Contents list avaliable at Directory of Open Access Journals (DOAJ)

JUTIN : Jurnal Teknik Industri Terintegrasi

Volume 6 Issue 4 2023, Page 1560-1569

ISSN: 2620-8962 (Online)

Journal Homepage: https://journal.universitaspahlawan.ac.id/index.php/jutin/index

Genetic Algorithm untuk Menyelesaikan *Russia-20-Nodes-TSP Instance*

Ekra Sanggala✉

Program Studi D4 Logistik Bisnis, Sekolah Vokasi, Universitas Logistik & Bisnis Internasional, Bandung

DOI: 10.31004/jutin.v6i4.22118

✉ Corresponding author:
[ekrasanggala@mail.ru]

Article Info

Kata kunci:
Genetic Algorithm;
TSP;
Metaheuristic;
Heuristic;
Instance

Abstrak

Travelling Salesman Problem (*TSP*) merupakan permasalahan penentuan rute terpendek yang diawali dari titik *start* untuk mengunjungi sekumpulan titik tepat sekali dan diakhiri dengan kembali ke titik *start*. *Genetic Algorithm* (*GA*) merupakan sebuah *metaheuristic* yang dapat diaplikasikan pada berbagai permasalahan optimasi, termasuk *TSP*. Untuk membuktikan kemampuan tersebut maka diperlukan sebuah *TSP Instance* yang akan diselesaikan oleh *Genetic Algorithm*. *Russia-20-Nodes-TSP Instance* merupakan salah satu *TSP Instance* yang terdapat pada *Russian TSP Instances*. Dengan menggunakan *Genetic Algorithm*, panjang rute terpendek dari *Russia-20-Nodes-TSP Instance* adalah 10.104 Km.

Keywords:
Genetic Algorithm;
TSP;
Metaheuristic;
Heuristic;
Instance

Abstract

Travelling Salesman Problem (TSP) is the problem for finding the shortest route starting from start node then visiting number of nodes exactly once and finally go back to start node. Genetic Algorithm is a metaheuristic that can be applied to various optimization problems, include TSP. To prove this capability, a TSP Instance is needed which will be solved by Genetic Algorithm. Russia-20-Nodes-TSP Instance is one of the TSP Instances found in Russian TSP Instances. Using Genetic Algorithm, the shortest route length of the Russia-20-Nodes-TSP Instance is 10.104 Km.

1. INTRODUCTION

Travelling Salesman Problem (*TSP*) merupakan permasalahan penentuan rute terpendek yang diawali dari titik *start* untuk mengunjungi sekumpulan titik tepat sekali dan diakhiri dengan kembali ke titik *start*. Pada dunia nyata banyak permasalahan yang dapat didefinisikan sebagai *TSP*, diantaranya adalah rute perjalanan turis, rute bus sekolah, rute pengiriman barang dan lain-lain. Jika sebuah *TSP* mempunyai banyak titik, maka akan menjadi sebuah *NP-Hard Problem* dimana untuk memperoleh solusi terbaiknya diperlukan waktu perhitungan yang tidak wajar untuk ditunggu (Wu, 2020). Algoritma yang bekerja berdasarkan *heuristic* dan

Received 23 October 2023; Received in revised form 8 November 2023 year; Accepted 16 November 2023
Available online 17 November 2023 /

metaheuristic dapat menjadi sebuah solusi untuk menyelesaikan *NP-Hard Problem* dengan waktu perhitungan yang wajar untuk ditunggu, walaupun solusi yang dihasilkan belum tentu merupakan solusi yang terbaik (Elshaer & Awad, 2020).

Genetic Algorithm (*GA*) merupakan proses pencarian dengan pendekatan *heuristic* yang dapat digunakan pada berbagai macam permasalahan optimasi. Fleksibilitas *Genetic Algorithm* membuat algoritma ini banyak digunakan untuk menyelesaikan masalah-masalah optimasi. Evolusi merupakan dasar dari *Genetic Algorithm*. Keberagaman spesies saat ini dan kesuksesan mereka untuk bertahan hidup merupakan sebuah alasan baik untuk mempercayai kekuatan dari evolusi. Spesies-spesies mempunyai kemampuan untuk beradaptasi dengan lingkungan mereka. Mereka berkembang menjadi struktur yang komplek yang memungkinkan mereka bertahan hidup di berbagai macam lingkungan. Perkawinan dan menghasilkan keturunan merupakan prinsip utama dari suksesnya proses evolusi. Hal-hal ini merupakan alasan yang baik untuk mengadopsi prinsip-prinsip evolusi pada penyelesaian masalah-masalah optimasi (Katoch et al., 2021).

