Jurnal Media Teknik dan Sistem Industri

Jurnal Media Teknik & Sistem Industri
Vol. 7 (no. 1) (2023) hal. 27 – 38
http://jurnal.unsur.ac.id/JMTSI

e-issn: 2581-0561
p-issn: 2581-0529
DOI 10.35194/jmtsi.v7i1.3039

Perbandingan *Savings Algorithm* dengan *Nearest Neighbour* dalam Menyelesaikan *Russian TSP Instances*

Ekra Sanggala[1*], Muhammad Ardhya Bisma[2]

[1,2]*Universitas Logistik & Bisnis Internasional*
Jl. Sariasih No.54 Bandung

[1*]ekrasanggala@mail.ru

[2]bisma@ulbi.ac.id

Comparison Savings Algorithm and Nearest Neighbour on Solving Russian TSP Instances

Dikirimkan: 02, 2023. Diterima: 03, 2023. Dipublikasikan: 03, 2023.

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. Several heuristics are popular for solving TSP, for example Savings Algorithm and Nearest Neighbour. Performance heuristics on solving TSP are diverse, so there is need of reference for choosing a heuristic. Comparing heuristics on solving instance can be a reference for choosing a heuristic. This paper will discuss about comparison Savings Algorithm and Nearest Neighbour on Solving Russian TSP Instances. For generating length of route, Savings Algorithm is better than Nearest Neighbour, while for generating CPU time, Nearest Neighbour is better than Savings Algorithm.*
Keywords— *TSP; Heuristic; Savings Algorithm; Nearest Neighbour; Russian TSP Instances*

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*. Beberapa *Heuristik* yang cukup populer untuk menyelesaikan *TSP* antara lain *Savings Algorithm* dan *Nearest Neighbour*. Kemampuan *Heuristik* dalam menyelesaikan *TSP* berbeda-beda, sehingga diperlukan sebuah acuan untuk menentukan *Heuristik* yang akan digunakan. Membandingkan *Heuristik* dalam menyelesaikan *instance* dapat menjadi acuan untuk pemilihan *Heuristik*. Pada *paper* ini akan dibahas mengenai perbandingan *Savings Algorithm* dan *Nearest Neighbour* dalam menyelesaikan *Russian TSP Instances*. Untuk panjang rute yang dihasilkan, maka *Savings Algorithm* lebih baik dibandingkan *Nearest Neighbour*, sedangkan untuk *CPU Time* yang dihasilkan, maka *Nearest Neighbour* lebih baik dibandingkan *Savings Algorithm*.
Kata kunci— TSP; *Heuristik*; *Savings Algorithm*; *Nearest Neighbour*; *Russian TSP Instances*

I. PENDAHULUAN

Sebuah perusahaan agar dapat menjadi kompetitif harus mendesain, memproduksi dan mendistribusikan produk dan jasanya dengan mempertimbangkan waktu pengiriman dan peningkatan kualitas [1]. Penentuan rute pengiriman produk dan jasa merupakan suatu hal yang sangat penting karena sangat membantu dalam penghematan biaya pengiriman dan penghematan waktu pengiriman [2]. *Travelling Salesman Problem* merupakan salah satu masalah penentuan rute yang paling penting dalam logistik [3].

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*. *TSP* ini termasuk ke dalam *Non-Deterministic Polynomial-Time Hard Problems* (*NP-Hard Problems*), dimana jika titik yang harus dikunjungi berjumlah banyak, maka

untuk mendapatkan rute terpendeknya diperlukan waktu yang sangat lama. Sebagai contoh jika terdapat 50 titik yang harus dikunjungi maka akan terdapat 50! (3,04140932017134 x 10^{64}) alternatif rute, dengan alternatif rute sebanyak ini tentunya akan memerlukan waktu yang sangat lama untuk mendapatkan rute terbaiknya, sekalipun perhitungannya menggunakan super komputer tercanggih [4].

Pada beberapa dekade terakhir ini, pada manajemen pendistribusian produk, telah terjadi peningkatan penggunaan metode-metode optimasi berdasarkan teknik-teknik *Operations Research* dan *Mathematical Programming*. Pada aplikasi dunia nyata, di Amerika Utara dan Eropa penggunaan metode-metode optimasi ini telah berhasil menghemat biaya transportasi sebanyak 5% sampai dengan 20%. Salah satu faktor keberhasilan ini adalah karena ditemukannya algoritma-algoritma yang mampu memberikan solusi yang baik dengan lama waktu perhitungan yang wajar [5].

Algoritma yang perhitungannya cepat walaupun "hanya" menghasilkan solusi yang mendekati optimal, merupakan opsi yang sangat baik untuk menyelesaikan *TSP* [6]. Algoritma jenis ini dikenal dengan sebutan *Heuristik*. "*Heuristik*" merupakan sebuah kata dari bahasa Yunani yang berarti "menemukan" atau "mengeksplorasi". *Heuristik* disebut sebagai *approximate techniques*. Tujuan utama dari *Heuristik* adalah untuk membangun sebuah model optimasi yang mudah dipahami dan mampu memberikan solusi yang baik dalam waktu perhitungan yang wajar [7].

Beberapa *Heuristik* yang cukup populer untuk menyelesaikan *TSP* antara lain [8]:

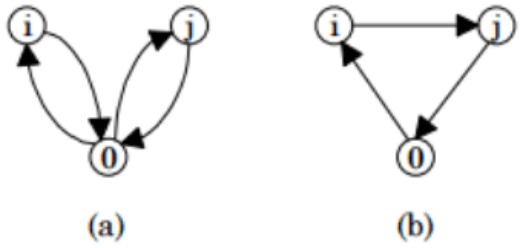
1. *Savings Algorithm*
2. *Nearest Neighbour*
3. *Nearest Insertion*
4. *Christofides Heuristics*
5. *Minimum Spanning Tree Heuristic*

Dikarenakan prinsip dasar kerja dari *Heuristik* adalah "mengeksplorasi", maka kemampuan setiap *Heuristik* dalam menyelesaikan *TSP* berbeda-beda, suatu *Heuristik* mungkin baik dalam menyelesaikan *TSP X* tapi mungkin kurang baik dalam menyelesaikan *TSP Y*, sedangkan *Heuristik* lainnya mungkin kurang baik dalam menyelesaikan *TSP X* tapi mungkin baik dalam menyelesaikan *TSP Y*. Dengan demikian tidak dapat ditentukan *Heuristik* mana yang merupakan *Heuristik* yang terbaik, yang dapat dilakukan adalah menguji *Heuristik* pada beberapa *TSP*, sehingga hasil pengujiannya dapat dijadikan acuan dalam memilih *Heuristik* untuk menyelesaikan *TSP*.

