Nilai serta tertib dalam memasukan/Input nilai secara Online/Offline.
6. Asisten harus dapat bertindak secara profesional sebagai seorang asisten praktikum dan dapat menjadi teladan bagi praktikan.
7. Asisten harus dapat memberikan penjelasan/pemahaman yang dibutuhkan oleh praktikan berkenaan dengan materi praktikum yang diasisteni sehingga praktikan dapat melaksanakan dan mengerjakan tugas praktikum dengan baik dan jelas.
8. Asisten tidak diperkenankan mengobrol sendiri apalagi sampai membuat gaduh.
9. Asisten dimohon mengkoordinasikan untuk meminta praktikan agar mematikan komputer untuk jadwal terakhir dan sudah dilakukan penilaian terhadap hasil kerja praktikan.
10. Asisten wajib untuk mematikan LCD Projector dan komputer asisten/praktikan apabila tidak digunakan.
11. Asisten tidak diperkenankan menggunakan akses internet selain untuk kegiatan praktikum, seperti Youtube/Game/Medsos/Streaming Film di komputer praktikan.

LAIN-LAIN

1. Pada Saat Responsi Harus menggunakan Baju Kemeja untuk Laki-laki dan Perempuan untuk Praktikan dan Asisten.
2. Ketidakhadiran praktikum dengan alasan apapun dianggap INHAL.
3. Izin praktikum mengikuti aturan izin SIMERU/KULIAH.
4. Yang tidak berkepentingan dengan praktikum dilarang mengganggu praktikan atau membuat keributan/kegaduhan.
5. Penggunaan lab diluar jam praktikum maksimal sampai pukul 21.00 dengan menunjukkan surat ijin dari Kepala Laboratorium Prodi Informatika.

Yogyakarta, 5 September 2022

Kepala Laboratorium Informatika

Lisna Zahrotun, S.T., M.Cs.
NIY. 60150773

DAFTAR ISI

HAK CIPTA 1
KATA PENGANTAR 2
DAFTAR PENYUSUN 3
HALAMAN REVISI 4
HALAMAN PERNYATAAN 5
VISI DAN MISI PRODI INFORMATIKA 6
TATA TERTIB LABORATORIUM INFORMATIKA 7
DAFTAR ISI 10
DAFTAR GAMBAR 11
DAFTAR TABEL 12
SKENARIO PRAKTIKUM SECARA DARING 13
PRAKTIKUM 1: REPRESENTASI SOLUSI (*ENCODING*) 14
PRAKTIKUM 2: NEIGHBORHOOD 19
PRAKTIKUM 3: SIMULATED ANNEALING 24
PRAKTIKUM 4: ANT COLONY OPTIMIZATION (ACO) 30
DAFTAR PUSTAKA 52

DAFTAR GAMBAR

Gambar 1.1 Label Gambar. 15

DAFTAR TABEL

SKENARIO PRAKTIKUM SECARA DARING

Nama Mata Praktikum :
Jumlah Pertemuan :

TABEL SKENARIO PRAKTIKUM DARING

Pertemuan ke	Judul Materi	Waktu (Lama praktikum sampai pengumpulan posttest)	Skenario Praktikum dari pemberian pre-test, post-test dan pengumpulannya serta mencantumkan metode yang digunakan misal video, whatsapp group, Google meet atau lainnya
1			
2			
3	dst	dst	Dst

PRAKTIKUM 1: REPRESENTASI SOLUSI (*ENCODING*)

Pertemuan ke : 1

Total Alokasi Waktu : 90 menit
- Materi : 15 menit
- Pre-Test : 15 menit
- Praktikum : 45 menit
- Post-Test : 15 menit

Total Bobot Penilaian : 100%
- Pre-Test : 20 %
- Praktik : 30 %
- Post-Test : 50 %

Pemenuhan CPL, CPMK dan Sub-CPMK:

CPL-03	Mampu menerapkan konsep teoritis bidang area Informatika terkait matematika dasar dan ilmu komputer untuk memodelkan masalah dan meningkatkan produktivitas
CPMK-01	Mampu menerapkan konsep teoritis bidang area Informatika terkait matematika dasar dan ilmu komputer untuk memodelkan masalah dan meningkatkan produktivitas
CPMK-02	Mampu memilih, membuat dan menerapakan teknik, sumber daya, penggunaan perangkat teknik modern dan implementasi berbagai teknik optimasi untuk memecahkan masalah
Sub-CPMK-02	Mampu menjelaskan dan membuat representasi (encoding) solusi suatu permasalahan optimasi

1.1. DESKRIPSI CAPAIAN PEMBELAJARAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu membuat representasi solusi biner, diskrit, real dan permutasi sesuai dengan permasalahan optimasi

1.2. INDIKATOR KETERCAPAIAN PEMBELAJARAN

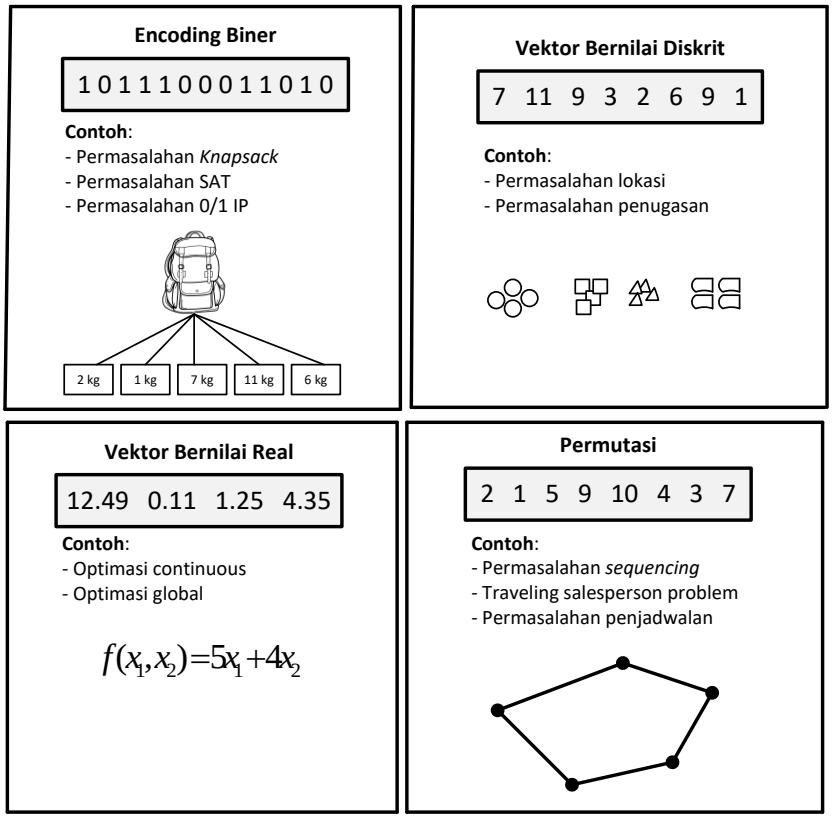
Indikator ketercapaian diukur dengan:

CPL-03	CPMK-02	Sub-CPMK-02	Kesesuaian hasil pembuatan representasi solusi menggunakan bahasa pemrograman

1.3. TEORI PENDUKUNG

Setiap solusi yang dicari secara iteratif memerlukan suatu presentasi yang disebut *encoding*. Encoding berperan penting karena memengaruhi efisiensi dan efektivitas optimasi metaheuristik. Encoing haruslah cocok dan relevan dengan permasalahan optimasi yang hendak dipecahkan. Gambar *1.1* berikut menunjukkan representasi solusi biner, diskrit, real dan permutasi.

Berdasarkan gambar bisa kita simpulkan bahwa encoding biner, diskrit dan permutasi menggunakan representasi bilangan integer. Sedangkan encoding *real* menggunakan bilangan *real*.



Gambar 1.1 Representasi atau encoding solusi biner, diskrit, real dan permutasi

1.4. HARDWARE DAN SOFTWARE

Hardware dan software yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Python, C/C++, PHP atau C#, dan lain sebagainya yang sudah dikuasai sebelumnya

1.5. PRE-TEST

Jawablah pertanyaan berikut (**Total Skor: 100):**

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1.	02	02	Buatlah program yang menjalankan 5 iterasi. Kemudian simpanlah tiap indeks iterasi ke dalam larik satu dimensi	20
2.			Ulangi pertanyaan ke-1 sebanyak 2 kali dan simpan setiap larik yang dihasilkan ke dalam larik. Sehingga hasil akhirnya membentuk larik multi-dimensi.	80

1.6. LANGKAH PRAKTIKUM

Aturan Penilaian (Total Skor: 100):

No	CPMK	Sub-CPMK	Pertanyaan	Dokumen Pendukung	Skor
1.	02	02	Selesaikan langkah praktikum 1	Hasil praktikum langkah 1	20
2.			Selesaikan langkah praktikum 2	Hasil praktikum langkah 2	30
3.			Selesaikan langkah praktikum 3	Hasil praktikum langkah 3	20
4.			Selesaikan langkah praktikum 4	Hasil praktikum langkah 4	30

Langkah-Langkah Praktikum:

1. Representasi Biner

Untuk membuat representasi biner, cukup tentukan panjang atau ukuran solusi (baris 9) kemudian simpan nilai acak yang dihasilkan di setiap iterasi ke dalam larik `ret` (baris 6).

```
1.  import random
2.
3.  def representasiBiner(jumlahBit):
4.      ret = []
5.      for _ in range(jumlahBit):
6.          ret.append(random.randint(0, 1))
7.      return ret
8.
9.  jumlahBit = 8
10. solusi = representasiBiner(jumlahBit)
11. print(solusi)
```

Output:

```
[1, 0, 1, 1, 0, 0, 0, 1]
```

2. Representasi Real

Untuk membuat representasi *real*, maka deklarasikan sebuah variabel larik multi-dimensi yang tiap elemennya berisi larik satu dimensi yang indeks-0 adalah batas bawah (*lower bound*) dan indeks-1 adalah batas atas (*upper bound*) (baris 9-13). Kemudian lakukan iterasi sebanyak jumlah elemen `variables` (baris 5) yang selanjutnya membangkitkan nilai acak antara indeks-0 dan indeks-1, lalu disimpan ke larik `ret` (baris 6).

```
1.  import random
2.
3.  def representasiReal(variables):
4.      ret = []
5.      for i in range(len(variables)):
6.          ret.append(random.uniform(variables[i][0], variables[i][1]))
7.      return ret
8.
9.  variables = [
10.     [5, 7.49],
11.     [7.5, 12.49],
12.     [12.5, 15]
13. ]
14. solusi = representasiReal(variables)
15. print(solusi)
```

Output:

```
[6.787611227399061, 7.839212519086781, 12.55589097767962]
```

3. Permutasi

Membuat representasi permutasi di Python sangat sederhana dan mudah. Cukup panggil *library* `permutations` dari `itertools` (baris 2) lalu buat himpunan permutasi mulai dari integer 1 hingga 4-1 (baris 3) yang akan menghasilkan kombinasi tiga digit angka. Selanjutnya bangkitkan nilai indeks secara acak untuk memanggil salah satu permutasi (baris 4).

```
1.  import random
2.  from itertools import permutations
3.  solusi = list(permutations(range(1,4)))
4.  indexOfRepresentation = random.randint(0, len(solusi)-1)
5.  print(solusi[indexOfRepresentation])
```

Output:

```
(3, 2, 1)
```

4. Diskrit

Representasi diskrit mirip dengan representasi real (langkah ke-2). Yang membedakan hanyalah nilai acak yang dibangkitkan adalah berupa integer.

```
1.  import random
2.
3.  def representasiDiskrit(variables):
4.      ret = []
5.      for i in range(len(variables)):
6.          ret.append(random.randint(variables[i][0], variables[i][1]))
7.      return ret
8.
9.  variables = [
```

```
10.     [1, 5],
11.     [1, 8],
12.     [1, 15],
13.     [1, 3]
14. ]
15. solusi = representasiDiskrit(variables)
16. print(solusi)
```

Output:

```
[5, 5, 4, 3]
```

1.7. POST TEST

Jawablah pertanyaan berikut (**Total Skor: 100):**

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1.	02	02	Apa arti `(0, 1)` pada langkah 1 baris ke-6?	20
2.			Pilihlah salah satu *search domain* (kolom paling kanan) dari daftar Test Function Optimization. Ubahlah rentang variabel desain representasi *real* sesuai rentang *search domain* yang dipilih tersebut. Kemudian eksekusilah programnya.	35
3.			Pada langkah 3 baris ke-3 gantilah `(1,4)` dengan `(1,9)`, kemudian eksekusilah. Jelaskanlah perbedaan hasil keduanya?	25
4.			Tambahlah 2 larik baru di variabel `variables` di baris 9 langkah 4. Tuliskanlah hasil eksekusinya.	20

1.8. HASIL CAPAIAN PRAKTIKUM

Diisi oleh asisten setelah semua assessment dinilai.

No	Bentuk Assessment	CPMK	Sub-CPMK	Bobot	Skor (0-100)	Nilai Akhir (Bobot x Skor)
1.	Pre-Test	02	02	20%		
2.	Praktik			30%		
3.	Post-Test			50%		
					Total Nilai	

LEMBAR JAWABAN PRE-TEST DAN POST-TEST PRAKTIKUM

Nama : **NIM :**	**Asisten:** **Paraf Asisten:**	**Tanggal:** **Nilai:**

PRAKTIKUM 2: NEIGHBORHOOD

Pertemuan ke : 1

Total Alokasi Waktu : 90 menit
- Materi : 15 menit
- Pre-Test : 15 menit
- Praktikum : 45 menit
- Post-Test : 15 menit

Total Bobot Penilaian : 100%
- Pre-Test : 20 %
- Praktik : 30 %
- Post-Test : 50 %

Pemenuhan CPL dan CPMK:

CPL-03	Mampu menerapkan konsep teoritis bidang area Informatika terkait matematika dasar dan ilmu komputer untuk memodelkan masalah dan meningkatkan produktivitas
CPMK-01	Mampu menerapkan konsep teoritis bidang area Informatika terkait matematika dasar dan ilmu komputer untuk memodelkan masalah dan meningkatkan produktivitas
Sub-CPMK-03	Mampu menjelaskan dan membuat neighborhood suatu solusi permasalahan optimasi

1.1. DESKRIPSI CAPAIAN PEMBELAJARAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu membuat *neighborhood* untuk solusi biner, diskrit, real dan permutasi

1.2. INDIKATOR KETERCAPAIAN PEMBELAJARAN

Indikator ketercapaian diukur dengan:

CPMK-01	Sub-CPMK-03	Kesesuaian hasil pembuatan *neighborhood* suatu solusi permasalahan optimasi menggunakan bahasa pemrograman

1.3. TEORI PENDUKUNG

Neighborhood adalah konsep penting yang menjadi salah satu langkah dalam pemecahan masalah optimasi. *Neighbor* merupakan himpunan kandidat solusi yang mirip atau dekat dengan solusi yang ada saat ini. Oleh karena itu struktur *neighborhood* memainkan peranan penting dalam metaheuristik. Jika strukturnya tidak memadai terhadap permasalahan yang dihadapi, maka metaheuristik akan gagal menemukan solusinya.