Secara umum tahapan-tahapan yang terdapat pada *Genetic Algorithm* adalah sebagai berikut ini (Katoch et al., 2021):

1. *Setting Parameters*
2. *Initialize Population*
3. *Crossover*
4. *Mutation*
5. *Genotype-Phenotype Mapping*
6. *Fitness*
7. *Selection*
8. *Termination*

Dengan adanya fleksibilitas *Genetic Algorithm* untuk menyelesaikan berbagai permasalahan optimasi, tentunya *Genetic Algorithm* dapat juga digunakan untuk menyelesaikan *TSP*. Kemampuan *Genetic Algorithm* untuk menyelesaikan *TSP* perlu diuji, agar dapat dibuktikan kemampuannya dalam menyelesaikan *TSP*. Untuk membuktikan kemampuan tersebut, maka diperlukan sebuah *TSP Instance* yang akan diselesaikan oleh *Genetic Algorithm*.

Sebuah *TSP Instance* yang dibuat oleh penulis adalah ***Russia-20-Nodes-TSP Instance***. *TSP Instance* ini dibuat berdasarkan 20 kota berpenduduk terbanyak di *Russia* (Sanggala & Bisma, 2023). *TSP Instance* ini menjadi menarik untuk diselesaikan karena jumlah titiknya tidak terlalu banyak, hanya 20 titik saja. Dengan jumlah titik sebanyak 20 titik, menjadikan *TSP Instance* tersebut sebagai permasalahan yang menarik karena waktu perhitungannya yang relatif cepat dan mempunyai kemungkinan solusi yang sangat banyak. Dengan demikian pada *paper* ini akan dibahas mengenai *Genetic Algorithm* untuk menyelesaikan *Russia-20-Nodes-TSP Instance*.

2. METHODS

Langkah-langkah dalam penelitian ini adalah sebagai berikut ini:

1. Penentuan *TSP Instance*
   TSP Instance yang diselesaikan adalah *Russia-20-Nodes-TSP Instance*.

2. Menghitung Jarak Antara Setiap Titik Pada *Russia-20-Nodes-TSP Instance*
   Untuk menghitung jarak antara setiap titik pada *Russia-20-Nodes-TSP Instance* akan digunakan *Haversine* Formula.

3. Pengembangan *Genetic Algorithm*
   Pada tahap ini dikembangkan sebuah *Genetic Algorithm* yang sesuai dengan permasalahan *TSP*.

4. Menulis *Program* Dalam *GNU Octave*
   Genetic Algorithm yang telah dikembangkan diterjemahkan ke dalam *Program* yang ditulis di dalam *GNU Octave*. Dengan demikian penyelesaian *Russia-20-Nodes-TSP Instance* dapat dilakukan dengan bantuan komputer.

5. Menjalankan *Program*
   Program yang telah selesai ditulis di dalam *GNU Octave* dijalankan agar penyelesaian *Russia-20-Nodes-TSP Instance* dapat dilakukan. *Program* ini akan menghasilkan solusi berupa rute urutan kunjungan dan panjang rutenya.

3. RESULT AND DISCUSSION

Berikut ini merupakan hasil penelitian berdasarkan langkah-langkah pada bagian *methods*.

A. *Russia-20-Nodes-TSP Instance*

Instance merupakan sekumpulan data yang akan diselesaikan pada sebuah penelitian. *Instance* yang akan digunakan pada penelitian ini adalah *Russia-20-Nodes TSP Instance*. Pada tabel 1 dapat dilihat data yang terdapat pada *instance* tersebut.