Savings Algorithm dan *Nearest Neighbour* merupakan *Heuristik* yang populer untuk menyelesaikan *TSP*. Pada *paper* ini akan dibahas mengenai perbandingan kedua *Heuristik* tersebut dalam menyelesaikan *Russian TSP Instances*. Dua kriteria penting yang dinilai adalah solusi yang dihasilkan dan waktu perhitungan (*CPU Time*) [9].

II. METODOLOGI PENELITIAN

Savings Algorithm ditemukan oleh Clarke dan Wright pada tahun 1964, prinsip dasar dari *Savings Algorithm* adalah penghematan (*Savings*) jarak tempuh yang diperoleh berdasarkan penggabungan dua rute, gambaran mengenai penggabungan tersebut dapat dilihat pada gambar 1, di mana titik 0 merupakan depot [10].



Gambar 1: Contoh Penggabungan 2 rute [10].

Pada gambar 1(a) terlihat bahwa titik i dan titik j dilayani oleh rute yang berbeda. Pada gambar 1(b) diperlihatkan bahwa titik i dan titik j dilayani oleh rute yang sama. Dapat diketahui jarak tempuh D_a pada gambar 1(a) adalah sebagai berikut ini [11]:

$$D_a = d_{0i} + d_{i0} + d_{0j} + d_{j0} \qquad (1)$$

Sedangkan untuk jarak tempuh D_b pada gambar 1(b) dapat diketahui sebagai berikut ini [11]:

$$D_b = d_{0i} + d_{ij} + d_{j0} \qquad (2)$$

Nilai *Savings* diperoleh dengan mengurangkan jarak tempuh D_a dengan jarak tempuh D_b, sehingga diperoleh persamaan nilai *Savings* S_{ij} berikut ini [11]:

$$S_{ij} = D_a - D_b = d_{i0} + d_{0j} - d_{ij} \qquad (3)$$

Pada *Savings Algorithm*, langkah pertama yang harus dilakukan adalah menghitung *Savings* untuk setiap pasangan titik. Kemudian nilai-nilai *Savings* yang telah diperoleh, diurutkan mulai dari *Savings* terbesar sampai ke *Savings* terkecil. Pembentukan rute dilakukan dengan memasukkan ke dalam rute terlebih dahulu pasangan-pasangan titik yang memiliki nilai *Savings* lebih tinggi [11].

Nearest Neighbour mungkin merupakan *Heuristik* yang paling sederhana dan paling mudah. Prinsip dasar dari *Nearest Neighbour* adalah mengunjungi titik terdekat yang belum dikunjungi. Langkah-langkah dalam *Nearest Neighbour* kurang lebih seperti berikut ini [6]:

1. Tentukan titik *Start*.

2. Kunjungi titik terdekat yang belum dikunjungi.
3. Apa masih ada titik yang belum terkunjungi? Jika masih ada, ulangi ke langkah kedua.
4. Kembali ke titik *Start*.

Russian TSP Instances merupakan sepuluh *TSP* yang dibuat oleh penulis berdasarkan sejarah *Russia* dan tempat-tempat di *Russia*. Diharapkan *TSP Instances* ini dapat menjadi "tempat" para peneliti untuk menjalankan algoritma yang digunakannya. Sepuluh *TSP* yang terdapat di *Russian TSP Instances* adalah sebagai berikut ini:

1. AK-47-TSP
   Memperingati senjata AK-47 yang ditemukan oleh *Mikhail Kalashnikov*.
2. Gagarin-108-TSP
   Memperingati *Yuri Gagarin* sebagai manusia pertama yang melakukan perjalanan ke luar angkasa dan mengorbit selamat 108 menit.
3. Mendeleev-101-TSP
   Memperingati *Dmitri Mendeleev* sebagai penemu sistem periodik. Unsur dengan nomor atom 101 diberi nama *Mendelevium* untuk mengenang *Mendeleev*.
4. Petersburg-182-TSP
   Terdiri dari 182 area berpenduduk di wilayah *Saint Petersburg*.
5. Popov-250-TSP
   Memperingati *Alexander Popov* sebagai penemu radio. Pada tanggal 24 Maret 1896, dia berhasil mentransmisikan sinyal radio sejauh 250 meter.
6. Russia-10-Nodes-TSP
   Terdiri dari 10 kota berpenduduk terbanyak di *Russia*.
7. Russia-20-Nodes-TSP
   Terdiri dari 20 kota berpenduduk terbanyak di *Russia*.
8. Siege-of-Leningrad-872-TSP
   Memperingati 872 hari blokade *Leningrad* saat perang dunia kedua.
9. World-Cup-Stadium-12-TSP
   Memperingati *World Cup 2018* di *Russia*. Sebanyak 12 stadion digunakan pada *World Cup* tersebut.
10. Yashin-270-TSP
    Memperingati *Lev Yashin* sebagai kiper terhebat dalam sejarah sepakbola. Dia berhasil mendapatkan lebih dari 270 *clean sheets* dalam karirnya.

Data koordinat dan jumlah penduduk untuk membuat *Russian TSP Instances* bersumber dari https://www.geonames.org [12]. Pada *Russian TSP Instances*, untuk menghitung jarak antar titik digunakan *Haversine Formula*, yang hasilnya dibulatkan keatas sampai nilai satuan terdekat. Bagi para pembaca yang membutuhkan *file* dari *Russian TSP Instances* dapat langsung menghubungi penulis melalui *e-mail*.