Satu solusi s' pada *neighborhood* $s(s' \in N(S))$ disebut sebagai satu *neighbor* s. Satu *neighbor* dibangkitkan oleh operator perpindahan m yang melakukan perturbasi kecil terhadap solusi s. Karakteristik khas yang wajib dimiliki oleh *neighborhood* adalah *locality*.

Locality merupakan efek yang diterima suatu solusi pada saat terjadinya perpindahan atau perturbasi pada representasi solusi. Jika representasi solusi mengalami perubahan kecil, maka solusi juga akan mengalami perubahan kecil. Kondisi ini disebut sebagai *strong locality*. *Strong locality* diperlukan agar metaheuristik bisa melakukan pencarian yang bermakna pada lanskap permasalahan. Sebaliknya, bisa terjadi representasi solusi yang mengalami perubahan kecil namun menyebabkan solusinya mengalami perubahan besar, ini yang disebut sebagai *weak locality*. Pada *weak locality* yang ekstrim, bisa menyebabkan pencarian konvergen ke pencarian acak di ruang pencarian.

1.4. HARDWARE DAN SOFTWARE

Hardware dan software yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Python, C/C++, PHP atau C#, dan lain sebagainya yang sudah dikuasai sebelumnya

1.5. PRE-TEST

Jawablah pertanyaan berikut (**Total Skor: 100):**

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1	01, 02	02	Lihat kembali Praktikum 1. Bangkitkanlah bilangan integer secara acak antara 1 hingga 10 dan cetaklah hasilnya.	40
1.	01, 02	02	Hitunglah jumlah kombinasi yang dihasilkan dari nilai integer (1, 5, 8). Tuliskan semua kombinasi yang dihasilkan tersebut!	60

1.6. LANGKAH PRAKTIKUM

Aturan Penilaian (Total Skor: 100):

No	CPMK	Sub-CPMK	Pertanyaan	Dokumen Pendukung	Skor
1.	01, 02	03	Selesaikan langkah praktikum 1	Hasil praktikum langkah 1	20
2.	01, 02	03	Selesaikan langkah praktikum 2	Hasil praktikum langkah 2	60
3.	01, 02	03	Selesaikan langkah praktikum 3	Hasil praktikum langkah 3	20

Langkah-Langkah Praktikum:

1. Neighborhood Biner

   Membuat *neighborhood* biner cukup menggunakan fungsi bawaan `list` dan `itertools` (baris 3). Larik `[0, 1]` merupakan rentang nilai integer dan `repeat=3` menunjukkan banyaknya digit pembentuk kombinasi suatu representasi solusi.

```
1.  import itertools
2.
3.  neighborhood = list(itertools.product([0, 1], repeat=3))
4.  print(neighborhood)
```

   Output

```
[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]
```

2. Neighborhood Diskrit

   Membuat *neighborhood* diskrit dilakukan dengan cara menentukan item-item tiap komponen (baris 3). Selanjutnya bangkitkan himpunan neighborhood menggunakan fungsi bawaan `list` dan `itertools.product` (baris 10).

```
1.  import itertools
2.
3.  components = [
4.      [1,2,3],
5.      [1,2,3,4,5],
6.      [1,2],
7.      [1,2,3,4,5,6,7,8]
8.  ]
9.
10. neighborhood = list(itertools.product(*components))
11. print(neighborhood)
```

Output

```
[(1, 1, 1, 1), (1, 1, 1, 2), (1, 1, 1, 3), (1, 1, 1, 4), (1, 1, 1, 5), (1, 1, 1, 6), (1, 1, 1, 7),
(1, 1, 1, 8), (1, 1, 2, 1), (1, 1, 2, 2), (1, 1, 2, 3), (1, 1, 2, 4), (1, 1, 2, 5), (1, 1, 2, 6),
(1, 1, 2, 7), (1, 1, 2, 8), (1, 2, 1, 1), (1, 2, 1, 2), (1, 2, 1, 3), (1, 2, 1, 4), (1, 2, 1, 5),
(1, 2, 1, 6), (1, 2, 1, 7), (1, 2, 1, 8), (1, 2, 2, 1), (1, 2, 2, 2), (1, 2, 2, 3), (1, 2, 2, 4),
(1, 2, 2, 5), (1, 2, 2, 6), (1, 2, 2, 7), (1, 2, 2, 8), (1, 3, 1, 1), (1, 3, 1, 2), (1, 3, 1, 3),
(1, 3, 1, 4), (1, 3, 1, 5), (1, 3, 1, 6), (1, 3, 1, 7), (1, 3, 1, 8), (1, 3, 2, 1), (1, 3, 2, 2),
(1, 3, 2, 3), (1, 3, 2, 4), (1, 3, 2, 5), (1, 3, 2, 6), (1, 3, 2, 7), (1, 3, 2, 8), (1, 4, 1, 1),
(1, 4, 1, 2), (1, 4, 1, 3), (1, 4, 1, 4), (1, 4, 1, 5), (1, 4, 1, 6), (1, 4, 1, 7), (1, 4, 1, 8),
(1, 4, 2, 1), (1, 4, 2, 2), (1, 4, 2, 3), (1, 4, 2, 4), (1, 4, 2, 5), (1, 4, 2, 6), (1, 4, 2, 7),
(1, 4, 2, 8), (1, 5, 1, 1), (1, 5, 1, 2), (1, 5, 1, 3), (1, 5, 1, 4), (1, 5, 1, 5), (1, 5, 1, 6),
(1, 5, 1, 7), (1, 5, 1, 8), (1, 5, 2, 1), (1, 5, 2, 2), (1, 5, 2, 3), (1, 5, 2, 4), (1, 5, 2, 5),
(1, 5, 2, 6), (1, 5, 2, 7), (1, 5, 2, 8), (2, 1, 1, 1), (2, 1, 1, 2), (2, 1, 1, 3), (2, 1, 1, 4),
(2, 1, 1, 5), (2, 1, 1, 6), (2, 1, 1, 7), (2, 1, 1, 8), (2, 1, 2, 1), (2, 1, 2, 2), (2, 1, 2, 3),
(2, 1, 2, 4), (2, 1, 2, 5), (2, 1, 2, 6), (2, 1, 2, 7), (2, 1, 2, 8), (2, 2, 1, 1), (2, 2, 1, 2),
(2, 2, 1, 3), (2, 2, 1, 4), (2, 2, 1, 5), (2, 2, 1, 6), (2, 2, 1, 7), (2, 2, 1, 8), (2, 2, 2, 1),
(2, 2, 2, 2), (2, 2, 2, 3), (2, 2, 2, 4), (2, 2, 2, 5), (2, 2, 2, 6), (2, 2, 2, 7), (2, 2, 2, 8),
(2, 3, 1, 1), (2, 3, 1, 2), (2, 3, 1, 3), (2, 3, 1, 4), (2, 3, 1, 5), (2, 3, 1, 6), (2, 3, 1, 7),
(2, 3, 1, 8), (2, 3, 2, 1), (2, 3, 2, 2), (2, 3, 2, 3), (2, 3, 2, 4), (2, 3, 2, 5), (2, 3, 2, 6),
(2, 3, 2, 7), (2, 3, 2, 8), (2, 4, 1, 1), (2, 4, 1, 2), (2, 4, 1, 3), (2, 4, 1, 4), (2, 4, 1, 5),
(2, 4, 1, 6), (2, 4, 1, 7), (2, 4, 1, 8), (2, 4, 2, 1), (2, 4, 2, 2), (2, 4, 2, 3), (2, 4, 2, 4),
(2, 4, 2, 5), (2, 4, 2, 6), (2, 4, 2, 7), (2, 4, 2, 8), (2, 5, 1, 1), (2, 5, 1, 2), (2, 5, 1, 3),
(2, 5, 1, 4), (2, 5, 1, 5), (2, 5, 1, 6), (2, 5, 1, 7), (2, 5, 1, 8), (2, 5, 2, 1), (2, 5, 2, 2),
(2, 5, 2, 3), (2, 5, 2, 4), (2, 5, 2, 5), (2, 5, 2, 6), (2, 5, 2, 7), (2, 5, 2, 8), (3, 1, 1, 1),
(3, 1, 1, 2), (3, 1, 1, 3), (3, 1, 1, 4), (3, 1, 1, 5), (3, 1, 1, 6), (3, 1, 1, 7), (3, 1, 1, 8),
(3, 1, 2, 1), (3, 1, 2, 2), (3, 1, 2, 3), (3, 1, 2, 4), (3, 1, 2, 5), (3, 1, 2, 6), (3, 1, 2, 7),
(3, 1, 2, 8), (3, 2, 1, 1), (3, 2, 1, 2), (3, 2, 1, 3), (3, 2, 1, 4), (3, 2, 1, 5), (3, 2, 1, 6),
(3, 2, 1, 7), (3, 2, 1, 8), (3, 2, 2, 1), (3, 2, 2, 2), (3, 2, 2, 3), (3, 2, 2, 4), (3, 2, 2, 5),
(3, 2, 2, 6), (3, 2, 2, 7), (3, 2, 2, 8), (3, 3, 1, 1), (3, 3, 1, 2), (3, 3, 1, 3), (3, 3, 1, 4),
(3, 3, 1, 5), (3, 3, 1, 6), (3, 3, 1, 7), (3, 3, 1, 8), (3, 3, 2, 1), (3, 3, 2, 2), (3, 3, 2, 3),
(3, 3, 2, 4), (3, 3, 2, 5), (3, 3, 2, 6), (3, 3, 2, 7), (3, 3, 2, 8), (3, 4, 1, 1), (3, 4, 1, 2),
(3, 4, 1, 3), (3, 4, 1, 4), (3, 4, 1, 5), (3, 4, 1, 6), (3, 4, 1, 7), (3, 4, 1, 8), (3, 4, 2, 1),
(3, 4, 2, 2), (3, 4, 2, 3), (3, 4, 2, 4), (3, 4, 2, 5), (3, 4, 2, 6), (3, 4, 2, 7), (3, 4, 2, 8),
(3, 5, 1, 1), (3, 5, 1, 2), (3, 5, 1, 3), (3, 5, 1, 4), (3, 5, 1, 5), (3, 5, 1, 6), (3, 5, 1, 7),
(3, 5, 1, 8), (3, 5, 2, 1), (3, 5, 2, 2), (3, 5, 2, 3), (3, 5, 2, 4), (3, 5, 2, 5), (3, 5, 2, 6),
(3, 5, 2, 7), (3, 5, 2, 8)]
```

3. Neighborhood Permutasi

   Membuat *neighborhood* permutasi cukup dengan menggunakan `list(permutations(range(integer awal, integer akhir - 1))` seperti yang ditunjukkan baris ke-4. Jadi, apabila ada integer awal = 2 dan integer akhir = 6, berarti permutasi yang terbentuk akan tersusun atas nilai integer = `{2, 3, 4, 5, 6}`

```
1.  import itertools
2.  from itertools import permutations
3.
4.  neighborhood = list(permutations(range(1,5)))
5.
6.  print(neighborhood)
```

Output

```
[(1, 2, 3, 4), (1, 2, 4, 3), (1, 3, 2, 4), (1, 3, 4, 2), (1, 4, 2, 3), (1, 4, 3, 2), (2, 1, 3, 4),
(2, 1, 4, 3), (2, 3, 1, 4), (2, 3, 4, 1), (2, 4, 1, 3), (2, 4, 3, 1), (3, 1, 2, 4), (3, 1, 4, 2),
(3, 2, 1, 4), (3, 2, 4, 1), (3, 4, 1, 2), (3, 4, 2, 1), (4, 1, 2, 3), (4, 1, 3, 2), (4, 2, 1, 3),
(4, 2, 3, 1), (4, 3, 1, 2), (4, 3, 2, 1)]
```

1.7. POST TEST

Jawablah pertanyaan berikut (**Total Skor: 100):**

No	CPMK	Sub-CPMK	Pertanyaan	Skor

1.	01, 02	03	Pilihlah salah satu langkah praktikum yang telah dikerjakan tadi. Kemudian tampilkan satu hasil permutasi saja yang indeksnya dibangkitkan secara acak. Misalnya dipilih langkah ke-3 dan indeks acak terpilih adalah 1, maka outputnya adalah: `(1, 2, 4, 3)`	100

1.8. HASIL CAPAIAN PRAKTIKUM

Diisi oleh asisten setelah semua assessment dinilai.

No	Bentuk Assessment	CPMK	Sub-CPMK	Bobot	Skor (0-100)	Nilai Akhir (Bobot x Skor)
1.	Pre-Test	01,02	02	20%		
2.	Praktik		03	30%		
3.	Post-Test			50%		
					Total Nilai	

LEMBAR JAWABAN PRE-TEST DAN POST-TEST PRAKTIKUM

Nama : **NIM :**	**Asisten:** **Paraf Asisten:**	**Tanggal:** **Nilai:**

PRAKTIKUM 3: SIMULATED ANNEALING

Pertemuan ke : 2 dan 3

Total Alokasi Waktu : 90 menit
- Materi : 15 menit
- Pre-Test : 15 menit
- Praktikum : 45 menit
- Post-Test : 15 menit

Total Bobot Penilaian : 100%
- Pre-Test : 20 %
- Praktik : 50 %
- Post-Test : 30 %

Pemenuhan CPL dan CPMK:

CPMK-01	Mampu menerapkan konsep teoritis bidang area Informatika terkait matematika dasar dan ilmu komputer untuk memodelkan masalah dan meningkatkan produktivitas
CPMK-02	Mampu memilih, membuat dan menerapakan teknik, sumber daya, penggunaan perangkat teknik modern dan implementasi berbagai teknik optimasi untuk memecahkan masalah
Sub-CPMK-04	Mampu menjelaskan dan menyelesaikan permasalahan optimasi metaheuristik tunggal atau pencarian lokal

1.1. DESKRIPSI CAPAIAN PEMBELAJARAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu menyelesaikan permasalahan optimasi metaheuristik berbasis pencarial lokal menggunakan algoritme Simulated Annealing.