Tabel 1: *Russia-20-Nodes TSP Instance*

Node_No	Name	Latitude	Longitude
1	Moscow	55,75222	37,61556
2	Saint Petersburg	59,93863	30,31413
3	Novosibirsk	55,04150	82,93460
4	Yekaterinburg	56,85190	60,61220
5	Nizhniy Novgorod	56,32867	44,00205
6	Samara	53,20007	50,15000
7	Omsk	54,99244	73,36859
8	Kazan	55,78874	49,12214
9	Rostov-na-Donu	47,23135	39,72328
10	Chelyabinsk	55,15402	61,42915
11	Ufa	54,74306	55,96779
12	Volgograd	48,71939	44,50183
13	Perm	58,01046	56,25017
14	Krasnoyarsk	56,01839	92,86717
15	Saratov	51,54056	46,00861
16	Voronezh	51,67204	39,18430
17	Tol'yatti	53,53030	49,34610
18	Krasnodar	45,04484	38,97603
19	Ulyanovsk	54,32824	48,38657
20	Izhevsk	56,84976	53,20448

B. Jarak Antara Setiap Titik Pada *Russia-20-Nodes-TSP Instance*

Untuk menghitung jarak antara setiap titik ini digunakan *Haversine* formula yang bentuk persamaannya seperti berikut ini (Idrizi, 2020):

$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 . \cos\phi_2 \ . sin^2\left(\frac{\Delta\lambda}{2}\right)$$
$$c = 2 \, . atan2\left[\sqrt{a}, \sqrt{(1-a)}\right]$$
$$d = R \, . c$$

Φ1 : *Latitude* titik pertama
Φ2 : *Latitude* titik kedua
λ1 : *Longitude* titik pertama
λ2 : *Longitude* titik kedua

ΔΦ : Φ1-Φ2
Δλ : λ1-λ2
R : Radius Bumi (6371 Km)

Berikut ini merupakan contoh penggunaan *Haversine* formula dalam menghitung jarak antara dua titik di Bumi:
Moscow (*Latitude*: 55,75222; *Longitude*: 37,61556)
Saint Petersburg (*Latitude*: 55,75222; *Longitude*: 37,61556)

$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \ . \ \cos\phi_2 \ . \ sin^2\left(\frac{\Delta\lambda}{2}\right)$$
$$a = 0{,}002477$$

$$c = 2\,.\,atan2\left[\sqrt{0{,}002477}, \sqrt{0{,}998761}\,\right]$$
$$c = 0{,}099581$$

$$d = 6371\,.\,0{,}099581$$
$$d = 634{,}4309 \approx 635\ km$$

Tabel 2: Jarak Antara Setiap Titik Pada *Russia-20-Nodes TSP Instance*

No	Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	Moscow	0	635	2812	1418	402	857	2236	719	959	1496	1166	911	1156	3353	724	466	794	1195	704	968
2	Saint Petersburg	635	0	3105	1783	896	1420	2584	1200	1540	1912	1632	1544	1492	3575	1350	1072	1355	1754	1251	1370
3	Novosibirsk	2812	3105	0	1398	2412	2127	610	2115	3083	1363	1715	2693	1658	634	2456	2880	2166	3267	2200	1848
4	Yekaterinburg	1418	1783	1398	0	1017	780	820	718	1773	196	374	1406	292	1967	1116	1499	804	1990	817	451
5	Nizhniy Novgorod	402	896	2412	1017	0	526	1834	324	1054	1096	773	847	761	2961	549	606	462	1303	356	566
6	Samara	857	1420	2127	780	526	0	1520	296	994	765	418	636	658	2725	337	762	65	1216	171	451
7	Omsk	2236	2584	610	820	1834	1520	0	1527	2474	760	1111	2084	1100	1230	1847	2277	1561	2660	1600	1269
8	Kazan	719	1200	2115	718	324	296	1527	0	1151	778	449	847	499	2681	515	797	252	1392	170	278
9	Rostov-na-Donu	959	1540	3083	1773	1054	994	2474	1151	0	1740	1406	393	1629	3697	661	496	977	250	996	1408
10	Chelyabinsk	1496	1912	1363	196	1096	765	760	778	1740	0	352	1359	449	1961	1097	1518	803	1943	841	545
11	Ufa	1166	1632	1715	374	773	418	1111	449	1406	352	0	1034	364	2308	753	1166	452	1619	492	291
12	Volgograd	911	1544	2693	1406	847	636	2084	847	393	1359	1034	0	1291	3309	332	501	633	586	679	1076
13	Perm	1156	1492	1658	292	761	658	1100	499	1629	449	364	1291	0	2201	973	1295	659	1862	636	224
14	Krasnoyarsk	3353	3575	634	1967	2961	2725	1230	2681	3697	1961	2308	3309	2201	0	3058	3465	2759	3888	2782	2406
15	Saratov	724	1350	2456	1116	549	337	1847	515	661	1097	753	332	973	3058	0	472	317	890	349	753
16	Voronezh	466	1072	2880	1499	606	762	2277	797	496	1518	1166	501	1295	3465	472	0	716	738	683	1075
17	Tol'yatti	794	1355	2166	804	462	65	1561	252	977	803	452	633	659	2759	317	716	0	1205	109	443
18	Krasnodar	1195	1754	3267	1990	1303	1216	2660	1392	250	1943	1619	586	1862	3888	890	738	1205	0	1233	1642
19	Ulyanovsk	704	1251	2200	817	356	171	1600	170	996	841	492	679	636	2782	349	683	109	1233	0	413
20	Izhevsk	968	1370	1848	451	566	451	1269	278	1408	545	291	1076	224	2406	753	1075	443	1642	413	0