Haversine Formula merupakan persamaan yang digunakan untuk menghitung jarak antara dua titik di Bumi, bentuk persamaannya adalah seperti berikut ini [13]:

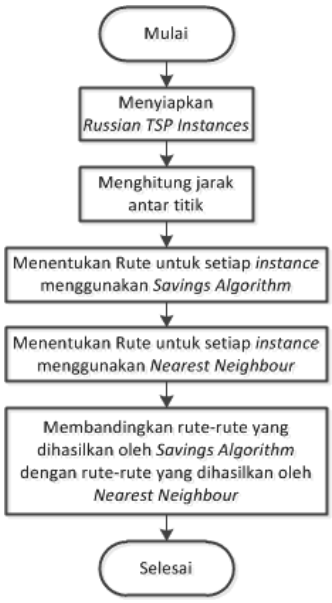
$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 . \cos\phi_2 \ . sin^2\left(\frac{\Delta\lambda}{2}\right) \quad (4)$$

$$c = 2 . atan2\left[\sqrt{a}, \sqrt{(1-a)}\right] \quad (5)$$

$$d = R . c \quad (6)$$

$\phi 1$: *Latitude* titik pertama.
$\phi 2$: *Latitude* titik kedua.
$\lambda 1$: *Longitude* titik pertama.
$\lambda 2$: *Longitude* titik kedua.
$\Delta\phi$: $\phi 1 - \phi 2$
$\Delta\lambda$: $\lambda 1 - \lambda 2$
R : Radius Bumi (6371 Km).

Langkah-langkah pada penelitian ini dapat digambarkan dengan menggunakan *flowchart* seperti berikut ini:



Gambar 2: *Flowchart* langkah-langkah penelitian

Untuk membantu menyelesaikan seluruh perhitungan digunakan perangkat lunak *GNU Octave*. *GNU Octave* merupakan alat multifungsi untuk melakukan analisis numerik. Berbagai alat bantu yang tersedia di dalam *GNU Octave* adalah sebagai berikut ini [14]:

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 [15].

III. HASIL PENELITIAN

Agar *Russian TSP Instances* dapat digunakan maka disimpan dalam *file* bertipe *text* yang disusun secara terstruktur sehingga dapat dibaca dengan baik oleh *GNU Octave*.

Instance Russia-10-Nodes_TSP akan digunakan sebagai contoh pengerjaan. Berikut ini merupakan contoh penggunaan *Haversine Formula* dalam menghitung jarak antara dua titik di Bumi:

Moscow (Lat: 55,75222; Long: 37,61556)
St.Petersburg (Lat:59,93863; Long: 30,31413)

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

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

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

Pada Tabel I dapat dilihat jarak antar titik berdasarkan hasil perhitungan *Haversine Formula*.

TABEL I
JARAK ANTAR TITIK (KM)

Node_No	1	2	3	4	5	6	7	8	9	10
1	0	635	2812	1418	402	857	2236	719	959	1496
2	635	0	3105	1783	896	1420	2584	1200	1540	1912
3	2812	3105	0	1398	2412	2127	610	2115	3083	1363
4	1418	1783	1398	0	1017	780	820	718	1773	196
5	402	896	2412	1017	0	526	1834	324	1054	1096
6	857	1420	2127	780	526	0	1520	296	994	765
7	2236	2584	610	820	1834	1520	0	1527	2474	760
8	719	1200	2115	718	324	296	1527	0	1151	778
9	959	1540	3083	1773	1054	994	2474	1151	0	1740
10	1496	1912	1363	196	1096	765	760	778	1740	0

Berikut ini merupakan contoh perhitungan nilai *Savings*:

$S_{ij} = d_{i0} + d_{0j} - d_{ij}$
$S_{23} = d_{21} + d_{13} - d_{23}$
$S_{23} = 635 + 2812 - 3105$
$S_{23} = 342\ Km$

Pada tabel II dapat dilihat nilai *Savings* antar titik. Dapat diketahui bahwa nilai *Savings* antara titik 3 dan 7 merupakan yang tertinggi yaitu 4438 Km, maka titik 3 dan 7 dimasukkan ke dalam rute dan dikunjungi secara berurutan. Nilai *Savings* tertinggi kedua adalah antara titik 7 dan 10 yaitu 2972 Km, maka titik 10 dimasukkan ke dalam rute dan dikunjungi setelah titik 7. Cara ini terus diulang hingga seluruh titik masuk ke dalam rute.

TABEL II
NILAI *SAVINGS* ANTAR TITIK (KM)

Node_No	Node_No	Savings
2	3	342
2	4	270
2	5	141
2	6	72
2	7	287
2	8	154
2	9	54
2	10	219
3	4	2832
3	5	802
3	6	1542
3	7	4438
3	8	1416
3	9	688
3	10	2945
4	5	803
4	6	1495
4	7	2834
4	8	1419
4	9	604
4	10	2718
5	6	733
5	7	804
5	8	797
5	9	307
5	10	802
6	7	1573
6	8	1280
6	9	822
6	10	1588
7	8	1428
7	9	721
7	10	2972
8	9	527
8	10	1437
9	10	715

Pada Tabel III dapat dilihat panjang rute dan *CPU Time* yang dihasilkan dari *Savings Algorithm* untuk setiap *instance*. Untuk rute yang dihasilkan dari *Savings Algorithm* dapat dilihat pada lampiran.

TABEL III
PANJANG RUTE DAN *CPU TIME* YANG DIHASILKAN DARI *SAVINGS ALGORITHM*

No	Instance	Panjang Rute (km)	CPU Time (det)
1	AK-47-TSP	23861	0,2652
2	Gagarin-108-TSP	30283	0,702
3	Mendeleev-101-TSP	34371	0,6708
4	Petersburg-182-TSP	1411	2,3712
5	Popov-250-TSP	53000	5,148
6	Russia-10-Nodes-TSP	8059	0,0468
7	Russia-20-Nodes-TSP	10043	0,078
8	Siege-of-Leningrad-872-TSP	4066	954,8041
9	World-Cup-Stadium-12-TSP	7250	0,1092
10	Yashin-270-TSP	40037	7,176

Nearest Neighbour menentukan titik yang dikunjungi berdasarkan titik terdekat yang belum dikunjungi. Dimulai dari depot yaitu titik 1, diketahui titik terdekat yang belum dikunjungi adalah titik 5, kemudian dari titik 5 diketahui titik terdekat yang belum dikunjungi adalah titik 8, cara ini terus diulang hingga seluruh titik dikunjungi.

Pada Tabel IV dapat dilihat panjang rute dan *CPU Time* yang dihasilkan dari *Nearest Neighbour* untuk setiap *instance*. Untuk rute yang dihasilkan dari *Nearest Neighbour* dapat dilihat pada lampiran.