1.2. INDIKATOR KETERCAPAIAN PEMBELAJARAN

Indikator ketercapaian diukur dengan:

CPMK-01,02	Sub-CPMK-04	Kesesuaian hasil implementasi algoritme pencarian lokal menggunakan bahasa pemrograman dengan permasalahan optimasi yang diberikan

1.3. TEORI PENDUKUNG

Algoritme Simulated Annealing (SA) mengadopsi dari proses pelelehan besi hingga menjadi bentuk yang keras. Algoritme SA bekerja pertama-tama dengan menentukan setting paramter atau *cooling schedule*, menetapkan variabel desain secara acak agar bisa menghasilkan solusi awal S_0 sesuai dengan fungsi objektifnya, dan temperatur awal T_{max}. Selanjutnya lakukan iterasi selama temperatur lebih besar dari nilai berhenti. Di setiap iterasi lakukan iterasi terdalam (*inner loop*) sebanyak iterasi maksimum yang di dalamnya melakukan:

1. Membangkitkan s' sebagai neighbor
2. Hitung ΔE
3. Hitung nilai metropolis
4. Lakukan seleksi apabila $\Delta E \leq 0$ atau random(0,1) < metropolis, maka solusi terbaik ditemukan
5. Sedangkan bila tidak terpenuhi maka perbarui temperatur

Algoritme Simulated Annealing

```
1.  Input: Setting parameter
2.  s = s0 /*Sesuai fungsi objektif */
3.  T = Tmax /* Sesuai teknik yang dipilih */
4.  while T > stoppingValue
5.     for i in range(maxIter)
6.        Bangkitkan neighbor s' secara acak
7.        ΔE = f(s') − f(s)
8.        metropolis = e^(−ΔE/kT)
9.        if ΔE ≤ 0 or random(0,1) < metropolis
10.          Output: Solusi terbaik ditemukan
11.       else
12.          T = c x T
13.    End for
14. End while
```

1.4. HARDWARE DAN SOFTWARE

Hardware dan software yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Python, C/C++, PHP atau C#, dan lain sebagainya yang sudah dikuasai sebelumnya

1.5. PRE-TEST

Jawablah pertanyaan berikut (**Total Skor: 100):**

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1.	01,02	04	Diberikan sebuah variabel x bernilai awal = 100. Buatlah iterasi yang dalam setiap iterasinya mengurangi nilai x tersebut sebesar 15%. Iterasi berhenti apabila nilai x < 10 Contoh output: 100,85,72,61,52,44,38,32,27,23,20,17,14,12,10	100

1.6. LANGKAH PRAKTIKUM

Aturan Penilaian (Total Skor: 100):

No	CPMK	Sub-CPMK	Pertanyaan	Dokumen Pendukung	Skor
1.	01,02	04	Selesaikan langkah praktikum 2	Hasil praktikum langkah 2	10
2.			Selesaikan langkah praktikum 3	Hasil praktikum langkah 3	5
3.			Selesaikan langkah praktikum 4	Hasil praktikum langkah 4	5
4.			Selesaikan langkah praktikum 5	Hasil praktikum langkah 5	5
5.			Selesaikan langkah praktikum 6	Hasil praktikum langkah 6	25
6.			Selesaikan langkah praktikum 7	Hasil praktikum langkah 7	20
7.			Selesaikan langkah praktikum 8	Hasil praktikum langkah 8	30

Langkah-Langkah Praktikum:

1. Tentukan fungsi objektif

   Minimumkan:

   $$f(x) = x^2$$

   dengan batasan: $-5 \leq x \leq 5$ bertipe *real*

2. Tentukan *setting* parameter

   Ada empat parameter utama yang dibuat yaitu:
   - rentang variabel antara -5 hingga 5
   - jumlah solusi awal yang akan digunakan untuk menghitung temperatur awal T_0
   - Iterasi maksimum, dan
   - Nilai berhenti yang akan dibandingkan dengan solusi yang ditemukan

   Selanjutnya buatlah sebuah *class* (baris 1), buat konstruktornya (baris 2-7) dan buat instansiasinya (baris 15)

```
1.  class SimulatedAnnealing:
2.      def __init__(self, varRanges, numOfInitSolution, maxIter, stoppingValue, minTemperature):
3.          self.varRanges = varRanges
4.          self.numOfInitSolution = numOfInitSolution
5.          self.maxIter = maxIter
6.          self.stoppingValue = stoppingValue
7.          self.minTemperature = minTemperature
8.
9.  varRanges = [-5,5]
10. numOfInitSolution = 4
11. maxIter = 5
12. stoppingValue = 0.005
13. minTemperature = 1
14.
15. run = SimulatedAnnealing(varRanges, numOfInitSolution, maxIter, stoppingValue, minTemperature)
```

3. Hitung temperatur awal

   Pertama tama kita buat *method* `getSolution` untuk menghitung fungsi minimum yang diformulasikan pada bagian teori. Method ini akan menggunakan sebuah parameter yang akan mengembalikan nilai hasil pangkat dua.

```
1.       def getSolution(self, var):
2.           return var**2
```

   Selanjutnya kita buat *method* `getInitialTemperature` untuk mendapatkan solusi awal. Solusi awal ini akan memanggil *method* `getSolution` yang nilai parameternya menggunakan fungsi `random.uniform(batasAwal, batasAkhir)` (baris 4) yaitu sesuai dengan variabel `varRanges = [-5,5]`. Pemangilan ini dilakukan sebanyak `numOfInitSolution` yaitu 4. Artinya akan dihasilkan sebanyak jumlah akumulasi keempat solusi (baris 5) hasil dari `getSolution`. Nilai tersebut kemudian dibagi dengan `self.numOfInitSolution` atau rerata sebagai nilai kembalian (*return*).

```
1.      def getInitTemperature(self):
2.          ret = 0
3.          for _ in range(self.numOfInitSolution):
4.              solution = self.getSolution(random.uniform(self.varRanges[0], self.varRanges[1]))
5.              ret+=solution
6.          return ret / self.numOfInitSolution
```

4. Buat solusi awal

   Solusi awal kita buat dengan memanggil fungsi `getSolution` yang parameternya diisi dengan nilai acak seperti yang sudah dikerjakan pada langkah ke-3 sebelumnya (baris 3-4).

   Langkah ke-2 dan 3 kita panggil di dalam sebuah *method* `mainSA`.

```
1.      def mainSA(self):
2.          temperature = self.getInitTemperature(self.getInitSolution())
3.          solutionVals = random.uniform(self.varRanges[0],self.varRanges[1])
4.          solution = self.getSolution(solutionVals)
```

5. Buat kandidat *neighbor*

   Untuk membuat sebuah titik *neighbor* maka kita fasilitasi dengan *method* `getCandidate` yang menerima parameter `varRanges`. Pertama-tama bangkitkan bilangan acak antara 0-1 (baris 2). Setelah nilai baru diperoleh, lakukan perhitungan seperti pada baris 3 berikut.

```
1.      def getCandidate(self, varRanges):
2.          u = random.uniform(0,1)
3.          val = varRanges[0] + u * (varRanges[1] - varRanges[0])
```

```
4.          return val
```

6. Buat rentang nilai variabel baru

Setiap kali selesai membuat titik *neighbor* baru, maka perlu dilanjutkan dengan membuat rentang variabel baru dengan menggunakan parameter dari kandidat *neighbor* yang dihasilkan dari langkah ke-5. Baris ke-2 dan 3 di *method* `getNewVarRanges` untuk menghitung batas bawah dan batas atas rentang. Baris ke-4 hingga 7 digunakan untuk menyeleksi apakah nilai rentang baru melewati batas bawah atau batas atas yaitu [-5, 5].

```
1.      def getNewVarRanges(self, candidate):
2.          lowerBound = -6 + candidate
3.          upperBound = 6 + candidate
4.          if lowerBound < self.varRanges[0]:
5.              lowerBound = self.varRanges[0]
6.          if upperBound > self.varRanges[1]:
7.              upperBound = self.varRanges[1]
8.          return [lowerBound, upperBound]
```

7. Buat kandidat titik *neighbor* dan rentang variabel baru awal

Baris 6 dan 7 memanggil *method* `getCandidate` dan `getNewVarRanges` yang telah dibuat pada langkah ke-5 dan 6. Keduanya berguna untuk membuat titik *neighbor* dan rentang variabel baru awal yang akan digunakan dalam iterasi pencarian solusi optimum.

```
1.      def mainSA(self):
2.          temperature = self.getInitTemprature()
3.          solutionVals = random.uniform(self.varRanges[0],self.varRanges[1])
4.          solution = self.getSolution(solutionVals)
5.
6.          candidate = self.getCandidate(self.varRanges)
7.          varRanges = self.getNewVarRanges(candidate)
```

Jika dijalankan outpunya adalah:

```
-2.34559424679695 [-5, 3.65440575320305]
```

8. Temukan solusi optimum

Ini adalah langkah terakhir sekaligus bagian inti dari Simulated Annealing. Iterasi `while` baris ke-9 hingga 25 akan terus dijalankan hingga dua kondisi terpenuhi yaitu nilai `solution` lebih kecil dari `stoppingValue` dan `temperature` kurang dari sama dengan `minTemperature`. Apabila tidak terpenuhi, maka nilai `temperature` akan dikurangi (baris 25).

Di setiap iterasi juga akan ada iterasi `for` sebanyak `maxIter`. Iterasi ini akan membandingkan antara `solution` dan `neighbor`. Apabila `deltaE` lebih kecil sama dengan `0` atau nilai acak (`0,1`) lebih kecil dari persamaan `metropolis`, maka `solution` saat ini diganti oleh `neighbor` (baris 19).

```
1.      def mainSA(self):
2.          temperature = self.getInitTemprature()
3.          solutionVals = random.uniform(self.varRanges[0],self.varRanges[1])
4.          solution = self.getSolution(solutionVals)
5.
6.          candidate = self.getCandidate(self.varRanges)
7.          varRanges = self.getNewVarRanges(candidate)
8.
9.          while temperature > self.stoppingValue:
10.             for i in range(self.maxIter):
11.                 candidate = self.getCandidate(varRanges)
12.                 varRanges = self.getNewVarRanges(candidate)
13.
14.                 neighbor = self.getSolution(candidate)
15.                 deltaE = neighbor - solution
16.                 metropolis = exp(-deltaE/temperature)
17.
18.                 if deltaE <= 0 or random.uniform(0,1) < metropolis:
19.                     solutionVals, solution = candidate, neighbor
20.
21.             if solution < self.stoppingValue and temperature <= self.minTemperature:
22.                 print(solution, solutionVals)
23.                 break
24.             else:
25.                 temperature = 0.8 * temperature
```

Output

```
0.0018774879272642209 0.04332998877526073
```

1.7. POST TEST

Jawablah pertanyaan berikut (**Total Skor: 100):**

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1.	01,02	04	Gantilah langkah praktikum ke-1 dengan masalah optimasi berikut: Minimumkan: $f(x,y) = x^2 + y^2$ dengan batasan: $-5.12 \leq x, y \leq 5.12$ Temukan solusi minimumnya!	100

1.8. HASIL CAPAIAN PRAKTIKUM

Diisi oleh asisten setelah semua assessment dinilai.

No	Bentuk Assessment	CPMK	Sub-CPMK	Bobot	Skor (0-100)	Nilai Akhir (Bobot x Skor)
1.	Pre-Test	01,02	04	20%		
2.	Praktik			50%		
3.	Post-Test			30%		
					Total Nilai	

LEMBAR JAWABAN PRE-TEST DAN POST-TEST PRAKTIKUM

Nama : **NIM :**	**Asisten:** **Paraf Asisten:**	**Tanggal:** **Nilai:**

PRAKTIKUM 4: ANT COLONY OPTIMIZATION (ACO) UNTUK TRAVELING SALESPERSON PROBLEM (TSP)

Pertemuan ke : 4 dan 5

Total Alokasi Waktu : 90 menit
- Materi : 15 menit
- Pre-Test : 15 menit
- Praktikum : 45 menit
- Post-Test : 15 menit

Total Bobot Penilaian : 100%
- Pre-Test : 20 %
- Praktik : 50 %
- Post-Test : 30 %

Pemenuhan CPL dan CPMK:

CPMK-03	Mampu merancang dan mengimplementasikan algoritma/metode optimasi dalam mengidentifikasi dan memecahkan masalah yang melibatkan perangkat lunak dan pemikiran komputasi
Sub-CPMK-07	Mampu menyelesaikan permasalahan optimasi metaheuristik menggunakan algoritme berbasis *swarm intelligence*

1.1. DESKRIPSI CAPAIAN PEMBELAJARAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu membuat program optimasi metaheuristik berbasis *swarm intelligence*

1.2. INDIKATOR KETERCAPAIAN PEMBELAJARAN

Indikator ketercapaian diukur dengan:

CPMK-03	Sub-CPMK-07	Ketepatan hasil implementasi algoritme *swarm intelligence* menggunakan bahasa pemrograman sesuai masalah optimasi yang diberikan

1.3. TEORI PENDUKUNG

Ant Colony Optimization (ACO) ditemukan oleh Marco Dorigo pada tahun 1992 sebagai hasil penelitian Doktoralnya di *Politecnico di Milano*, Italia. ACO terinspirasi dari perilaku kawanan semut dalam menemukan sumber makanan. Semut menggunakan feromon sebagai media komunikasi berbasis biologi. Feromon bisa disebut sebagai jejak yang ditinggalkan di jalan setiap kali semut melintas. Semakin banyak semut yang melintas di jalan tersebut, semakin kuat feromonnya. Hal ini menjadi dasar utama mengapa ACO pada awalnya memang digunakan untuk memecahkan masalah pencarian rute terpendek.
Secara umum, pencarian rute terpendek menggunakan ACO dengan menerapkan formulasi sebagai berkut:

Probabilitas pemilihan rute berikutnya

$$p_{ij}^{t} = \frac{\left[\tau_{ij}^{t}\right]^{\alpha}\left[\frac{1}{d_{ij}}\right]^{\beta}}{\sum_{j \notin tabu\ list}\left[\tau_{ij}^{t}\right]^{\alpha}\left[\frac{1}{d_{ij}}\right]^{\beta}}$$

Perbarui feromon

$$\tau_{ij}^{t+1} = (1-\rho)\tau_{ij}^{t} + \sum_{k=1}^{m} \Delta\tau_{ij}^{(k)}$$

dengan

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, & \text{jika semut ke} - k \text{ melewati pasangan } (i,j) \\ 0, & \text{jika yang lain} \end{cases}$$

$\rho \in [0,1]$ = tingkat penguapan feromon
$\Delta\tau_{ij}^{(k)} = \frac{Q}{L_k}$ = jumlah feromon ditambahkan pada waktu t sepanjang rute i ke j ketika semut ke-k berjalan dengan jarak perjalanannya L_k
Q= konstanta positif yang digunakan untuk penambahan feromon

1.4. HARDWARE DAN SOFTWARE

Hardware dan software yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Python, C/C++, PHP atau C#, dan lain sebagainya yang sudah dikuasai sebelumnya

1.5. PRE-TEST

Jawablah pertanyaan berikut (**Total Skor: 100**):