C. Pengembangan *Genetic Algorithm*

Genetic Algorithm yang dikembangkan bisa berbeda-beda walaupun masalah yang akan diselesaikan merupakan masalah yang sama. Perbedaan ini dikarenakan setiap orang mempunyai cara dan kreatifitas masing-masing dalam mengembangkan sebuah *Genetic Algorithm*. *Genetic Algorithm* untuk menyelesaikan *TSP* yang dikembangkan pada penelitian ini, sangat mungkin berbeda dengan *Genetic Algorithm* yang dikembangkan pada penelitian lainnya, walaupun tujuannya sama yaitu untuk menyelesaikan *TSP*.

1. *Setting Parameters*
   Parameter-parameter *Genetic Algorithm* yang digunakan pada penelitian ini adalah sebagai berikut:
   a) *Crossover Probability*
      Crossover Probability merupakan parameter yang menyatakan seberapa besar peluang sepasang orang tua akan melakukan perkawinan silang (Alam et al., 2020). Rentang nilai untuk parameter ini adalah dari 0 sampai dengan 1. Nilai yang akan digunakan sekarang adalah 0,5.

   b) *Mutation Probability*
      Mutation Probability merupakan parameter yang menyatakan seberapa besar peluang suatu individu akan mengalami mutasi (Alam et al., 2020). Rentang nilai untuk parameter ini adalah dari 0 sampai dengan 1. Nilai yang akan digunakan sekarang adalah 0,5.

   c) *Generation*
      Generation merupakan parameter yang menyatakan banyaknya generasi yang akan dihasilkan oleh *Genetic Algorithm* (Alam et al., 2020). Batasan nilai untuk parameter ini adalah semua bilangan positif. Nilai yang akan digunakan sekarang adalah 1000.

   d) *Individual*
      Individual merupakan parameter yang menyatakan banyaknya individu yang akan hidup pada setiap generasi (Alam et al., 2020). Batasan nilai untuk parameter ini adalah semua bilangan positif. Nilai yang akan digunakan sekarang adalah 50.

   e) *Survivor/Elite*
      Survivor/Elite merupakan parameter yang menyatakan banyaknya individu terbaik pada setiap generasi yang akan tetap hidup pada generasi berikutnya (Alam et al., 2020). Rentang nilai untuk parameter ini adalah 0 sampai dengan nilai parameter *individual* dikurangi 1. Nilai yang akan digunakan sekarang adalah 10.

2. *Initialize Population*
   Merupakan pembentukan individu-individu yang akan menjadi populasi awal atau populasi yang hidup di generasi 0 (Alam et al., 2020). Karena jumlah titik pada *Russia-20-Nodes-TSP Instance* adalah 20 titik, maka setiap individu akan terdiri dari 21 gen. Gen 1 dan Gen 21 akan berisi titik 1 (*Moscow*) karena titik tersebut akan menjadi titik awal dan titik kembali dari rute yang terbentuk. Gen 2 sampai dengan Gen 20 akan terisi oleh 19 titik lainnya. Cara *random* akan digunakan untuk memilih titik-titik yang akan mengisi Gen 2 sampai dengan Gen 20 pada individu-individu di populasi awal ini.