TABEL IV
PANJANG RUTE DAN *CPU TIME* YANG DIHASILKAN DARI *NEAREST NEIGHBOUR*

No	Instance	Panjang Rute (km)	CPU Time (det)
1	AK-47-TSP	27803	0,078
2	Gagarin-108-TSP	36738	0,2028
3	Mendeleev-101-TSP	38067	0,156
4	Petersburg-182-TSP	1573	0,3588
5	Popov-250-TSP	58309	0,7176
6	Russia-10-Nodes-TSP	8671	0,0312
7	Russia-20-Nodes-TSP	12334	0,0312
8	Siege-of-Leningrad-872-TSP	4267	7,7688
9	World-Cup-Stadium-12-TSP	8656	0,0624
10	Yashin-270-TSP	46234	0,8736

IV. PEMBAHASAN

Untuk menghitung nilai perbandingan panjang rute digunakan persamaan berikut ini:

$$Nilai\ Perbandingan = \frac{Rute\ SA - Rute\ NN}{Rute\ SA} \quad (7)$$

Rute *SA* : Panjang Rute dari *Savings Algorithm*.
Rute *NN*: Panjang Rute dari *Nearest Neighbour*.

Untuk contoh perhitungan, digunakan *instance AK-47-TSP*, maka perhitungannya seperti berikut ini:

$$Nilai\ Perbandingan = \frac{23861 - 27803}{23861}$$

$$Nilai\ Perbandingan = -16{,}5207\%$$

Nilai Perbandingan -16,5207% berarti panjang rute yang dihasilkan dari *Nearest Neighbour* lebih panjang 16,5207% dibandingkan panjang rute yang dihasilkan dari *Savings Algorithm*.

Untuk menghitung nilai perbandingan *CPU Time* digunakan persamaan berikut ini:

$$Nilai\ Perbandingan = \frac{CPU\ SA - CPU\ NN}{CPU\ SA} \quad (8)$$

CPU SA : *CPU Time* dari *Savings Algorithm*.
CPU NN: *CPU Time* dari *Nearest Neighbour*.

Untuk contoh perhitungan, digunakan *instance AK-47-TSP*, maka perhitungannya seperti berikut ini:

$$Nilai\ Perbandingan = \frac{0{,}2652 - 0{,}078}{0{,}2652}$$

$$Nilai\ Perbandingan = 70{,}5882\%$$

Nilai Perbandingan 70,5882% berarti *CPU Time* yang dihasilkan dari *Nearest Neighbour* lebih cepat 70,5882% dibandingkan *CPU Time* yang dihasilkan dari *Savings Algorithm*.

Pada Tabel V dapat dilihat perbandingan panjang rute dan *CPU Time* yang dihasilkan dari *Savings Algorithm* dengan panjang rute dan *CPU Time* yang dihasilkan dari *Nearest Neighbour*.

Panjang rute yang dihasilkan maka *Savings Algorithm* lebih baik dibandingkan *Nearest Neighbour* karena panjang rute yang dihasilkan dari *Savings Algorithm* lebih pendek dibandingkan panjang rute yang dihasilkan dari *Nearest Neighbour*. Sedangkan untuk *CPU Time* yang dihasilkan, maka *Nearest Neighbour* lebih baik dibandingkan *Savings Algorithm* karena *CPU Time* yang dihasilkan dari *Nearest Neighbour* lebih cepat dibandingkan *CPU Time* yang dihasilkan dari *Savings Algorithm*.

TABEL V
PERBANDINGAN PANJANG RUTE DAN CPU TIME YANG DIHASILKAN DARI *SAVINGS ALGORITHM* DENGAN *NEAREST NEIGHBOUR*

		Savings Algorithm vs Nearest Neighbour	
No	**Instance**	**Panjang Rute**	**CPU Time**
1	AK-47-TSP	-16,5207%	70,5882%
2	Gagarin-108-TSP	-21,3156%	71,1111%
3	Mendeleev-101-TSP	-10,7533%	76,7442%
4	Petersburg-182-TSP	-11,4812%	84,8684%
5	Popov-250-TSP	-10,0170%	86,0606%
6	Russia-10-Nodes-TSP	-7,5940%	33,3333%
7	Russia-20-Nodes-TSP	-22,8119%	60,0000%
8	Siege-of-Leningrad-872-TSP	-4,9434%	99,1863%
9	World-Cup-Stadium-12-TSP	-19,3931%	42,8571%
10	Yashin-270-TSP	-15,4782%	87,8261%

V. KESIMPULAN

Savings Algorithm dan *Nearest Neighbour* mampu menyelesaikan *Russian TSP Instances* dengan lama waktu perhitungan yang wajar. Jika dalam penentuan rute, yang diprioritaskan adalah rute yang lebih pendek maka dapat digunakan *Savings Algorithm*, tetapi jika dalam penentuan rute, yang diprioritaskan adalah *CPU Time* yang lebih cepat maka dapat digunakan *Nearest Neighbour*. Agar dapat diketahui lebih lanjut mengenai kehandalan *Savings Algorithm* dan *Nearest Neighbour* dalam menyelesaikan *TSP*, maka perlu dilakukan pengujian lebih lanjut pada *instance* yang berbeda.

REFERENSI

[1] A. Yalaoui, H. Chehade, F. Yalaoui and L. Amodeo, *Optimization of Logistics*, ISTE, London, 2012.

[2] D. Waters, *Logistics An Introduction to Supply Chain Management*, Palgrave Macmillan, New York, 2003.

[3] N. Labadie, C. Prins and C. Prodhon, *Metaheuristics for Vehicle Routing Problems*, ISTE, London, 2016.

[4] D.L. Applegate, R.E. Bixby, V. Chvatal and W.J. Cook, *The Traveling Salesman Problem A Computational Study*, Princeton University Press, New Jersey, 2006.

[5] P. Toth and D. Vigo, *The Vehicle Routing Problem*, SIAM, Philadelphia, 2002.

[6] C. Nilsson, *Heuristics for The Traveling Salesman Problem*, Linkoping University.

[7] K. Kumar, D. Zindani and J. Paulo Davim, *Metaheuristics Optimizing Engineering Problems Through Heuristic Techniques*, CRC Press, USA, 2020.

[8] M. Chiarandini, *Construction Heuristics for Traveling Salesman Problem*, University of Southern, Denmark 2020.

[9] K. Arora, S. Agarwal and R. Tanwar, *Solving TSP using Genetic Algorithm and Nearest Neighbour Algorithm and their Comparison*, International Journal of Scientific & Engineering Research, Volume 7, Issue 1, January 2016.

[10] J. Lysgaard, *Clarke & Wright's Savings Algorithm*, Department of Management Science and Logistics, The Aarhus School of Business, Sep. 2007.