No	CPMK	Sub-CPMK	Pertanyaan	Skor

1.	03	07	Buatlah program untuk mencetak matriks berukuran 4x4 yang nilai tiap selnya berupa bilangan yang dibangkitkan secara acak bertipe *real* antara 1-10.
			Contoh outputnya:
			`2.12 3.00 1.9 4.5`
			`5.81 7.04 2.0 5.5`
			`1.11 2.45 3.1 2.08`
			`9.6 8.32 7.5 8.8`
				100

1.6. LANGKAH PRAKTIKUM

Aturan Penilaian (Total Skor: 100):

No	CPMK	Sub-CPMK	Pertanyaan	Dokumen Pendukung	Skor
1.	03	07	Selesaikan langkah praktikum 1	Hasil praktikum langkah 1	5
2.			Selesaikan langkah praktikum 2	Hasil praktikum langkah 2	5
3.			Selesaikan langkah praktikum 3	Hasil praktikum langkah 3	10
4.			Selesaikan langkah praktikum 4	Hasil praktikum langkah 4	10
5.			Selesaikan langkah praktikum 5	Hasil praktikum langkah 5	10
6.			Selesaikan langkah praktikum 6	Hasil praktikum langkah 6	10
7.			Selesaikan langkah praktikum 7	Hasil praktikum langkah 7	10
8.			Selesaikan langkah praktikum 8	Hasil praktikum langkah 8	10
9.			Selesaikan langkah praktikum 9	Hasil praktikum langkah 9	10
10.			Selesaikan langkah praktikum 10	Hasil praktikum langkah 10	10
11.			Selesaikan langkah praktikum 11	Hasil praktikum langkah 11	10

Langkah-Langkah Praktikum:

1. Buat variabel awal dan *setting* parameter

```
1.  matriks = [
```

```
2.      [0, 360, 185, 335, 160, 340, 334, 362, 163, 204],
3.      [360, 0, 293, 579, 269, 601, 583, 610, 370, 318],
4.      [185, 293, 0, 405, 261, 408, 409, 202, 80.6, 21.4],
5.      [335, 579, 405, 0, 313, 4, 24, 56.5, 164, 241],
6.      [160, 269, 26.1, 313, 0, 383, 382, 225, 104, 45.8],
7.      [340, 601, 408, 4, 383, 0, 22.5, 56.5, 164, 244],
8.      [334, 583, 409, 24, 382, 22.5, 0, 75.9, 181, 336],
9.      [362, 610, 202, 56.5, 225, 56.5, 75.9, 0, 120, 200],
10.     [163, 370, 80.6, 164, 104, 164, 181, 120, 0, 80.9],
11.     [204, 318, 21.4, 241, 45.8, 244, 336, 200, 80.9, 0]
12. ]
13.
14. cityNames = [
15.     "Jogja", "Sunan Gunung Jati (Cirebon)", "Sunan Kudus", "Sunan Giri (Gresik)", "Sunan
    Kalijaga (Demak)",
16.     "Sunan Gresik", "Sunan Ampel (Surabaya)", "Sunan Drajat (Lamongan)", "Sunan Bonang
    (Tuban)", "Sunan Muria (Kudus)"
17. ]
18.
19. parameters = {
20.     'Q': 100,
21.     'rho': 0.6,
22.     'antSize': 15,
23.     'matriks': matriks,
24.     'maxIter': 25,
25.     'cityNames': cityNames
26. }
```

2. Buat *class* dan konstruktor terdiri dari dua parameter, yaitu `parameters` dan `start`. `Start` berguna untuk menentukan apakah alamat atau kota awal

```
1.  import random, sys
2.
3.  class AntColonyOptimizationTSP:
4.      def __init__(self, parameters, start):
5.          self.params = parameters
6.          self.start = start
```

3. Buat *method* utama `ACOTSProblem`

Baris ke-2 mendeklarasikan variabel `tabuList` yang akan digunakan untuk menyimpan seluruh alamat yang dikunjungi setiap semut. Baris ke-4 s/d 9 akan menyeleksi apakah titik awal kota yang akan dikunjungi setiap semut berawal dari kota yang sama (baris-5 dan 6) atau berawal dari kota yang berbeda (baris ke-7 dan 8).

```
1.      def ACOTSProblem(self):
2.          tabuList = []
3.          for iter in range(self.params['maxIter']):
4.              for i in range(self.params['antSize']):
5.                  if self.start:
6.                      nextCity = self.start[0]
7.                  else:
8.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
9.                  tabuList.append([nextCity])
10.             print(tabuList)
11.             tabuList = []
```

Kita coba instansiasi

```
aco = AntColonyOptimizationTSP(parameters, start = [0])
aco.ACOTSProblem()
```

Hasilnya:

```
[[0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0]]
```

Berarti titik berangkat setiap semut berawal di kota ke-0, yaitu Jogja. Bila `start = [1]` berarti titik awal berangkat semut berawal di kota Cirebon (Sunan Gunung Jati), begitu seterusnya.

Sebaliknya bila `start = []` atau tidak diisi, maka setiap semut akan ditugaskan ke titik berangkat/awal yang berbeda-beda. Hal ini dikarenakan fungsi acak yang dieksekusi oleh baris ke-8. Contoh hasilnya adalah:

```
[[9], [1], [1], [1], [3], [0], [1], [5], [7], [9], [0], [6], [3], [0], [7]]
```

Apabila baris ke-3 dieksekusi maka akan menghasilkan sebanyak 15 `tabuList` sesuai dengan parameter `maxIter` seperti contoh di berikut

```
[[7], [5], [9], [4], [3], [2], [1], [4], [3], [2], [5], [4], [4], [6], [4]]
[[4], [4], [9], [6], [3], [8], [6], [6], [5], [6], [4], [5], [1], [0], [0]]
[[3], [4], [3], [7], [7], [5], [0], [6], [0], [9], [0], [1], [8], [2], [1]]
[[5], [9], [4], [9], [3], [4], [1], [6], [8], [4], [7], [8], [7], [1], [0]]
```

```
[[6], [3], [5], [8], [3], [0], [1], [3], [8], [7], [4], [3], [9], [8], [2]]
[[6], [5], [3], [9], [4], [1], [8], [1], [1], [9], [3], [7], [3], [5], [8]]
[[0], [9], [3], [0], [5], [2], [6], [2], [8], [8], [3], [8], [2], [9], [2]]
[[1], [0], [5], [4], [2], [5], [5], [8], [6], [3], [6], [8], [7], [3], [7]]
[[5], [0], [9], [3], [8], [0], [7], [3], [3], [2], [9], [1], [8], [4], [7]]
[[3], [4], [2], [8], [6], [5], [6], [4], [7], [8], [7], [6], [1], [5], [7]]
[[4], [5], [6], [3], [2], [3], [5], [2], [6], [7], [9], [9], [3], [8], [0]]
[[5], [2], [7], [5], [1], [1], [0], [1], [3], [3], [1], [3], [3], [6], [9]]
[[3], [3], [5], [4], [9], [1], [0], [9], [1], [1], [8], [6], [1], [6], [0]]
[[5], [0], [9], [2], [1], [9], [4], [7], [6], [2], [9], [7], [0], [7], [1]]
[[3], [2], [5], [9], [1], [5], [6], [4], [8], [1], [0], [0], [9], [2], [2]]
[[7], [8], [7], [3], [6], [1], [3], [9], [7], [5], [4], [2], [5], [2], [3]]
[[9], [1], [5], [4], [8], [0], [5], [4], [6], [1], [4], [9], [0], [7], [2]]
[[3], [4], [7], [1], [5], [9], [9], [6], [6], [4], [0], [6], [6], [3], [5]]
[[6], [3], [4], [4], [9], [3], [2], [3], [0], [3], [8], [5], [2], [6], [1]]
[[3], [1], [6], [6], [4], [7], [4], [8], [1], [3], [8], [4], [0], [3], [7]]
[[2], [3], [5], [7], [9], [2], [3], [3], [7], [7], [2], [2], [5], [1], [5]]
[[5], [5], [0], [0], [2], [5], [9], [1], [0], [9], [5], [9], [5], [7], [7]]
[[8], [2], [3], [4], [9], [4], [2], [7], [0], [2], [1], [9], [0], [1], [1]]
[[7], [7], [0], [8], [1], [3], [6], [2], [8], [0], [9], [6], [8], [8], [3]]
[[4], [3], [0], [1], [2], [9], [0], [0], [8], [7], [9], [3], [2], [0], [2]]
```

4. Dapatkan tujuan alamat berikutnya

Dimulai dari baris ke-14 dengan mendeklarasikan larik **`temp`**, **`city`**, **`pairedCity`**, dan **`matriks`**. Iterasi terluar (*outer loop*) atau iterasi pertama di baris ke-15 akan berlangsung sebanyak ukuran matriks-1. Di setiap iterasi terluar ini terlebih dahulu menjalankan iterasi sebanyak isi **`tabuList`** saat itu (baris ke-16) atau disebut iterasi kedua.

Di setiap iterasi **`tabuList`** ini akan dilakukan:

- Pembangkitan bilangan acak **`r`** di antara **`0`** dan **`1`** (baris ke-17)
- Lakukan iterasi lagi (iterasi ketiga) sebanyak jumlah matriks (baris ke-19). Di dalam iterasi ketiga ini lakukan:
  - Iterasi keempat sebanyak isi **`tabuList[j]`** (baris ke-20). Di tiap iterasi keempat ini melakukan:
    a. Menambah elemen berupa isi **`tabuList`** (baris ke-21) dan **`cityID`** (baris ke-22) ke variabel **`temp`**.
    b. Baris 23-28 akan menyeleksi bahwa apabila banyak isi elemen **`tabuList[j]`** sama dengan banyaknya elemen **`cityID`** pada larik **`temp`** (baris ke-23), maka dilakukan penambahan elemen ke larik **`city`** berupa isi elemen terakhir larik **`tabuList[j]`** dan **`cityID`** sendiri (baris 24 dan 25). Sedangkan jika tidak terpenuhi maka tambahkan elemen ke larik **`city`** berupa **`cityID`** dan **`cityID`**. Baris 24 dan 25 ini menunjukkan bahwa pasangan kota atau alamatnya sama. Contoh Jogja ke Jogja.
    c. Larik city yang sudah diisi tersebut akan dimasukkan ke larik pairedCity (baris ke-29)

```
1.        def ACOTSProblem(self):
2.          tabuList = []
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
11.             print(tabuList)
12.             #tabuList = []
13.
14.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
15.             for i in range(len(matriks)-1):
16.                 for j in range(len(tabuList)):
17.                     r = random.uniform(0,1)
18.                     # print(tabuList[j])
19.                     for cityID in range(len(matriks)):
20.                         for k in tabuList[j]:
21.                             temp.append(k)
22.                             temp.append(cityID)
23.                         if temp.count(cityID) == len(tabuList[j]):
24.                             city.append(tabuList[j][-1])
25.                             city.append(cityID)
26.                         else:
27.                             city.append(cityID)
28.                             city.append(cityID)
```

```
29.                         pairedCity.append(city)
30.                         city = []; temp = []
31.                     print(pairedCity)
32.                     pairedCity = []
33.                 sys.exit()
```

Hasil sementara apabila kita jalankan adalah seperti berikut:

```
iterasi ke- 0
[[1], [3], [4], [3], [8], [7], [8], [1], [4], [3], [0], [3], [8], [7], [8]]
[[1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9]]
[[3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9]]
[[4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [4, 5], [4, 6], [4, 7], [4, 8], [4, 9]]
[[3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9]]
[[8, 0], [8, 1], [8, 2], [8, 3], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9]]
[[7, 0], [7, 1], [7, 2], [7, 3], [7, 4], [7, 5], [7, 6], [7, 7], [7, 8], [7, 9]]
[[8, 0], [8, 1], [8, 2], [8, 3], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9]]
[[1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9]]
[[4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [4, 5], [4, 6], [4, 7], [4, 8], [4, 9]]
[[3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9]]
[[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9]]
[[3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9]]
[[8, 0], [8, 1], [8, 2], [8, 3], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9]]
[[7, 0], [7, 1], [7, 2], [7, 3], [7, 4], [7, 5], [7, 6], [7, 7], [7, 8], [7, 9]]
[[8, 0], [8, 1], [8, 2], [8, 3], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9]]
```

Daro output di atas terlihat bahwa ke-15 semut memiliki titik awal dari kota/alamat yang berbeda-beda. Semut-1 dari kota 1, semut-2 dari kota 3, semut-3 dari kota 4, semut-4 dari kota 3 dan seterusnya hingga semut-15 dari kota 8,
Perhatikan pula bahwa bisa terjadi bahwa ada lebih dari satu semut yang bermula dari kota yang sama. Contohnya semut-2, 4, 10 dan 12 yang sama-sama berawal dari kota 4.

5. Buat *method* `getDistance`

Method ini digunakan untuk mendapatkan jarak berdasarkan pasangan matriks antara baris dan kolom. Sebagai contoh misalnya ada pasangan matriks [3, 1] maka jika dilihat dari variabel larik matriks di langkah ke-1 ditemukan di baris ke-4 tepat di kolom yaitu sebesar 185.