	Gen																				
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Individual 1	1	11	7	3	6	5	13	16	9	17	2	14	15	8	4	18	19	20	10	12	1
Individual 2	1	19	2	18	5	4	17	6	15	10	8	13	3	7	12	11	14	9	16	20	1

Gambar 1: Contoh Individu Dari Cara *Random*

3. *Crossover*
   Crossover merupakan perkawinan silang yang dilakukan oleh sepasang orang tua (Alam et al., 2020). Cara *crossover* yang akan digunakan kali ini adalah *one-point crossover*. Cara ini akan memilih secara *random* satu posisi Gen yang akan menjadi posisi persilangan antara sepasang orang tua (Alam et al., 2020). Pada *Genetic Algorithm* kali ini, setiap proses *crossover* akan menghasilkan dua anak. Dua anak yang dihasilkan dari *crossover* ini seringkali tidak memiliki Gen yang lengkap, dan Gen yang hilang pada satu anak akan menjadi Gen ganda pada saudaranya. Oleh karena itu satu anak akan

melakukan pertukaran Gen ganda dengan saudaranya untuk melengkapi Gen yang hilang. Proses pertukaran Gen ganda yang akan digunakan sekaran adalah dengan memas angkan setiap Gen ganda berdasarkan nomor Gen ganda terkecil dengan nomor posisi Gen terbesar.

One-Point Crossover	9

	Gen																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Parent 1	1	11	2	7	16	14	19	5	9	4	3	10	15	6	12	13	8	18	20	17	1
Parent 2	1	18	16	15	12	4	5	14	20	13	19	6	8	17	9	3	2	10	11	7	1

	Gen																					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	**Lost Gen**
Offspring 1	1	11	2	7	16	14	19	5	9	13	19	6	8	17	9	3	2	10	11	7	1	4, 12, 15,18 & 20
Offspring 2	1	18	16	15	12	4	5	14	20	4	3	10	15	6	12	13	8	18	20	17	1	2, 7, 9, 11 & 19

	Gen																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Offspring 1	1	11	2	7	16	14	19	5	9	13	20	6	8	17	15	3	4	10	18	12	1
Offspring 2	1	18	16	15	12	4	5	14	20	2	3	10	9	6	7	13	8	11	19	17	1

Gambar 2: Contoh *One-Point Crossover*

4. *Mutation*

Mutation merupakan perubahan Gen yang terjadi pada suatu individu (Alam et al., 2020). Pada *Genetic Algorithm* kali ini, mutasi akan terjadi pada anak-anak yang dihasilkan dari *crossover*. Untuk kali ini cara mutasinya adalah dengan memilih sepasang Gen secara *random* untuk bertukar tempat.

	Gen																					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	**Mutation Position**
Offspring 1	1	11	2	7	16	14	19	5	9	13	20	6	8	17	15	3	4	10	18	12	1	7 & 18
Offspring 2	1	18	16	15	12	4	5	14	20	2	3	10	9	6	7	13	8	11	19	17	1	3 & 15

	Gen																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Offspring 1	1	11	2	7	16	14	10	5	9	13	20	6	8	17	15	3	4	19	18	12	1
Offspring 2	1	18	7	15	12	4	5	14	20	2	3	10	9	6	16	13	8	11	19	17	1

Gambar 3: Contoh *Mutation*

5. *Genotype-Phenotype Mapping*
   Genotype adalah susunan Gen pada individu yang belum menjadi solusi untuk menyelesaikan permasalahan yang akan diselesaikan (Alam et al., 2020). *Genotype* ini harus diterjemahkan menjadi *Phenotype* agar dapat menjadi sebuah solusi (Alam et al., 2020). Dikarenakan untuk kali ini susunan Gen pada individu sudah dapat langsung menjadi solusi, maka tidak perlu dilakukan penerjemahan dari *Genotype* ke *Phenotype*.