[11] H. Paessens, *The Savings Algorithm fot The Vehicle Routing Problem*, European Journal of Operational Research 34, North-Holland, 1988.

[12] https://download.geonames.org/export

[13] B. Idrizi, *Necessity for Geometric Corrections of Distances in Web and Mobile Maps*, International Conference on Cartography and GIS, Bulgaria, 2020.

[14] A. Quarteroni and F. Saleri, *Scientific Computing with MATLAB and Octave Second Edition*, Springer, 2006.

[15] J.S. Hansen, *GNU Octave Beginner's Guide*, PACKT, Birmingham, 2011.

LAMPIRAN 1

Rute yang Dihasilkan dari *Savings Algorithm*

No	*Instance*	Rute
1	AK-47-TSP	1-23-29-25-26-27-24-15-38-39-4-35-3-43-44-47-17-32-40-2-30-22-45-6-33-10-19-13-36-37-16-20-42-9-18-7-34-41-21-14-12-28-46-11-5-31-8-1
2	Gagarin-108-TSP	1-48-95-83-101-2-81-94-65-22-93-63-64-72-45-86-6-30-32-47-53-88-17-69-33-67-42-70-9-20-16-104-59-13-90-10-71-82-19-50-103-57-74-62-89-37-36-102-107-18-7-85-34-12-75-28-78-46-106-11-31-54-8-4-38-39-23-96-92-29-15-77-80-25-51-66-91-24-26-27-73-108-84-79-68-5-49-14-21-41-87-52-105-61-76-35-56-3-99-98-97-100-43-60-44-58-40-55-1
3	Mendeleev-101-TSP	1-31-96-5-11-46-60-28-12-86-14-91-99-95-41-21-34-75-70-9-80-42-20-18-7-65-16-33-6-85-90-22-45-30-79-2-74-40-69-32-64-17-59-13-36-37-51-55-54-10-19-53-47-44-58-63-48-84-89-94-35-3-88-83-78-43-73-68-49-93-98-100-101-52-50-97-92-87-82-72-77-67-62-57-27-26-24-56-61-25-66-15-29-39-38-4-71-23-76-8-81-1
4	Petersburg-182-TSP	1-169-5-139-8-17-2-116-38-23-46-45-165-40-14-13-21-81-37-10-9-18-71-73-20-75-27-61-93-84-164-144-57-147-174-162-151-59-3-171-161-142-88-156-148-89-62-29-79-110-137-77-173-140-99-163-178-68-33-172-125-80-175-121-154-138-91-180-134-95-177-152-53-167-146-76-155-107-145-159-120-133-87-119-64-122-143-153-141-51-126-123-131-114-19-128-115-39-132-72-92-96-34-78-97-102-55-117-179-103-98-118-150-56-170-176-181-182-58-136-149-111-94-158-74-12-90-160-48-83-86-106-108-36-65-47-63-100-54-60-42-104-44-82-112-49-124-11-135-113-109-101-85-157-32-105-43-70-69-22-35-15-52-28-25-166-168-50-7-127-66-67-129-41-130-24-30-26-16-31-6-4-1
5	Popov-250-TSP	1-235-246-87-244-243-52-105-242-232-41-218-204-9-70-42-189-161-147-7-18-107-20-174-6-188-86-203-231-217-64-230-229-228-63-216-72-202-45-173-187-65-201-22-215-93-214-213-200-186-94-30-172-158-159-67-160-33-146-102-104-16-131-130-145-69-144-32-143-157-40-171-156-55-81-2-101-83-95-185-48-199-198-184-170-169-155-58-142-17-88-127-128-129-132-36-120-59-13-118-90-71-10-114-113-103-50-117-37-121-110-89-119-62-74-116-115-19-57-82-112-53-47-141-44-126-140-60-183-212-240-76-227-226-225-56-35-98-3-99-211-97-197-100-210-196-182-43-168-154-108-224-241-239-61-238-247-245-111-248-249-84-250-109-237-223-209-195-167-181-153-139-125-124-123-24-122-27-26-138-73-137-91-136-66-152-135-51-134-25-80-151-166-150-77-15-194-180-165-29-92-149-96-23-164-179-4-39-38-193-236-208-79-68-178-8-163-192-177-54-148-78-133-28-75-162-46-12-175-85-34-205-21-219-220-233-14-221-190-106-11-176-191-207-31-5-222-49-206-234-1
6	Russia-10-Nodes-TSP	1-2-5-8-4-3-7-10-6-9-1
7	Russia-20-Nodes-TSP	1-16-9-18-12-15-19-17-6-11-10-7-3-14-4-13-20-8-5-2-1
8	Siege-of-Leningrad-872-TSP	1-625-169-631-629-628-627-626-624-623-621-620-619-618-617-616-615-613-614-697-630-622-8-699-640-17-653-652-650-649-648-647-645-644-585-584-582-578-586-587-583-569-568-566-558-31-497-561-557-556-555-554-551-550-549-546-548-492-16-491-487-486-52-483-481-482-28-25-489-490-488-26-501-500-504-503-30-24-508-130-50-502-499-168-166-498-496-495-493-494-434-135-432-430-429-43-485-420-421-425-426-424-157-422-32-484-105-70-480-479-478-15-542-541-476-69-474-473-472-471-470-22-540-469-467-538-35-539-678-677-675-676-34-674-672-673-604-537-536-534-533-532-603-97-671-680-132-39-682-681-115-685-128-19-114-679-131-123-764-766-80-704-707-77-714-720-718-717-154-121-175