Baris ke-3 akan melakukan iterasi sebanyak elemen di dalam larik **`pairedCities`** (lihat langkah ke-4 di bagian output). Di dalam iterasi ini juga ada iterasi sebanyak ukuran matriks (baris ke-4). Iterasi terdalam (*inner loop*) juga dieksekusi di baris ke-5 sebanyak ukuran **`matriks[j]`** atau baris. Baris ke-6 dan 7 menyeleksi jika nilai elemen ke-i indeks ke-0 dari larik **`pairedCities`** sama dengan indeks **`j`** matriks dan nilai elemen ke-i indeks ke-1 dari larik **`pairedCities`** sama dengan indeks **`k`** **`matriks[j]`**. Apabila keduanya terpenuhi maka nilai **`matriks[j][k]`** ditambahkan ke larik **`rets`**.

```
1.      def getDistance(self, pairedCities):
2.          rets = []
3.          for i in pairedCities:
4.              for j in range(len(self.params['matriks'])):
5.                  for k in range(len(self.params['matriks'][j])):
6.                      if i[0] == j and i[1] == k:
7.                          rets.append(self.params['matriks'][j][k])
8.          return rets
```

Jika dijalankan maka nilai kembalian **`rets`** berupa:

```
[[5, 0], [5, 1], [5, 2], [5, 3], [5, 4], [5, 5], [5, 6], [5, 7], [5, 8], [5, 9]]
[340, 601, 408, 4, 383, 0, 22.5, 56.5, 164, 244]
```

Selanjutnya kita panggil methodnya di dalam method utama **`ACOTSProblem`** di baris ke-32.

```
1.      def ACOTSProblem(self):
2.          tabuList = []
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
11.             print(tabuList)
12.             #tabuList = []
13.
14.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
```

```
            for i in range(len(matriks)-1):
                for j in range(len(tabuList)):
                    r = random.uniform(0,1)
                    # print(tabuList[j])
                    for cityID in range(len(matriks)):
                        for k in tabuList[j]:
                            temp.append(k)
                            temp.append(cityID)
                        if temp.count(cityID) == len(tabuList[j]):
                            city.append(tabuList[j][-1])
                            city.append(cityID)
                        else:
                            city.append(cityID)
                            city.append(cityID)
                        pairedCity.append(city)
                        city = []; temp = []
                    print(pairedCity)
                    pairedDistances = self.getDistance(pairedCity)
                    pairedCity = []
                    print(pairedDistances)
                    print()
                pairedDistances = []
                sys.exit()
```

Hasilnya menjadi:

```
[[6], [9], [3], [4], [3], [0], [8], [9], [0], [3], [3], [7], [2], [9], [2]]
[[6, 0], [6, 1], [6, 2], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6, 9]]
[334, 583, 409, 24, 382, 22.5, 0, 75.9, 181, 336]

[[9, 0], [9, 1], [9, 2], [9, 3], [9, 4], [9, 5], [9, 6], [9, 7], [9, 8], [9, 9]]
[204, 318, 21.4, 241, 45.8, 244, 336, 200, 80.9, 0]

[[3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9]]
[335, 579, 405, 0, 313, 4, 24, 56.5, 164, 241]

[[4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [4, 5], [4, 6], [4, 7], [4, 8], [4, 9]]
[160, 269, 26.1, 313, 0, 383, 382, 225, 104, 45.8]

...

[[2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [2, 9]]
[185, 293, 0, 405, 261, 408, 409, 202, 80.6, 21.4]
```

6. Buat *method* `getProbNextCities`

Method ini digunakan untuk menghitung probabilitas tujuan kota atau alamat berikutnya. Method ini akan menerima dua argumen yaitu `distancePaired` dan `feromon`. Parameter `distancePaired` didapatkan dari langkah ke-5. Sedangkan `feromon` diperoleh dari perhitungan feromon awal dan perubahan feromon di tiap iterasinya.

Baris ke-3 menjalankan iterasi sebanyak elemen dalam larik `distancePaired`. Baris ke-4 hingga 7 menyeleksi nilai `distance`. Jika `distance` bernilai `0` maka `val` diisi `0`. Selain `0` maka `val` diisi hasil dari `1/distance` dikali `feromon`. Berapapun nilai `val` akan dimasukkan ke larik `ret` (baris ke-8).

```
1.      def getProbNextCities(self, distancePaired, feromon):
2.          ret = []
3.          for distance in distancePaired:
4.              if distance == 0:
5.                  val = 0
6.              else:
7.                  val = (1/distance) * feromon
8.              ret.append(val)
9.          return ret
```

Untuk menjalankan *method* ini, maka di method utama `ACOTSProblem` harus dideklarasikan `feromon` awal (baris ke-2). Selanjutnya panggil method `getProbNextCities` seperti pada baris ke-34.

```
1.      def ACOTSProblem(self):
2.          tabuList = []; feromon = 1/len(self.params['matriks'])
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
```

```
11.             print(tabuList)
12.             #tabuList = []
13.
14.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
15.             for i in range(len(matriks)-1):
16.                 for j in range(len(tabuList)):
17.                     r = random.uniform(0,1)
18.                     # print(tabuList[j])
19.                     for cityID in range(len(matriks)):
20.                         for k in tabuList[j]:
21.                             temp.append(k)
22.                             temp.append(cityID)
23.                         if temp.count(cityID) == len(tabuList[j]):
24.                             city.append(tabuList[j][-1])
25.                             city.append(cityID)
26.                         else:
27.                             city.append(cityID)
28.                             city.append(cityID)
29.                         pairedCity.append(city)
30.                         city = []; temp = []
31.                     print(pairedCity)
32.                     pairedDistances = self.getDistance(pairedCity)
33.                     pairedCity = []
34.                     probNextCities = self.getProbNextCities(pairedDistances, feromon)
35.                     print(probNextCities, sum(probNextCities))
36.                     print(pairedDistances)
37.                     print()
38.                 sys.exit()
```

Jika dijalankan maka outputnya akan menjadi sebagai berikut. Adapun `sum(probNextCities)` di baris ke-35 merupakan penjumlahan seluruh elemen dalam larik `probNextCities`.

```
iterasi ke- 0
[[9], [9], [6], [8], [0], [4], [4], [2], [8], [2], [3], [2], [7], [8], [0]]
[[9, 0], [9, 1], [9, 2], [9, 3], [9, 4], [9, 5], [9, 6], [9, 7], [9, 8], [9, 9]]
[0.0004901960784313725, 0.0003144654088050315, 0.004672897196261683, 0.0004149377593360963,
0.0021834061135371182, 0.0004098360655737055, 0.0002976190476190476, 0.0005,
0.0012360939431396787, 0] 0.010519451612703802
[204, 318, 21.4, 241, 45.8, 244, 336, 200, 80.9, 0]

[[9, 0], [9, 1], [9, 2], [9, 3], [9, 4], [9, 5], [9, 6], [9, 7], [9, 8], [9, 9]]
[0.0004901960784313725, 0.0003144654088050315, 0.004672897196261683, 0.0004149377593360963,
0.0021834061135371182, 0.0004098360655737055, 0.0002976190476190476, 0.0005,
0.0012360939431396787, 0] 0.010519451612703802
[204, 318, 21.4, 241, 45.8, 244, 336, 200, 80.9, 0]

[[6, 0], [6, 1], [6, 2], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6, 9]]
[0.00029940119760479047, 0.00017152658662092626, 0.0002444987775061125, 0.004166666666666667,
0.0002617801047120419, 0.0044444444444444444, 0, 0.0013175230566534915, 0.0005524861878453039,
0.0002976190476190476] 0.011755946069672825
[334, 583, 409, 24, 382, 22.5, 0, 75.9, 181, 336]

...

[[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9]]
[0, 0.0002777777777777778, 0.0005405405405405405, 0.00029850746268656717, 0.0006250000000000001,
0.0002941176470588235, 0.00029940119760479047, 0.00027624309392265195, 0.0006134969325153375,
0.0004901960784313725] 0.0037152807305378616
[0, 360, 185, 335, 160, 340, 334, 362, 163, 204]
```

7. Buat method `getNextCities`

Method ini akan memberikan kembalian berupa id kota (nilai integer) yang merupakan kota berikutnya yang akan dikunjungi oleh semut.
Parameter yang digunakan adalah `probNextCities` yang diperoleh di langkah ke-6 dan nilai acak `r` antara `0` hingga `1`. Baris ke-4 hingga 7 akan menyeleksi jika nilai hasil penjumlahan `sum(probNextCities)` tidak sama dengan `0` maka variabel `tmp` akan diisi hasil penjumlahan antara `tmp` dengan hasil pembagian nilai `probNextCities[i]` degan `sum(probNextCities)`. Jika tidak memenuhi berarti `tmp` diisi `0`.
Selanjutnya baris ke-8 hingga 10 menyeleksi apakah nilai acak `r` kurang dari `tmp`. Jika ya, maka nilai indeks `i` akan dikembalikan seperti baris ke-11 dan dipaksa berhenti dengan `break` (baris ke-10).

```
1.      def getNextCites(self, probNextCities, r):
2.          tmp = 0
3.          for i in range(len(probNextCities)):
```

```
4.              if sum(probNextCities) != 0:
5.                  tmp = tmp + (probNextCities[i]/sum(probNextCities))
6.              else:
7.                  tmp = 0
8.              if r < tmp:
9.                  i
10.                 break
11.         return i
```

Kemudian kita panggil method **getNextCities** di dalam method utama **ACOTSProblem** (baris ke-34) dan diisi ke variabel **nextCities**. Nah, nilai **nextCities** inilah yang akan ditambahkan ke larik **tabuList[j]** (baris 35).

```
1.      def ACOTSProblem(self):
2.          tabuList = []; feromon = 1/len(self.params['matriks'])
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
11.             print(tabuList)
12.             #tabuList = []
13.
14.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
15.             for i in range(len(matriks)-1):
16.                 for j in range(len(tabuList)):
17.                     r = random.uniform(0,1)
18.                     # print(tabuList[j])
19.                     for cityID in range(len(matriks)):
20.                         for k in tabuList[j]:
21.                             temp.append(k)
22.                             temp.append(cityID)
23.                         if temp.count(cityID) == len(tabuList[j]):
24.                             city.append(tabuList[j][-1])
25.                             city.append(cityID)
26.                         else:
27.                             city.append(cityID)
28.                             city.append(cityID)
29.                         pairedCity.append(city)
30.                         city = []; temp = []
31.                     pairedDistances = self.getDistance(pairedCity)
32.                     pairedCity = []
33.                     probNextCities = self.getProbNextCities(pairedDistances, feromon)
34.                     nextCities = self.getNextCites(probNextCities, r)
35.                     tabuList[j].append(nextCities)
36.             print(tabuList)
37.             sys.exit()
```

Apabila dijalankan maka hasilnya adalah:

```
iterasi ke- 0
[[9], [2], [9], [3], [1], [9], [9], [3], [2], [1], [4], [2], [4], [7], [2]]
[[9, 2, 7, 3, 6, 5, 0, 1, 4, 8],
 [2, 8, 6, 3, 7, 5, 0, 9, 4, 1],
 [9, 4, 8, 1, 6, 3, 7, 5, 2, 0],
 [3, 5, 7, 1, 6, 8, 0, 2, 9, 4],
 [1, 9, 2, 8, 3, 5, 6, 7, 0, 4],
 [9, 2, 8, 4, 0, 3, 7, 5, 6, 1],
 [9, 3, 5, 7, 8, 2, 6, 4, 0, 1],
 [3, 5, 4, 9, 0, 8, 2, 1, 7, 6],
 [2, 4, 9, 3, 6, 5, 1, 8, 0, 7],
 [1, 9, 2, 0, 4, 6, 7, 8, 5, 3], [4, 9, 1, 6, 5, 3, 7, 8, 2, 0], [2, 9, 0, 3, 6, 4, 5, 7, 1, 8], [4,
6, 3, 7, 5, 0, 1, 2, 9, 8], [7, 3, 5, 2, 9, 4, 0, 8, 6, 1], [2, 9, 3, 5, 6, 8, 4, 0, 7, 1]]
```

Perhatikanlah, **tabuList** yang awalnya hanya berisi satu tujuan/kota/alamat sekarang sudah diisi dengan tujuan/kota/alamat berikutnya yang akan dikunjungi dan tidak ada satu kota pun yang memiliki tujuan ke kota yang sama.

8. Pasangkan kota asal dan kota tujuan

Setiap rute yang dihasilkan oleh langkah ke-7 akan dipecah-pecah menjadi pasangan kota asal dan tujuan. Baris ke-36 akan melakukan iterasi sebanyak elemen dalam **tabuList**. Di setiap iterasi tersebut dilakukan

iterasi juga sebanyak ukuran `tabuList[k]` dikurangi `1` (baris 37). Pada iterasi ini nilai `tabuList[k][l]` dan `tabuList[k][l+1]` akan ditambahkan ke larik `city` (baris 38, 39). Selanjutnya larik `city` akan ditambahkan ke larik `pairedCity` (baris 40). Setelah iterasi baris 37 selesai, maka pasangan kota terakhir (kota akhir dan kota awal) ditambahkan lagi ke larik `pairedCity` (baris 42).

```
1.      def ACOTSProblem(self):
2.          tabuList = []; feromon = 1/len(self.params['matriks'])
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
11.             print(tabuList)
12.
13.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
14.             for i in range(len(matriks)-1):
15.                 for j in range(len(tabuList)):
16.                     r = random.uniform(0,1)
17.                     for cityID in range(len(matriks)):
18.                         for k in tabuList[j]:
19.                             temp.append(k)
20.                             temp.append(cityID)
21.                         if temp.count(cityID) == len(tabuList[j]):
22.                             city.append(tabuList[j][-1])
23.                             city.append(cityID)
24.                         else:
25.                             city.append(cityID)
26.                             city.append(cityID)
27.                         pairedCity.append(city)
28.                         city = []; temp = []
29.                     pairedDistances = self.getDistance(pairedCity)
30.                     pairedCity = []
31.                     probNextCities = self.getProbNextCities(pairedDistances, feromon)
32.                     nextCities = self.getNextCites(probNextCities, r)
33.                     tabuList[j].append(nextCities)
34.             print(tabuList)
35.             print()
36.             for k in range(len(tabuList)):
37.                 for l in range(len(tabuList[k])-1):
38.                     city.append(tabuList[k][l])
39.                     city.append(tabuList[k][l+1])
40.                     pairedCity.append(city)
41.                     city = []
42.                 pairedCity.append([tabuList[k][-1], tabuList[k][0]])
43.                 print(pairedCity)
44.                 pairedCity = []
45.             sys.exit()
```

Hasil eksekusinya adalah:

```
iterasi ke- 0
[[9], [6], [5], [5], [9], [8], [1], [0], [5], [9], [4], [8], [1], [8], [9]]
[[9, 8, 1, 4, 5, 3, 6, 7, 2, 0], [6, 3, 5, 7, 4, 8, 0, 9, 2, 1], [5, 1, 8, 4, 0, 2, 9, 6, 3, 7], [5,
3, 6, 7, 9, 4, 2, 8, 0, 1], [9, 4, 7, 6, 5, 3, 8, 1, 0, 2], [8, 2, 1, 7, 5, 3, 6, 4, 9, 0], [1, 9,
5, 3, 6, 8, 2, 4, 7, 0], [0, 6, 3, 5, 7, 8, 9, 4, 2, 1], [5, 6, 3, 7, 9, 2, 1, 8, 0, 4], [9, 7, 6,
3, 5, 0, 4, 2, 8, 1], [4, 2, 9, 1, 8, 0, 5, 3, 7, 6], [8, 3, 5, 6, 0, 7, 9, 2, 4, 1], [1, 3, 6, 5,
7, 9, 2, 8, 4, 0], [8, 9, 2, 1, 7, 3, 5, 6, 0, 4], [9, 2, 8, 0, 1, 4, 3, 5, 6, 7]]

[[9, 8], [8, 1], [1, 4], [4, 5], [5, 3], [3, 6], [6, 7], [7, 2], [2, 0], [0, 9]]
[[6, 3], [3, 5], [5, 7], [7, 4], [4, 8], [8, 0], [0, 9], [9, 2], [2, 1], [1, 6]]
[[5, 1], [1, 8], [8, 4], [4, 0], [0, 2], [2, 9], [9, 6], [6, 3], [3, 7], [7, 5]]
[[5, 3], [3, 6], [6, 7], [7, 9], [9, 4], [4, 2], [2, 8], [8, 0], [0, 1], [1, 5]]
[[9, 4], [4, 7], [7, 6], [6, 5], [5, 3], [3, 8], [8, 1], [1, 0], [0, 2], [2, 9]]
[[8, 2], [2, 1], [1, 7], [7, 5], [5, 3], [3, 6], [6, 4], [4, 9], [9, 0], [0, 8]]
[[1, 9], [9, 5], [5, 3], [3, 6], [6, 8], [8, 2], [2, 4], [4, 7], [7, 0], [0, 1]]
[[0, 6], [6, 3], [3, 5], [5, 7], [7, 8], [8, 9], [9, 4], [4, 2], [2, 1], [1, 0]]
[[5, 6], [6, 3], [3, 7], [7, 9], [9, 2], [2, 1], [1, 8], [8, 0], [0, 4], [4, 5]]
[[9, 7], [7, 6], [6, 3], [3, 5], [5, 0], [0, 4], [4, 2], [2, 8], [8, 1], [1, 9]]
[[4, 2], [2, 9], [9, 1], [1, 8], [8, 0], [0, 5], [5, 3], [3, 7], [7, 6], [6, 4]]
[[8, 3], [3, 5], [5, 6], [6, 0], [0, 7], [7, 9], [9, 2], [2, 4], [4, 1], [1, 8]]
[[1, 3], [3, 6], [6, 5], [5, 7], [7, 9], [9, 2], [2, 8], [8, 4], [4, 0], [0, 1]]
[[8, 9], [9, 2], [2, 1], [1, 7], [7, 3], [3, 5], [5, 6], [6, 0], [0, 4], [4, 8]]
[[9, 2], [2, 8], [8, 0], [0, 1], [1, 4], [4, 3], [3, 5], [5, 6], [6, 7], [7, 9]]
```