6. *Fitness*
   Fitness adalah sebuah nilai yang menyatakan seberapa baik kualitas suatu individu (Alam et al., 2020). Pada *Genetic Algorithm* kali ini, semakin baik nilai *fitness* suatu individu maka semakin besar kemungkinannya individu tersebut akan terpilih untuk menjadi orang tua pada proses *crossover*. Untuk kali ini persamaan untuk menghitung nilai *fitness* adalah sebagai berikut:

   $$F_i = \frac{1}{L_i}$$

   F_i : Nilai *fitness* individu ke-i
   L_i : Panjang rute individu ke-i

	Gen																						
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**Length (Km)**	**Fitness**
Individual 1	1	11	7	3	6	5	13	16	9	17	2	14	15	8	4	18	19	20	10	12	1	24741	0,0000404187
Individual 2	1	19	2	18	5	4	17	6	15	10	8	13	3	7	12	11	14	9	16	20	1	23539	0,0000424827
Individual 3	1	11	2	7	16	14	10	5	9	13	20	6	8	17	15	3	4	19	18	12	1	25805	0,0000387522
Individual 4	1	18	7	15	12	4	5	14	20	2	3	10	9	6	16	13	8	11	19	17	1	26796	0,0000373190
Individual 5	1	20	18	16	14	12	10	8	6	4	2	3	5	7	9	11	13	15	17	19	1	28816	0,0000347029

Gambar 4: Contoh Nilai *Fitness*

7. *Selection*
   Selection adalah pemilihan sejumlah individu untuk digunakan pada proses selanjutnya (Alam et al., 2020). Dikarenakan pada *Genetic Algorithm* kali ini jumlah individu pada setiap generasi adalah 50 individu dan jumlah individu yang menjadi *survivor/elite* adalah 10 individu, maka diperlukan 40 individu lagi yang akan hidup pada generasi selanjutnya. Pada *Genetic Algorithm* kali ini, proses pemilihan individu yang akan hidup pada generasi selanjutnya adalah dengan memilih semua anak yang dihasilkan pada setiap generasi. Oleh karena itu dikarenakan setiap *crossover* akan menghasilkan 2 anak, maka untuk menghasilkan 40 anak diperlukan 20 pasang orang tua atau individu. Proses pemilihan 40 individu untuk menjadi orang tua ini dilakukan dengan cara *Roulette Wheel*, dimana individu yang memiliki nilai *fitness* lebih baik akan memiliki peluang lebih besar untuk terpilih (Alam et al., 2020).

8. *Termination*
   Termination adalah terhentinya proses evolusi (Alam et al., 2020). Pada *Genetic Algorithm* kali ini, *termination* akan terjadi jika proses evolusi sudah mencapai 1000 generasi.

D. Menulis *Program* Dalam *GNU Octave*

Untuk membantu menyelesaikan seluruh perhitungan digunakan perangkat lunak *GNU Octave*. *GNU Octave* merupakan alat multifungsi untuk melakukan analisis numerik. Berbagai alat yang tersedia di dalam *GNU Octave* adalah sebagai berikut ini (Ramos, 2020):

1. Sekumpulan *function* untuk menyelesaikan berbagai masalah.
2. Bahasa pemrograman yang dapat digunakan untuk mengembangkan kemampuan *GNU Octave*.
3. Berbagai fasilitas untuk melakukan *plotting*.

Nama *GNU Octave* diambil dari seorang ahli kimia yang bernama Octave Levenspiel. *GNU Octave* ini merupakan proyek resmi dari *GNU* dan lisensi *source code* berada di bawah *GNU General Public License* (*GPL*), sehingga siapa pun boleh menggunakannya untuk berbagai tujuan (Nagar & Nagar, 2018).

Kode *program* yang ditulis ada kurang lebih 586 baris. Jika kode tersebut ditampilkan pada *paper* ini tentu akan menghabiskan tempat, oleh karena itu bagi pembaca yang membutuhkan kodenya dapat langsung menghubungi penulis melalui surat elektronik.

E. Menjalankan *Program*

Setelah *program* selesai ditulis, maka program tersebut dijalankan agar *Genetic Algorithm* dapat menyelesaikan *Russia-20-Nodes-TSP Instance*. Dikarenakan *Genetic Algorithm* bekerja menggunakan bilangan *random*, tentunya hasil dari setiap perhitungan dapat berbeda-beda, oleh karena itu *program* akan dijalankan sebanyak 10 kali untuk kemudian dari 10 solusi yang dihasilkan diambil satu solusi terbaiknya.