-782-787-138-91-180-796-728-729-726-110-727-665-730-596-733-137-731-732-804-799-798-794-789-775-771-770-774-780-779-152-134-177-95-792-797-793-795-783-777-773-53-786-788-784-778-776-772-769-765-763-762-760-51-141-752-126-756-754-750-64-153-668-143-122-670-743-741-740-119-744-746-87-758-757-755-761-759-767-768-146-167-781-785-790-791-801-802-803-800-872-864-856-854-852-76-839-826-120-753-748-747-749-751-133-815-745-739-742-812-808-807-811-822-821-825-835-844-846-850-848-843-155-845-859-867-866-869-862-868-870-871-855-847-837-145-831-830-107-836-840-841-849-853-860-858-861-159-865-863-857-851-833-829-817-814-813-810-816-819-820-823-828-832-842-838-834-186-827-824-818-809-806-805-738-736-737-667-669-602-531-535-468-477-475-412-407-408-85-416-419-423-431-433-11-124-440-439-49-437-438-436-48-435-428-427-371-367-113-418-417-415-414-413-411-410-409-350-109-406-404-403-466-464-463-530-601-78-600-599-598-529-460-461-462-465-399-101-402-400-401-398-339-405-345-346-353-355-158-74-364-363-358-356-354-94-352-348-349-351-279-360-357-286-284-288-290-149-293-136-291-289-285-281-280-278-347-340-338-337-335-333-332-397-396-394-395-393-391-392-330-331-328-326-324-102-323-327-329-334-336-342-341-344-276-111-343-273-274-277-282-150-56-294-98-292-287-283-118-179-275-272-271-270-269-117-266-265-263-262-55-259-325-183-255-256-258-260-261-264-268-267-201-196-195-257-187-191-189-194-193-190-192-188-185-200-202-199-197-198-103-211-213-218-219-221-226-230-236-240-241-244-249-250-252-253-254-251-247-245-243-238-234-228-224-220-217-214-209-210-206-207-205-203-204-208-212-215-216-222-229-232-231-235-237-239-242-246-248-318-319-320-86-321-322-106-317-315-312-311-306-233-176-225-227-223-170-181-182-302-295-296-299-58-298-300-301-304-305-307-308-310-313-314-316-108-387-389-390-388-458-459-528-184-527-457-456-386-385-36-309-380-303-379-297-370-366-368-12-361-359-362-365-90-369-372-160-373-374-375-376-377-83-381-382-383-384-451-65-453-454-455-63-525-526-597-735-734-60-666-54-100-524-42-523-520-519-521-522-104-518-44-515-47-452-450-449-82-443-378-112-445-442-444-448-447-446-514-517-513-512-127-66-67-7-441-506-505-41-510-511-129-509-516-595-594-592-593-664-663-724-79-725-721-722-29-723-662-62-89-148-156-661-591-161-171-590-507-589-88-719-716-713-715-59-709-710-712-711-151-162-701-702-695-690-33-125-172-173-147-140-68-178-686-72-92-96-683-684-84-605-687-93-163-691-99-144-696-57-174-708-71-73-20-700-75-698-10-37-164-27-61-688-689-606-543-544-545-547-553-607-608-609-692-693-694-40-14-81-21-18-9-705-2-706-3-660-142-658-659-657-656-588-654-655-703-13-116-38-23-46-139-165-610-612-45-611-5-552-559-560-562-563-564-6-565-567-570-573-574-575-576-577-579-580-581-651-4-642-643-646-641-638-639-571-572-637-632-633-634-635-636-1
9	World-Cup-Stadium-12-TSP	1-7-10-8-12-9-5-4-6-11-2-3-1
10	Yashin-270-TSP	1-174-208-166-263-154-193-168-250-178-171-109-164-169-177-147-122-204-129-232-225-123-247-100-82-94-261-47-187-134-175-170-64-44-21-92-62-63-71-231-128-161-242-186-85-5-41-157-69-214-8-40-238-86-203-262-181-67-78-183-227-143-83-223-107-72-25-26-111-23-133-202-126-90-65-50-24-79-249-76-212-206-229-14-189-116-207-28-91-197-188-38-172-255-253-37-185-95-22-239-3-119-7-179-30-53-265-108-10-256-115-105-259-4-149-228-48-218-210-113-13-45-77-194-27-121-74-11-190-234-144-163-216-246-217-20-211-33-84-244-6-182-17-151-19-106-101-35-195-258-36-162-88-61-118-73-140-260-150-270-173-148-49-102-56-135-18-81-264-192-114-117-184-9-257-70-243-215-89-12-58-156-103-15-237-32-66-219-68-230-16-87-245-52-46-220-57-43-42-99-266-60-104-51-75-224-254-251-198-34-153-98-2-125-97-55-233-146-240-96-199-59-54-176-31-152-39-29-132-252-269-222-221-142-130-180-201-145-120-236-248-124-196-267-160-226-127-131-93-139-167-268-155-137-136-158-191-110-141-241-159-165-80-205-112-138-213-209-235-200-1