9. Buat method `getPairedCityName`

Method ini akan memasangkan antara ID kota dengan ID nama kota di variabel cityNames seperti setting parameter di langkah ke-1 baris 14.
Method ini melakukan iterasi sebanyak elemen tabuList. Di setiap iterasi tabuList terdapat iterasi sebanyak ukuran cityNames. Pada tiap iterasi ini akan menyeleksi apabila nilai indeks i sama dengan j maka nama kota dari larik cityNames[j] akan dimasukkan ke larik rets.

```
1.      def getPairedCityName(self, tabuList):
2.          rets = []
3.          for i in tabuList:
4.              for j in range(len(self.params['cityNames'])):
5.                  if i == j:
6.                      rets.append(self.params['cityNames'][j])
7.          return rets
```

Selanjutnya kita panggil method `getPairedCityName` di baris 46.

```
1.      def ACOTSProblem(self):
2.          tabuList = []; feromon = 1/len(self.params['matriks'])
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
11.             print(tabuList)
12.
13.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
14.             for i in range(len(matriks)-1):
15.                 for j in range(len(tabuList)):
16.                     r = random.uniform(0,1)
17.                     for cityID in range(len(matriks)):
18.                         for k in tabuList[j]:
19.                             temp.append(k)
20.                             temp.append(cityID)
21.                         if temp.count(cityID) == len(tabuList[j]):
22.                             city.append(tabuList[j][-1])
23.                             city.append(cityID)
24.                         else:
25.                             city.append(cityID)
26.                             city.append(cityID)
27.                         pairedCity.append(city)
28.                         city = []; temp = []
29.                     pairedDistances = self.getDistance(pairedCity)
30.                     pairedCity = []
31.                     probNextCities = self.getProbNextCities(pairedDistances, feromon)
32.                     nextCities = self.getNextCites(probNextCities, r)
33.                     tabuList[j].append(nextCities)
34.             print(tabuList)
35.             print()
36.             for k in range(len(tabuList)):
37.                 for l in range(len(tabuList[k])-1):
38.                     city.append(tabuList[k][l])
39.                     city.append(tabuList[k][l+1])
40.                     pairedCity.append(city)
41.                     city = []
42.                 pairedCity.append([tabuList[k][-1], tabuList[k][0]])
43.                 pairedDistances = self.getDistance(pairedCity)
44.                 print(pairedDistances)
45.                 pairedCity = []
46.                 name = self.getPairedCityName(tabuList[k])
47.                 print(name)
48.                 print(pairedDistances, sum(pairedDistances), 'kilometer')
49.             sys.exit()
```

Hasil eksekusinya adalah seperti berikut:

```
iterasi ke-0
[[1], [5], [4], [2], [1], [2], [2], [0], [0], [5], [9], [0], [6], [2], [9]]
[[1, 9, 5, 7, 3, 6, 8, 4, 2, 0], [5, 3, 7, 4, 2, 9, 8, 0, 6, 1], [4, 9, 2, 0, 5, 3, 7, 6, 1, 8],
[2, 3, 5, 8, 4, 9, 6, 7, 0, 1], [1, 4, 9, 6, 3, 5, 7, 0, 2, 8], [2, 6, 3, 5, 8, 4, 9, 7, 0, 1], [2,
9, 8, 7, 6, 3, 5, 0, 4, 1], [0, 2, 9, 8, 6, 3, 5, 7, 4, 1], [0, 9, 4, 2, 7, 5, 3, 6, 8, 1], [5, 3,
6, 0, 2, 9, 4, 8, 7, 1], [9, 2, 0, 4, 1, 5, 3, 6, 7, 8], [0, 4, 2, 9, 8, 1, 7, 3, 5, 6], [6, 7, 3,
5, 0, 8, 9, 2, 1, 4], [2, 6, 3, 5, 8, 9, 4, 1, 7, 0], [9, 2, 7, 8, 4, 5, 6, 3, 0, 1]]
[318, 244, 56.5, 56.5, 24, 181, 104, 26.1, 185, 360]
```

```
['Sunan Gunung Jati (Cirebon)', 'Sunan Muria (Kudus)', 'Sunan Gresik', 'Sunan Drajat (Lamongan)',
'Sunan Giri (Gresik)', 'Sunan Ampel (Surabaya)', 'Sunan Bonang (Tuban)', 'Sunan Kalijaga (Demak)',
'Sunan Kudus', 'Jogja']
[318, 244, 56.5, 56.5, 24, 181, 104, 26.1, 185, 360] 1555.1 kilometer

[4, 56.5, 225, 26.1, 21.4, 80.9, 163, 334, 583, 601]
['Sunan Gresik', 'Sunan Giri (Gresik)', 'Sunan Drajat (Lamongan)', 'Sunan Kalijaga (Demak)', 'Sunan
Kudus', 'Sunan Muria (Kudus)', 'Sunan Bonang (Tuban)', 'Jogja', 'Sunan Ampel (Surabaya)', 'Sunan
Gunung Jati (Cirebon)']
[4, 56.5, 225, 26.1, 21.4, 80.9, 163, 334, 583, 601] 2094.9 kilometer

[45.8, 21.4, 185, 340, 4, 56.5, 75.9, 583, 370, 104]
['Sunan Kalijaga (Demak)', 'Sunan Muria (Kudus)', 'Sunan Kudus', 'Jogja', 'Sunan Gresik', 'Sunan
Giri (Gresik)', 'Sunan Drajat (Lamongan)', 'Sunan Ampel (Surabaya)', 'Sunan Gunung Jati (Cirebon)',
'Sunan Bonang (Tuban)']
[45.8, 21.4, 185, 340, 4, 56.5, 75.9, 583, 370, 104] 1785.6 kilometer
...
[21.4, 202, 120, 104, 383, 22.5, 24, 335, 360, 318]
['Sunan Muria (Kudus)', 'Sunan Kudus', 'Sunan Drajat (Lamongan)', 'Sunan Bonang (Tuban)', 'Sunan
Kalijaga (Demak)', 'Sunan Gresik', 'Sunan Ampel (Surabaya)', 'Sunan Giri (Gresik)', 'Jogja', 'Sunan
Gunung Jati (Cirebon)']
[21.4, 202, 120, 104, 383, 22.5, 24, 335, 360, 318] 1889.9 kilometer
```

10. Mendapatkan rute terpendek tiap iterasi

Hasil dari langkah ke-9 berupa seluruh rute di satu iterasi. Oleh karena itu selanjutnya kita harus mendapatkan rute terpendek saja di iterasi tersebut.

Pertama, deklarasikan dua variabel baru yaitu finalDistance dan allDistance (baris 2). Pada baris 45, isi larik finalDistance dengan larik yang berisi name dan pairedDistances. Selanjutnya hitung sum(pairedDistances) untuk dimasukkan ke larik allDistances (baris 46).

Kemudian kita dapatkan indeks dari larik allDistances yang memiliki nilai elemen terkecil (baris 49). Selanjutnya kita cetak rute yang terpendek seperti yang ditunjukkan baris ke-50 dan 51.

```
1.      def ACOTSProblem(self):
2.          tabuList = []; feromon = 1/len(self.params['matriks']); finalDistance = [];
    allDistances = []
3.          for iter in range(self.params['maxIter']):
4.              print('iterasi ke-', iter)
5.              for i in range(self.params['antSize']):
6.                  if self.start:
7.                      nextCity = self.start[0]
8.                  else:
9.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
10.                 tabuList.append([nextCity])
11.             print(tabuList)
12.
13.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
14.             for i in range(len(matriks)-1):
15.                 for j in range(len(tabuList)):
16.                     r = random.uniform(0,1)
17.                     for cityID in range(len(matriks)):
18.                         for k in tabuList[j]:
19.                             temp.append(k)
20.                             temp.append(cityID)
21.                         if temp.count(cityID) == len(tabuList[j]):
22.                             city.append(tabuList[j][-1])
23.                             city.append(cityID)
24.                         else:
25.                             city.append(cityID)
26.                             city.append(cityID)
27.                         pairedCity.append(city)
28.                         city = []; temp = []
29.                     pairedDistances = self.getDistance(pairedCity)
30.                     pairedCity = []
31.                     probNextCities = self.getProbNextCities(pairedDistances, feromon)
32.                     nextCities = self.getNextCites(probNextCities, r)
33.                     tabuList[j].append(nextCities)
34.             print(tabuList)
35.             print()
36.             for k in range(len(tabuList)):
```

```
                for l in range(len(tabuList[k])-1):
                    city.append(tabuList[k][l])
                    city.append(tabuList[k][l+1])
                    pairedCity.append(city)
                    city = []
                pairedCity.append([tabuList[k][-1], tabuList[k][0]])
                pairedDistances = self.getDistance(pairedCity)
                name = self.getPairedCityName(tabuList[k])
                finalDistance.append([name, pairedDistances])
                allDistances.append(sum(pairedDistances))
                pairedCity = []
            print(allDistances)
            minIndex = allDistances.index(min(allDistances))
            print(finalDistance[minIndex])
            print(min(allDistances), 'kilometer')
            finalDistance = []; allDistances = []; tabuList = []
            print()
```

Hasil eksekusinya menjadi:

```
iterasi ke- 0
[[6], [6], [4], [4], [3], [0], [1], [4], [1], [3], [2], [4], [3], [0], [0]]
[[6, 3, 7, 2, 9, 4, 5, 0, 1, 8], [6, 5, 3, 7, 9, 2, 0, 1, 4, 8], [4, 2, 8, 9, 7, 3, 5, 6, 1, 0], [4,
9, 2, 7, 3, 5, 0, 8, 1, 6], [3, 5, 6, 7, 2, 8, 0, 9, 4, 1], [0, 2, 8, 4, 7, 6, 5, 3, 9, 1], [1, 0,
8, 2, 9, 4, 5, 7, 3, 6], [4, 9, 2, 6, 0, 5, 3, 7, 8, 1], [1, 3, 5, 6, 7, 9, 2, 8, 0, 4], [3, 5, 7,
6, 1, 0, 2, 4, 9, 8], [2, 8, 6, 5, 3, 7, 9, 4, 0, 1], [4, 6, 2, 9, 3, 5, 7, 8, 0, 1], [3, 0, 9, 2,
8, 6, 5, 7, 4, 1], [0, 8, 7, 5, 3, 4, 9, 2, 1, 6], [0, 2, 1, 4, 5, 3, 8, 7, 6, 9]]
[1983.7, 1403.4, 1573.6, 2167.7, 1645.8, 1616.0, 1773.8, 1969.7, 1575.4, 1816.1000000000001, 1403.4,
2025.9, 1974.0, 1933.6999999999998, 2033.9]
[['Sunan Ampel (Surabaya)', 'Sunan Gresik', 'Sunan Giri (Gresik)', 'Sunan Drajat (Lamongan)', 'Sunan
Muria (Kudus)', 'Sunan Kudus', 'Jogja', 'Sunan Gunung Jati (Cirebon)', 'Sunan Kalijaga (Demak)',
'Sunan Bonang (Tuban)'], [22.5, 4, 56.5, 200, 21.4, 185, 360, 269, 104, 181]]
1403.4 kilometer

iterasi ke- 1
[[3], [6], [0], [3], [1], [6], [0], [5], [6], [7], [8], [2], [0], [8], [3]]
[[3, 8, 0, 4, 2, 9, 7, 6, 5, 1], [6, 1, 9, 2, 8, 5, 3, 0, 4, 7], [0, 9, 3, 5, 7, 6, 2, 1, 4, 8], [3,
5, 6, 8, 7, 0, 9, 4, 2, 1], [1, 4, 9, 6, 3, 5, 7, 2, 8, 0], [6, 5, 3, 0, 1, 7, 8, 4, 2, 9], [0, 2,
9, 4, 8, 7, 6, 3, 5, 1], [5, 3, 6, 7, 9, 2, 0, 4, 1, 8], [6, 8, 7, 4, 9, 3, 5, 2, 0, 1], [7, 8, 6,
5, 3, 9, 2, 0, 4, 1], [8, 5, 3, 7, 2, 9, 4, 6, 0, 1], [2, 9, 3, 8, 0, 6, 5, 4, 1, 7], [0, 5, 6, 7,
3, 9, 4, 2, 1, 8], [8, 9, 7, 3, 5, 6, 4, 2, 1, 0], [3, 5, 6, 0, 8, 4, 2, 9, 7, 1]]
[2012.9, 1966.9, 1819.4, 1837.4, 1540.9, 1939.0, 1541.1, 1473.3, 2352.8, 1813.9, 1939.7, 2409.9,
1633.8, 1588.0, 2064.0]
[['Sunan Gresik', 'Sunan Giri (Gresik)', 'Sunan Ampel (Surabaya)', 'Sunan Drajat (Lamongan)', 'Sunan
Muria (Kudus)', 'Sunan Kudus', 'Jogja', 'Sunan Kalijaga (Demak)', 'Sunan Gunung Jati (Cirebon)',
'Sunan Bonang (Tuban)'], [4, 24, 75.9, 200, 21.4, 185, 160, 269, 370, 164]]
1473.3 kilometer

iterasi ke- 2
[[8], [0], [5], [4], [6], [8], [7], [8], [0], [8], [1], [6], [5], [1], [2]]
[[8, 2, 9, 0, 3, 5, 6, 7, 4, 1], [0, 7, 4, 8, 9, 2, 6, 3, 5, 1], [5, 3, 8, 0, 4, 2, 9, 7, 6, 1], [4,
8, 7, 5, 3, 6, 9, 2, 0, 1], [6, 5, 2, 9, 0, 1, 8, 4, 7, 3], [8, 5, 3, 6, 0, 9, 2, 7, 1, 4], [7, 5,
3, 4, 2, 8, 9, 6, 0, 1], [8, 9, 7, 5, 3, 6, 2, 4, 0, 1], [0, 4, 7, 5, 6, 3, 8, 2, 1, 9], [8, 9, 4,
6, 5, 3, 2, 0, 1, 7], [1, 4, 2, 8, 0, 5, 6, 3, 7, 9], [6, 3, 7, 4, 9, 2, 5, 0, 8, 1], [5, 7, 6, 3,
8, 2, 9, 0, 4, 1], [1, 9, 4, 2, 7, 6, 3, 5, 0, 8], [2, 0, 1, 4, 9, 8, 6, 5, 3, 7]]
[1607.4, 2191.3, 1998.4, 1479.9, 1795.4, 1936.4, 2201.1, 1925.4, 1547.6, 2215.2, 1499.7, 2236.7,
1656.4, 1568.8000000000002, 1406.6999999999998]
[['Sunan Kudus', 'Jogja', 'Sunan Gunung Jati (Cirebon)', 'Sunan Kalijaga (Demak)', 'Sunan Muria
(Kudus)', 'Sunan Bonang (Tuban)', 'Sunan Ampel (Surabaya)', 'Sunan Gresik', 'Sunan Giri (Gresik)',
'Sunan Drajat (Lamongan)'], [185, 360, 269, 45.8, 80.9, 181, 22.5, 4, 56.5, 202]]
1406.6999999999998 kilometer

. . .

iterasi ke- 24
[[7], [7], [3], [1], [5], [8], [4], [9], [2], [5], [4], [1], [1], [2], [0]]
[[7, 8, 6, 3, 5, 4, 2, 9, 0, 1], [7, 5, 3, 6, 9, 8, 2, 0, 1, 4], [3, 5, 6, 7, 4, 9, 2, 0, 8, 1], [1,
2, 9, 8, 4, 7, 3, 5, 6, 0], [5, 3, 6, 8, 7, 2, 9, 4, 1, 0], [8, 2, 7, 3, 5, 6, 1, 9, 4, 0], [4, 8,
2, 9, 3, 5, 7, 6, 0, 1], [9, 0, 7, 5, 3, 6, 2, 1, 4, 8], [2, 9, 4, 0, 3, 5, 7, 6, 8, 1], [5, 3, 4,
2, 9, 8, 7, 6, 0, 1], [4, 3, 5, 6, 8, 0, 9, 2, 7, 1], [1, 8, 6, 3, 0, 4, 9, 7, 5, 2], [1, 2, 8, 4,
9, 7, 3, 6, 5, 0], [2, 9, 8, 7, 5, 6, 3, 1, 4, 0], [0, 8, 7, 6, 3, 5, 4, 2, 9, 1]]
[1933.5, 1621.0, 1691.6, 1501.3, 1567.1999999999998, 1635.3999999999999, 1546.4, 1806.4, 1542.6,
1936.3, 1989.9, 2073.3, 1526.4, 1518.3, 1495.4]
```

```
[['Jogja', 'Sunan Bonang (Tuban)', 'Sunan Drajat (Lamongan)', 'Sunan Ampel (Surabaya)', 'Sunan Giri (Gresik)', 'Sunan Gresik', 'Sunan Kalijaga (Demak)', 'Sunan Kudus', 'Sunan Muria (Kudus)', 'Sunan Gunung Jati (Cirebon)'], [163, 120, 75.9, 24, 4, 383, 26.1, 21.4, 318, 360]]
1495.4 kilometer
```