Tabel 3: Panjang Rute Dari 10 Kali Perhitungan

Perhitungan	Panjang Rute (Km)	Perhitungan	Panjang Rute (Km)
1	10529	6	11591
2	10822	7	11950
3	11208	8	10104
4	10529	9	11191
5	11237	10	10961

Pada tabel 3 dapat dilihat bahwa dari 10 kali perhitungan yang dilakukan, diperoleh panjang rute terpendek adalah 10.104 Km yang berasal dari perhitungan ke-8. Dengan demikian solusi yang dihasilkan dari perhitungan ke-8 terpilih untuk menjadi solusi dari *Russia-20-Nodes-TSP Instance*.

Tabel 4: Rute Solusi Dari *Russia-20-Nodes-TSP Instance*

Rute	Name	Jarak (Km)
0	Moscow	
15	Voronezh	466
8	Rostov-na-Donu	496
17	Krasnodar	250
11	Volgograd	586
14	Saratov	332
16	Tol'yatti	317
5	Samara	65
10	Ufa	418
9	Chelyabinsk	352
6	Omsk	760
2	Novosibirsk	610
13	Krasnoyarsk	634
3	Yekaterinburg	1967
12	Perm	292
19	Izhevsk	224
7	Kazan	278
18	Ulyanovsk	170
4	Nizhniy Novgorod	356
1	Saint Petersburg	896
0	Moscow	635

Panjang Rute	**10104**

4. CONCLUSION

Genetic Algorithm dapat digunakan untuk menyelesaikan berbagai permasalahan optimasi, salah satunya adalah *TSP*. *Genetic Algorithm* merupakan salah satu *metaheuristic*, oleh karena itu tidak dapat langsung digunakan untuk menyelesaikan permasalahan optimasi, tetapi perlu dikembangkan terlebih dahulu sesuai dengan permasalahan yang akan diselesaikan.

Agar dapat diketahui kehandalan *Genetic Algorithm* dalam menyelesaikan *TSP*, perlu dilakukan perbandingan dengan algoritma lainnya, seperti *Savings Algorithm*, *Nearest Neighbour*, *Nearest Insertion* dan lain-lain, dengan demikian dapat diketahui kehandalan *Genetic Algorithm* dibandingkan dengan algoritma lainnya.

Dikarenakan *Genetic Algorithm* bekerja berdasarkan *heuristic* tentunya solusi yang dihasilkan belum tentu solusi yang terbaik. Untuk meningkatkan kehandalan *Genetic Algorithm* dapat dilakukan dengan melakukan *hybrid* antara *Genetic Algorithm* dengan berbagai *heuristic*, seperti *Savings Algorithm*, *Nearest Neighbour, Nearest Insertion* dan lain-lain.

5. REFERENCES

Alam, T., Qamar, S., Dixit, A., & Benaida, M. (2020). Genetic algorithm: Reviews, implementations, and applications. *ArXiv Preprint ArXiv:2007.12673*.

Elshaer, R., & Awad, H. (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, *140*, 106242.

Idrizi, B. (2020). Necessity for geometric corrections of distances in web and mobile maps. *International Conference on Cartography and GIS, Bulgaria*.

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, *80*, 8091–8126.

Nagar, S., & Nagar, S. (2018). *Introduction to Octave*. Springer.

Ramos, V. S. (2020). SIHR: a MATLAB/GNU Octave toolbox for single image highlight removal. *Journal of Open Source Software*, *5*(45), 1822.

Sanggala, E., & Bisma, M. A. (2023). Perbandingan Savings Algorithm dengan Nearest Neighbour dalam Menyelesaikan Russian TSP Instances. *Jurnal Media Teknik & Sistem Industri*, *7*(1).

Wu, Z. (2020). A Comparative Study of solving Traveling Salesman Problem with Genetic Algorithm, Ant Colony Algorithm, and Particle Swarm Optimization. *Proceedings of the 2020 2nd International Conference on Robotics Systems and Vehicle Technology*, 95–99.