LAMPIRAN 2

Rute yang Dihasilkan dari *Nearest Neighbour*

No	Instance	Rute
1	AK-47-TSP	1-23-4-39-38-29-15-25-8-31-5-11-46-12-34-21-14-41-42-9-20-18-7-16-33-6-45-22-2-40-30-32-17-47-44-10-19-13-36-37-28-35-3-43-26-27-24-1
2	Gagarin-108-TSP	1-95-83-101-2-81-94-55-40-30-65-45-72-22-93-63-64-48-97-98-3-99-35-56-100-60-44-58-47-88-17-32-69-33-67-20-18-7-107-102-104-16-59-13-90-10-71-19-82-57-103-50-74-62-89-37-36-28-75-12-106-11-5-49-14-21-34-85-9-70-42-6-86-41-87-52-105-76-61-43-53-78-46-54-31-8-4-23-96-92-29-39-38-15-77-80-25-51-66-91-26-27-73-24-84-108-79-68-1
3	Mendeleev-101-TSP	1-31-5-86-14-96-91-21-34-75-70-18-7-65-20-9-80-42-6-85-90-45-22-79-2-74-40-69-30-32-64-17-59-13-54-10-19-53-55-37-51-36-16-33-47-44-63-58-78-83-88-93-3-35-94-89-84-48-95-41-12-46-11-60-28-81-76-8-4-71-23-29-39-38-15-66-25-61-26-27-24-56-57-62-67-82-87-92-77-72-97-50-52-101-100-98-99-43-73-68-49-1
4	Petersburg-182-TSP	1-169-8-139-46-23-13-21-81-10-75-37-14-40-165-45-5-6-4-17-38-116-2-9-18-71-59-73-20-57-144-99-163-68-178-33-172-125-173-147-140-162-151-174-77-121-154-180-91-138-175-80-134-177-152-95-53-167-146-76-155-145-107-120-133-87-126-123-131-114-19-128-115-39-132-72-92-96-84-93-61-27-164-3-142-88-148-156-89-62-161-171-24-30-26-16-31-25-28-52-70-69-15-22-35-34-97-78-143-122-153-141-51-64-119-101-109-85-32-43-105-157-113-90-12-74-158-94-111-179-117-55-102-150-118-56-136-149-58-170-181-182-176-98-103-160-48-49-124-11-135-166-168-50-130-41-129-67-127-66-7-82-47-65-36-108-106-86-83-112-44-104-42-100-54-60-137-110-79-29-63-159-1
5	Popov-250-TSP	1-234-221-14-206-49-222-207-191-5-11-176-106-162-75-12-190-205-21-34-85-175-161-18-7-107-20-174-67-160-33-146-16-104-102-147-28-133-78-46-148-54-177-31-192-178-8-163-208-193-38-39-92-29-165-149-96-23-164-179-4-180-194-15-77-166-80-25-151-51-134-135-152-66-136-91-137-138-73-26-27-124-123-24-122-125-139-153-195-209-223-181-167-237-109-250-84-249-248-247-111-245-61-238-239-241-105-52-242-230-229-228-63-64-216-72-22-215-93-214-200-201-45-202-203-86-188-6-173-187-65-186-94-81-2-83-101-95-185-48-199-213-212-198-183-197-97-211-226-225-35-56-98-3-99-224-100-210-196-182-60-169-155-170-184-55-156-171-40-157-143-142-58-141-47-88-127-128-129-130-131-145-69-144-32-17-53-44-140-126-172-30-158-159-42-189-70-204-218-41-219-232-220-233-87-244-243-231-217-227-76-240-9-132-59-13-118-120-36-37-121-110-89-119-62-74-117-116-115-103-50-114-113-10-71-90-19-82-112-57-43-168-154-108-235-68-79-236-246-150-1
6	Russia-10-Nodes-TSP	1-5-8-6-10-4-7-3-9-2-1
7	Russia-20-Nodes-TSP	1-5-8-19-17-6-15-12-9-18-16-2-20-13-4-10-11-7-3-14-1
8	Siege-of-Leningrad-872-TSP	1-619-616-615-613-5-552-553-547-545-544-543-606-607-608-609-610-693-692-694-40-165-45-612-139-46-23-622-8-630-699-640-17-38-703-116-2-705-9-18-71-59-709-708-73-20-700-10-698-37-14-81-13-21-75-57-696-144-99-163-691-68-178-33-172-125-173-147-140-162-151-174-711-712-710-77-714-718-717-720-121-154-180-91-138-787-175-782-80-695-701-702-704-707-690-685-115-128-19-114-679-123-131-51-752-64-750-754-756-126-760-762-763-765-766-764-771-775-774-770-773-772-769-776-778-53-783-152-134-177-789-794-796-798-797-792-95-793-795-786-788-784-781-785-790-791-864-859-856-854-852-76-146-768-767-839-835-844-846-850-848-155-843-845-847-855-853-849-841-840-836-830-107-145-831-837-825-821-822-120-826-133-751-749-747-748-753-755-757-758-87-746-744-119-740-737-741-743-122-670-143-668-667-153-141-671-669-78-600-601-599-598-529-460-530-463-464-465-399-101-402-404-403-406-408-109-409-410-411-413-414-415-417-418-113-423-419-157-422-421-32-420-484-105-485-43-425-426-424-427-428-367-90-365-362-359-74-361-12-364-363-358-360-286-357-284-281-280-279-278-276-111-343-273-274-272-179-275-277-347-344-342-341-340-338-337-336-334-117-269-265-266-262-55-263-259-255-183-325-327-329-323-102-324-392-326-328-330-331-333-332-335-398-397-396-395-393-391-394-462-461-97-531-603-532-533-534-535-536-537-604-539-35-22-538-467-469-540-541-15-542-476-69-474-473-472-471-470-468-466-407-475-412-85-416-477-70-480-479-478-481-52-483-482-486-28-25-489-490-488-166-168-495-493-494-496-498-135-432-430-429-431-433-11-124-49-439-440-112-443-378-438-436-435-48-374-373-160-372-369-371-368-366-370-290-289-149-291-136-293-56-287-292-98-216-215-212-207-205-203-204-201-195-196-191-189-187-257-260-261-264-267-268-270-271-348-349-351-94-352-354-355-356-158-353-350-346-345-405-339-401-400-602-673-675-676-34-674-672-677-678-605-84-683-684-92-96-72-132-39-681-682-686-680-687-93-61-688-689-27-164-611-614-617-620-618-621-565-567-570-571-573-572-574-575-576-577-4-579-580-581-584-585-582-578-569-568-566-563-6-562-560-559-623-624-626-627-625-169-631-628-629-632-633-634-635-636-637-638-639-641-642-643-646-648-644-645-647-649-650-652-651-653-654-655-657-658-659-3-660-713-715-716-719-88-142-171-161-591-89-62-148-156-661-662-723-724-663-664-593-592-594-595-516-517-44-515-514-513-66-512-127-67-510-41-129-511-509-508-24-30-503-50-502-499-500-26-501-497-561-31-492-548-546-549-550-551-554-555-556-558-557-564-583-586-587-588-656-589-507-590-504-130-505-506-441-7-446-447-448-444-442-445-82-449-450-65-451-453-454-47-452-455-522-104-518-521-519-520-523-524-42-100-54-666-60-734-733-730-596-665-727-110-726-79-29-722-721-725-729-728-804-731-732-137-735-597-526-525-63-457-456-458-459-528-184-527-390-389-388-387-386-108-36-383-382-381-83-379-380-303-305-304-307-308-310-313-314-316-106-317-86-320-319-318-248-246-312-311-306-233-181-182-302-170-223-227-176-235-237-239-242-243-245-238-234-228-226-221-219-218-217-214-211-103-209-210-206-198-197-199-202-200-185-192-190-193-194-188-213-220-222-224-229-232-231-225-295-296-299-58-298-297-300-301-375-376-377-437-434-16-491-487-697-706-780-779-777-167-866-867-869-862-868-870-871-860-858-861-857-851-842-838-832-833-829-828-834-186-827-824-820-823-819-816-818-809-810-813-814-817-811-812-742-808-807-805-806-738-739-745-815-759-761-736-799-801-802-803-800-872-865-863-159-384-385-309-315-321-322-251-254-253-252-250-247-244-241-240-236-230-249-294-283-150-118-208-282-285-288-258-256-1
9	World-Cup-Stadium-12-TSP	1-3-7-10-8-9-5-6-4-11-2-12-1
10	Yashin-270-TSP	1-174-235-200-124-196-160-226-267-167-93-139-112-138-213-209-131-127-268-155-137-159-165-80-205-136-241-141-109-164-171-250-168-193-208-154-178-169-177-147-263-122-204-129-232-82-100-247-123-225-236-248-120-145-180-130-142-221-222-134-175-170-252-269-132-29-152-39-158-191-54-110-166-94-201-187-261-47-92-21-71-231-44-128-64-161-242-186-85-5-41-157-69-214-8-19-151-17-182-6-106-237-32-66-68-230-31-176-16-87-245-52-46-220-57-43-59-199-99-96-240-233-146-97-2-98-125-153-251-34-198-224-254-55-266-75-62-63-40-238-211-217-246-216-163-20-33-84-144-234-190-11-74-121-27-244-101-103-15-156-58-12-114-89-215-117-9-184-257-70-243-18-81-135-102-264-49-148-173-150-260-140-270-73-118-61-88-162-36-258-195-35-192-56-219-194-77-45-256-10-259-115-105-149-228-4-108-265-53-30-179-181-67-78-183-227-262-203-86-210-113-13-48-218-7-3-239-253-255-37-185-38-188-197-91-28-207-116-14-229-206-172-95-22-119-189-249-79-24-50-65-76-212-143-83-223-107-111-72-25-26-126-202-23-133-90-51-104-60-42-1