11. Dapatkan solusi terbaik

Berdasarkan rute terpendek yang diperoleh di tiap iterasi dari langkah ke-10, maka selanjutnya kita akan pilih rute terpendek final yang menjadi solusi terbaik dari seluruh rute terpendek di tiap iterasi.
Deklarasikan dua variabel baru yaitu `finalResults` dan `bestSolutions` (baris 2). Isi larik `finalResults` dengan larik `finalDistances[minIndex]` (baris 45). Isi juga larik `bestSolutions` dengan nilai minimal dari larik `allDistances` (baris 46).

```
1.      def ACOTSProblem(self):
2.          tabuList = []; feromon = 1/len(self.params['matriks']); finalDistance = []; allDistances = []; finalResults = []; bestSolutions = []
3.          for iter in range(self.params['maxIter']):
4.              for i in range(self.params['antSize']):
5.                  if self.start:
6.                      nextCity = self.start[0]
7.                  else:
8.                      nextCity = random.randint(0, len(self.params['matriks'])-1)
9.                  tabuList.append([nextCity])
10.
11.             temp = []; city = []; pairedCity = []; matriks = self.params['matriks']
12.             for i in range(len(matriks)-1):
13.                 for j in range(len(tabuList)):
14.                     r = random.uniform(0,1)
15.                     for cityID in range(len(matriks)):
16.                         for k in tabuList[j]:
17.                             temp.append(k)
18.                             temp.append(cityID)
19.                         if temp.count(cityID) == len(tabuList[j]):
20.                             city.append(tabuList[j][-1])
21.                             city.append(cityID)
22.                         else:
23.                             city.append(cityID)
24.                             city.append(cityID)
25.                         pairedCity.append(city)
26.                         city = []; temp = []
27.                     pairedDistances = self.getDistance(pairedCity)
28.                     pairedCity = []
29.                     probNextCities = self.getProbNextCities(pairedDistances, feromon)
30.                     nextCities = self.getNextCites(probNextCities, r)
31.                     tabuList[j].append(nextCities)
32.             for k in range(len(tabuList)):
33.                 for l in range(len(tabuList[k])-1):
34.                     city.append(tabuList[k][l])
35.                     city.append(tabuList[k][l+1])
36.                     pairedCity.append(city)
37.                     city = []
38.                 pairedCity.append([tabuList[k][-1], tabuList[k][0]])
39.                 pairedDistances = self.getDistance(pairedCity)
40.                 name = self.getPairedCityName(tabuList[k])
41.                 finalDistance.append([name, pairedDistances])
42.                 allDistances.append(sum(pairedDistances))
43.                 pairedCity = []
44.             minIndex = allDistances.index(min(allDistances))
45.             finalResults.append(finalDistance[minIndex])
46.             bestSolutions.append(min(allDistances))
47.             finalDistance = []; allDistances = []; tabuList = []
48.         shortestRoutes = finalResults[bestSolutions.index(min(bestSolutions))]
49.         print('Rute terpendek ziarah makam wali songo:')
50.         for i in shortestRoutes[0]:
51.             print(i)
52.         print(shortestRoutes[0][0])
53.         print(sum(shortestRoutes[1]), 'kilometer')
```

Apabila variabel `shortestRoutes` dicetak, maka outputnya seperti ini:

```
[['Sunan Gunung Jati (Cirebon)', 'Sunan Kalijaga (Demak)', 'Sunan Kudus', 'Sunan Muria (Kudus)', 'Sunan Bonang (Tuban)', 'Sunan Drajat (Lamongan)', 'Sunan Gresik', 'Sunan Giri (Gresik)', 'Sunan Ampel (Surabaya)', 'Jogja'], [269, 26.1, 21.4, 80.9, 120, 56.5, 4, 24, 334, 360]]
```

Sedangkan hasil ahir eksekusinya adalah:

```
Rute terpendek ziarah makam wali songo:
Jogja
Sunan Gunung Jati (Cirebon)
Sunan Kalijaga (Demak)
Sunan Kudus
Sunan Muria (Kudus)
Sunan Bonang (Tuban)
Sunan Drajat (Lamongan)
Sunan Giri (Gresik)
Sunan Gresik
Sunan Ampel (Surabaya)
Jogja
1294.4 kilometer
```

1.7. POST TEST

Jawablah pertanyaan berikut (**Total Skor: 100**):

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1.	CPL-03	CPMK-07	Carilah rute terpendek dari tempat tinggal Anda saat ini di Jogja ke keenam kampus UAD hingga kembali ke tempat tinggal Anda. Gantilah variabel matriks pada langkah 1 dengan pasangan jarak antara tempat tinggal Anda dengan keenam kampus UAD. Gunakan Google Maps untuk mendapatkan jaraknya. Adapun peta keenam kampus UAD bisa dilihat di Peta Kampus - Universitas Ahmad Dahlan (uad.ac.id)	100

1.8. HASIL CAPAIAN PRAKTIKUM

Diisi oleh asisten setelah semua assessment dinilai.

No	Bentuk Assessment	CPMK	Sub-CPMK	Bobot	Skor (0-100)	Nilai Akhir (Bobot x Skor)
1.	Pre-Test	03	07	20%		
2.	Praktik			50%		
3.	Post-Test			30%		
					Total Nilai	

PRAKTIKUM 5: ALGORITMA GENETIKA

Pertemuan ke : 6 dan 7

Total Alokasi Waktu : 90 menit
- Materi : 15 menit
- Pre-Test : 15 menit
- Praktikum : 45 menit
- Post-Test : 15 menit

Total Bobot Penilaian : 100%
- Pre-Test : 20 %
- Praktik : 50 %
- Post-Test : 30 %

Pemenuhan CPL dan CPMK:

CPMK-03	Mampu merancang dan mengimplementasikan algoritma/metode optimasi dalam mengidentifikasi dan memecahkan masalah yang melibatkan perangkat lunak dan pemikiran komputasi
Sub-CPMK-06	Mampu menyelesaikan permasalahan optimasi metaheuristik menggunakan algoritme berbasis evolusioner

1.1. DESKRIPSI CAPAIAN PEMBELAJARAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu membuat program optimasi metaheuristik berbasis evolusioner

1.2. INDIKATOR KETERCAPAIAN PEMBELAJARAN

Indikator ketercapaian diukur dengan:

CPMK-03	Sub-CPMK-06	Ketepatan hasil implementasi algoritme evolusioner menggunakan bahasa pemrograman sesuai masalah optimasi yang diberikan

1.3. TEORI PENDUKUNG

Pada saat John Holland mengusulkan Genetic Algorithm (GA) atau Algoritme Genetika di tahun 1975 masih sangat sederhana, sehingga sering disebut sebagai GA klasik atau *canonical* GA. Pada awal GA klasik ditemukan, kebanyakan digunakan untuk menyelasaikan permasalahan optimasi diskrit yang memiliki ciri khas penemuan solusi optimum yang tidak terlalu cepat. Namun di balik kelemahannya itu GA klasik memiliki heuristik yang baik untuk masalah kombinatorial.
Secara umum, algoritme GA klasik adalah sebagai berikut:

```
Bangkitkan populasi awal
Hitung fitness tiap kromosom
Lakukan seleksi orang tua
ulangi
       lakukan rekombinasi dengan kawin silang
       lakukan mutasi
       hitung fitness tiap individu
       lakukan seleksi orang tua
sampai kriteria berhenti terpenuhi
```

Pada praktikum ini problem yang dipecahkan adalah masalah mencari nilai minimum dari fungsi *sum square* yang diberikan sebagai berikut.

$$f(x) = \sum_{i=1}^{N} i \times {x_i}^2$$

Dengan syarat dan batasan

$$-10 \leq x \leq 10$$
$$N = 3$$

1.4. HARDWARE DAN SOFTWARE

Hardware dan software yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Python, C/C++, PHP atau C#, dan lain sebagainya yang sudah dikuasai sebelumnya

1.5. PRE-TEST

Jawablah pertanyaan berikut (**Total Skor: 100**):

No	CPMK	Sub-CPMK	Pertanyaan	Skor

1.	03	06	Bangkitkanlah 10 bilangan acak tipe *real* antara 1-10 yang disimpan dalam array.
			Contoh outputnya:
			`[2.12 3.00 1.9 4.5 5.81 7.04 2.0 5.5 1.11 2.45]`
			
			Selanjutnya, pilihlah salah satu bilangan dalam array tersebut secara acak. Kemudian ubahlah nilai terpilih tersebut dengan nilai acak *real* juga antara 1-10.
			Misalnya bilangan terpilih adalah indeks ke-3 yaitu 4.5
			Nilai acak yang dibangkitkan adalah 2.8.
			Sehingga array terbaru setelah salah satu elemen diganti menjadi:
			`[2.12 3.00 1.9 2.8 5.81 7.04 2.0 5.5 1.11 2.45]`
				100

1.6. LANGKAH PRAKTIKUM

Aturan Penilaian (Total Skor: 100):

No	CPMK	Sub-CPMK	Pertanyaan	Dokumen Pendukung	Skor
1.	03	06	Bangkitkan populasi awal	Hasil praktikum langkah 1	10
2.			Hitung nilai fitness tiap individu	Hasil praktikum langkah 2	10
3.			Lakukan seleksi orang tua	Hasil praktikum langkah 3	10
4.			Lakukan rekombinasi kawin silang di tiap generasi	Hasil praktikum langkah 4	25
5.			Lakukan mutasi di tiap generasi	Hasil praktikum langkah 5	15
6.			Hitung nilai fitness tiap individu di tiap generasi	Hasil praktikum langkah 6	10
7.			Lakukan seleksi orang tua di tiap generasi	Hasil praktikum langkah 7	10
8.			Tampilkan solusi optimum	Hasil praktikum langkah 8	10

Langkah-Langkah Praktikum:

```
1.  from operator import index
2.  import random, sys
3.  from collections import deque
4.
5.  class GeneticAlgorithm:
6.      def __init__(self, parameters):
7.          self.numOfChromosome = parameters['numOfChromosome']
8.          self.numOfDimension = parameters['numOfDimension']
9.          self.lowerBound = parameters['lowerBound']
10.         self.upperBound = parameters['upperBound']
11.         self.numOfDimension = parameters['numOfDimension']
```

```
12.         self.cr = parameters['crossoverRate']
13.         self.mr = parameters['mutationRate']
14.         self.maxGeneration = parameters['maxGen']
15.         self.stoppingFitness = parameters['stoppingFitness']
16.
17.     def sumSquaresTestFunction(self, chromosome):
18.         sumResult = 0
19.         for i in range(len(chromosome)):
20.             sumResult = sumResult + ((i+1) * pow(chromosome[i],2))
21.         return abs(sumResult)
22.
23.     def calcFitnessValue(self, objectiveValue):
24.         return 1 / (1 + objectiveValue)
25.
26.     def replaceChromosomesElement(self, chromosomes, chromosome, index):
27.         chromosomes[index] = chromosome
28.         return chromosomes
29.
30.     def selectCandidateChromosomes(self, probCummulatives, chromosomes):
31.         rets = []
32.         for i in range(len(probCummulatives)):
33.             # Langkah 1. Bangkitkan nilai acak [0,1]
34.             #=====================================
35.             randomValue = random.uniform(0,1)
36.             # Langkah 2. Bandingkan nilai acak dengan nilai probabilitas kumulatif ke-i
37.             #=========================================================================
38.             if randomValue > probCummulatives[i] and randomValue <= probCummulatives[i+1]:
39.                 rets.append({'index':i,'chromosome':chromosomes[i+1]})
40.         return rets
41.
42.     def selectRoletteWheelChromosome(self, fitnessValues, chromosomes):
43.         probCummulative = 0; probCummulatives = []
44.         for fitnessValue in fitnessValues:
45.             # Langkah 1. Hitung Probabilitas Tiap Fitness Value Kromosom
46.             #===========================================================
47.             probability = fitnessValue / sum(fitnessValues)
48.             # Langkah 2 Probabilitas Kumulatif Tiap Fitness Value Kromosom
49.             #===========================================================
50.             probCummulative = probCummulative + probability
51.             probCummulatives.append(probCummulative)
52.         # print(probabilities)
53.             #Contoh Output: [0.24, 0.30, 0.63, 0.84, 1.0]
54.
55.         # Langkah 3. Tentukan Index kromosom yang akan diganti
56.         #=====================================================
57.         selectedCandidateChromosomes = self.selectCandidateChromosomes(probCummulatives,
    chromosomes)
58.         # print(selectCandidateChromosomes)
59.             #Contoh Output 1: [] #Contoh Output 2: [{'index': 3, 'chromosome': [1.08, -3.8, 5.09]}]
60.             #Artinya ada kemungkinan tidak ada elemen. Sehingga harus diulangi hingga mendapatkan
    minimal 1 elemen
61.         while len(selectedCandidateChromosomes) == 0:
62.            selectedCandidateChromosomes = self.selectCandidateChromosomes(probCummulatives,
    chromosomes)
63.             # Pasti mendapatkan index
64.         return selectedCandidateChromosomes
65.
66.     def generateRandomValues(self):
67.         rets = []
68.         for i in range(self.numOfChromosome):
69.             if random.uniform(0,1) < self.cr:
70.                 rets.append(i)
71.         return rets
72.
73.     def mainGA(self):
74.         # Langkah 1 Buat populasi awal
75.         #============================
```

```
76.         varValues = []; chromosomes = [];
77.         for _ in range(self.numOfChromosome):
78.             for _ in range(self.numOfDimension):
79.                 randomValue = random.uniform(self.lowerBound, self.upperBound)
80.                 varValues.append(randomValue)
81.             chromosomes.append(varValues)
82.             varValues = []
83.         #print(chromosomes)
84.             #Contoh Output: [[-3.4, 5.4, 2.4], [-7.8, -5.4, -1.1], [7.3, -4.6, -2.07], [-7.6, 6.2,
    -1.7], [-7.6, 4.8, 0.9]]
85.
86.         # Langkah 2 Hitung Fungsi Objektif Tiap Chromosome
87.         #=================================================
88.         objectiveValues = []
89.         for chromosome in chromosomes:
90.             objectiveValues.append(self.sumSquaresTestFunction(chromosome))
91.         # print(objectiveValues)
92.             #Contoh Output: [1.11, 17.6, 9.6, 10.3, 15.03]
93.
94.         # Langkah 3 Hitung Nilai Fitness Tiap Objective Value
95.         #=====================================================
96.         fitnessValues = []
97.         for objectiveValue in objectiveValues:
98.             fitnessValues.append(self.calcFitnessValue(objectiveValue))
99.         #print(fitnessValues)
100.            #Contoh Output: [0.16, 0.07, 0.28, 0.04, 0.17]
101.
102.        # Langkah 4 Seleksi Chromosomes
103.        #==============================
104.        candidateNewChromosomes = self.selectRoletteWheelChromosome(fitnessValues, chromosomes)
105.
106.        # Langkah 5. Buat populasi baru
107.        #==============================
108.        for candidateNewChromosome in candidateNewChromosomes:
109.            chromosomes = self.replaceChromosomesElement(chromosomes,
    candidateNewChromosome['chromosome'], candidateNewChromosome['index'])
110.        # print(chromosomes)
111.            # Pasti berbeda dengan kromosom sebelumnya (baris 72)
112.
113.        for k in range(self.maxGeneration):
114.            print('Generation-', k)
115.            # Langkah 6. Crossover (kawin silang)
116.            #====================================
117.            # Langkah 6.1. Bangkitkan nilai acak dan ambil indeksnya
118.            randomIndexValues = self.generateRandomValues()
119.            #print(randomIndexValues)
120.                #Contoh Output: [], [1], atau [1,3]. Artinya bisa kosong maupun berisi index
    sebanyak 1 atau lebih.  Padahal yang dibutuhkan minimum 2 index agar terpentung sepasang orang tua.
    Oleh karena itu harus diulangi agar minimal berisi index
121.            while len(randomIndexValues) <= 1:
122.                randomIndexValues = self.generateRandomValues()
123.            # print(randomIndexValues)
124.                #Contoh Output: pasti minimal mengandung 2 elemen index
125.
126.            # Langkah 6.2. Ambil nilai kromosom berdasarkan index random
127.            selectedChromosomesToCrossover = []
128.            for i in randomIndexValues:
129.                selectedChromosomesToCrossover.append({'chromosomes':chromosomes[i], 'index':i})
130.            # print(selectedChromosomesToCrossover)
131.                #Contoh Output: [{'chromosomes': [1.1, -5.4, -9.9], 'index': 0}, {'chromosomes':
    [1.1, -5.4, -9.9], 'index': 1}, {'chromosomes': [5.9, 1.5, -0.7], 'index': 3}]
132.
133.            # Langkah 6.3. Bangkitkan pasangan index parent untuk dicrossover
134.            parentCandidatesIndex = []
135.            for i in selectedChromosomesToCrossover:
136.                for j in selectedChromosomesToCrossover:
137.                    if i['index'] != j['index']:
```

```
                        parentCandidatesIndex.append([i['index'],j['index']])
            #print(parentCandidatesIndex)
            #Contoh Output: [[0, 1], [0, 2], [0, 3], [1, 0], [1, 2], [1, 3], [2, 0], [2, 1], [2, 3], [3, 0], [3, 1], [3, 2]].
            # Yang perlu diambil pasangan yang unik saja. Tidak boleh sama. Contoh: [0,1] sama dengan [1,0]. Jadi cukup diambil [0,1] saja. Sehingga perlu difilter.

            # Langkah 6.4. Membentuk pasangan index hingga menjadi unik (tidak redundan)
            # Langkah 6.4.1 Mengurutkan semua pasangan
            sortedParentIndexes = []
            for parentIndex in parentCandidatesIndex:
                parentIndex.sort()
                sortedParentIndexes.append(parentIndex)
            # print(sortedParentIndexes)
                #Contoh Output: [[0, 1], [0, 2], [0, 1], [1, 2], [0, 2], [1, 2]]
            # Langkah 6.4.2. Memfilter pasangan indeks
            finalParentIndexes = []
            for sortedParentIndex in sortedParentIndexes:
                if sortedParentIndex not in finalParentIndexes:
                    finalParentIndexes.append(sortedParentIndex)
            #print(finalParentIndexes)
                #Contoh Output: [[0, 1], [0, 2], [0, 3], [1, 2], [1, 3], [2, 3]]

            # Langkah 6.5 Membuat Offsets
            tempOffsets = []; offsets = []
            for parentsIndex in finalParentIndexes:
                # Langkah 6.5.1. Bangkitkan nilai integer acak sebagai titik potong
                cutPointIndex = random.randint(0, self.numOfDimension-1)
                # Langkah 6.5.2 Buat kromosom anak (offset) hasil kawin silang parents
                # Jika cutpointIndex == numOfDimension
                if cutPointIndex == self.numOfDimension-1:
                    for i in range(self.numOfDimension):
                        if i < self.numOfDimension-1:
                            tempOffsets.append(chromosomes[parentsIndex[1]][i])
                        else:
                            tempOffsets.append(chromosomes[parentsIndex[0]][cutPointIndex])
                else:
                    for i in range(self.numOfDimension):
                        if i <= cutPointIndex:
                            tempOffsets.append(chromosomes[parentsIndex[0]][i])
                        else:
                            tempOffsets.append(chromosomes[parentsIndex[1]][i])
                # print(chromosomes[parentsIndex[0]],' X ', chromosomes[parentsIndex[1]])
                # print(tempOffsets)
                #Contoh Output: [-9.1, 4.4, -9.5]  X  [-1.2, 0.4, 5.05] Hasil kromosom baru: [-9.1, 0.4, 5.05]
                offsets.append(tempOffsets)
                tempOffsets = []
            # print(offsets)
            # Contoh Output: [[4.4, -1.8, 3.5], [4.1, 9.1, 3.5], [4.4, -1.8, 3.3]]

            # Langkah 7. Mutasi
            #===================
            tempChromosomes = []
            # Langkah 7.1 Buat populasi gabungan chromosomes dan offsets
            chromosomesOffsets = chromosomes + offsets
            # Langkah 7.2 Hitung nilai objektif dan fitness populasi gabungan
            for chromosome in chromosomesOffsets:
                objectiveValue = self.sumSquaresTestFunction(chromosome)
                fitnessValue = self.calcFitnessValue(objectiveValue)
                tempChromosomes.append([fitnessValue, chromosome])
            #print(tempChromosomes)
            #Contoh Output: Indeks-0 nilai fitness, indeks-1 kromosom [[0.07, [9.02, -5.3, -3.1]], [0.07, [-7.3, 1.8, -7.5]], [0.07, [-7.3, 1.8, -7.5]], [0.09, [8.5, 5.9, -7.8]], [0.08, [7.5, 7.9, 1.6]], [0.09, [9.02, 5.9, -7.8]]]
            #print()

```

```
            # Langkah 7.3 Urutkan populasi secara menurun berdasarkan nilai value
            tempChromosomes.sort(reverse=True)
            #print(tempChromosomes)
            #Contoh Output: terurut [[0.07, [-7.3, 1.8, -7.5]], [0.07, [-7.3, 1.8, -7.5]], [0.07, [9.02, -5.3, -3.1]], [0.08, [7.5, 7.9, 1.6]], [0.09, [8.5, 5.9, -7.8]], [0.09, [9.02, 5.9, -7.8]]]
            chromosomes = []
            # Langkah 7.4 Buat populasi baru
            for i in range(len(tempChromosomes)):
                # Langkah 7.5 Memastikan jumlah kromosom sesuai ukuran populasi
                if i <= self.numOfChromosome-1:
                    chromosomes.append(tempChromosomes[i][1])
            tempChromosomes = []
            # Langkah 7.6 Hitung jumlah mutasi
            numOfMutation = round(self.mr * (self.numOfChromosome * self.numOfDimension))
            # Langkah 7.7 Pilih kromosom dan gen dalam kromosom secara acak
            # print('BEFORE')
            # print(chromosomes)
            #Contoh Output: BEFORE [[-6.08, -7.6, 4.5], [-6.08, -7.6, 4.5], [-3.9, -8.7, 8.7], [-6.08, -7.6, 8.7], [-7.6, -3.6, 8.8]]
            # print()
            for i in range(numOfMutation):
                # Langkah 7.8 Pilih index kromosom secara acak
                selectedChromosomeIndex = random.randint(0, self.numOfChromosome-1)
                # Langkah 7.9 Pilih index gen kromosom secara acak
                selectedGenIndex = random.randint(0, self.numOfDimension-1)
                mutatedChromosome = chromosomes[selectedChromosomeIndex]
                # Langkah 7.10 Ganti gen kromosom termutasi dengan nilai acak sesuai rentang variabel desain
                mutatedChromosome[selectedGenIndex] = random.uniform(self.lowerBound, self.upperBound)
                # Langkah 7.11 Update kromosom termutasi dengan kromosom yang gennya telah dimutasi
                chromosomes[selectedChromosomeIndex] = mutatedChromosome
            # print(chromosomes)
            #Contoh Output: Berbeda dengan Before [[1.6, -7.6, 4.5], [1.6, -7.6, 4.5], [0.9, -8.7, 8.7], [-6.08, -7.6, 8.7], [-7.6, -3.6, 8.8]]

            # Langkah 8 Hitung nilai objektif dan fitness populasi baru
            #============================================================
            for chromosome in chromosomes:
                objectiveValue = self.sumSquaresTestFunction(chromosome)
                fitnessValue = self.calcFitnessValue(objectiveValue)
                tempChromosomes.append([fitnessValue, chromosome, objectiveValue])
            # Langkah 8.1 Dapatkan kromosom dengan nilai fitness tertinggi, yang otomatis nilai objektif terendah
            bestChromosome = max(tempChromosomes)
            print(bestChromosome)
            #Contoh Output: Berhenti di generasi ke-72.
            # Generation- 0 [0.051, [3.57, 0.92, 1.16], 18.5] Generation- 1 [0.051, [3.57, 0.92, 1.16], 18.57] Generation- 2 [0.05, [3.57, 0.92, 1.16], 18.5] ... Generation- 72 [0.97, [-0.08, 0.02, 0.08], 0.028]
            if self.stoppingFitness <= bestChromosome[0]:
                break
            tempChromosomes = []

parameters = {
    'numOfChromosome': 30,
    'lowerBound': -10,
    'upperBound': 10,
    'numOfDimension': 3,
    'crossoverRate': 0.25,
    'mutationRate': 0.1,
    'maxGen': 80,
    'stoppingFitness': 0.95
}

runGA = GeneticAlgorithm(parameters)
runGA.mainGA()
```

1.7. POST TEST

Jawablah pertanyaan berikut (**Total Skor: 100**):

No	CPMK	Sub-CPMK	Pertanyaan	Skor
1.	CPL-03	CPMK-06	Ubahlah parameter jumlah koromosom, generasi maksimal dan kriteria berhenti fitness. Lakukan berulang-ulang, lalu berikan penjelasan dan uraian mengenai pengetahuan, pengalaman kesimpulan apa yang Anda dapatkan.	100

1.8. HASIL CAPAIAN PRAKTIKUM

Diisi oleh asisten setelah semua assessment dinilai.

No	Bentuk Assessment	CPMK	Sub-CPMK	Bobot	Skor (0-100)	Nilai Akhir (Bobot x Skor)
1.	Pre-Test	03	07	20%		
2.	Praktik			50%		
3.	Post-Test			30%		
					Total Nilai	

LEMBAR JAWABAN PRE-TEST DAN POST-TEST PRAKTIKUM

Nama : **NIM :**	**Asisten:** **Paraf Asisten:**	**Tanggal:** **Nilai:**

DAFTAR PUSTAKA

1.


LABORATORIUM
TEKNIK INFORMATIKA
UNIVERSITAS AHMAD DAHLAN


UNIVERSITAS AHMAD DAHLAN