LAMPIRAN 3

Listing Program *Savings Algorithm*

```
clc;

clear;

cd ("Instances (Original)");
INSTANCES=readdir(".");
cd ("..");

global DISTANCE;

for i=3:size(INSTANCES,1)
  INSTANCE=char(INSTANCES(i))
  INSTANCE=INSTANCE(1,1:length(INSTANCE)-4);

  START_FINISH_NODE=1;

  %cd ("Instances (Converted)");
  %SAVINGS=load(strcat(INSTANCE,"_Savings.txt"));
  %cd ("..");

  cd ("Instances (Converted)");
  START_TIME=cputime;
  DISTANCE=load(strcat(INSTANCE,"_Distance.txt"));
  cd ("..");

  %SAVINGS
  INDEX_SAVINGS=0;
  for j=2:size(DISTANCE,1)
    for k=j+1:size(DISTANCE,1)
      if j!=k
        INDEX_SAVINGS=INDEX_SAVINGS+1;
        SAVINGS(INDEX_SAVINGS,:)=[j k DISTANCE(j,1)+DISTANCE(1,k)-DISTANCE(j,k)];
      endif
    endfor
  endfor

  SORTED_SAVINGS=[(1:size(SAVINGS,1))' sortrows(SAVINGS,-3)];
  clear('SAVINGS');

  %INDIVIDUAL SAVINGS ALGORITHM
  %INSERT NODE BASED ON HIGHEST SAVINGS
  INDIVIDUAL_SAVINGS_ALGORITHM=SORTED_SAVINGS(1,2:3);
  SORTED_SAVINGS(1,5)=1;


  while size(find(SORTED_SAVINGS(:,5)==0),1)>0
    %SAVINGS_COMPETITION
    k=0;
    if size(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),1)>0
      k=k+1;
      SAVINGS_COMPETITION(k,1:4)=...

[SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==IN
DIVIDUAL_SAVINGS_ALGORITHM(1,1))(1,1),1:4)];
    endif

    if size(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,3)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),1)>0
      k=k+1;
      SAVINGS_COMPETITION(k,1:4)=...

[SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,3)==IN
DIVIDUAL_SAVINGS_ALGORITHM(1,1))(1,1),1:4)];
    endif

    if
size(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVI
DUAL_SAVINGS_ALGORITHM,2))),1)>0
      k=k+1;
```

```
    SAVINGS_COMPETITION(k,1:4)=...

[SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==IN
DIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2)))(1,1),1:4)];
  endif

  if
size(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,3)==INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVI
DUAL_SAVINGS_ALGORITHM,2))),1)>0
    k=k+1;
    SAVINGS_COMPETITION(k,1:4)=...

[SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,3)==IN
DIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2)))(1,1),1:4)];
  endif

  %CANDIDATE NODE
  CANDIDATE_NODE=sortrows(SAVINGS_COMPETITION,-4)(1,1:3);
  clear('SAVINGS_COMPETITION');

  %INSERT NODE BASED ON CANDIDATE NODE
  if INDIVIDUAL_SAVINGS_ALGORITHM(1,1)==CANDIDATE_NODE(1,2)
    INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY=INDIVIDUAL_SAVINGS_ALGORITHM;
    clear('INDIVIDUAL_SAVINGS_ALGORITHM');
    INDIVIDUAL_SAVINGS_ALGORITHM(1,1)=CANDIDATE_NODE(1,3);

INDIVIDUAL_SAVINGS_ALGORITHM(1,2:size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2)+1)=INDIVIDUAL_SAVI
NGS_ALGORITHM_DUMMY;
    ELIMINATED_NODE=CANDIDATE_NODE(1,2);
  elseif INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2))==CANDIDATE_NODE(1,2)
    INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY=INDIVIDUAL_SAVINGS_ALGORITHM;
    clear('INDIVIDUAL_SAVINGS_ALGORITHM');

INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2)+1)=CANDIDATE_NODE(1
,3);

INDIVIDUAL_SAVINGS_ALGORITHM(1,1:size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2))=INDIVIDUAL_SAVING
S_ALGORITHM_DUMMY;
    ELIMINATED_NODE=CANDIDATE_NODE(1,2);
  elseif INDIVIDUAL_SAVINGS_ALGORITHM(1,1)==CANDIDATE_NODE(1,3)
    INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY=INDIVIDUAL_SAVINGS_ALGORITHM;
    clear('INDIVIDUAL_SAVINGS_ALGORITHM');
    INDIVIDUAL_SAVINGS_ALGORITHM(1,1)=CANDIDATE_NODE(1,2);

INDIVIDUAL_SAVINGS_ALGORITHM(1,2:size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2)+1)=INDIVIDUAL_SAVI
NGS_ALGORITHM_DUMMY;
    ELIMINATED_NODE=CANDIDATE_NODE(1,3);
  elseif INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2))==CANDIDATE_NODE(1,3)
    INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY=INDIVIDUAL_SAVINGS_ALGORITHM;
    clear('INDIVIDUAL_SAVINGS_ALGORITHM');

INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2)+1)=CANDIDATE_NODE(1
,2);

INDIVIDUAL_SAVINGS_ALGORITHM(1,1:size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2))=INDIVIDUAL_SAVING
S_ALGORITHM_DUMMY;
    ELIMINATED_NODE=CANDIDATE_NODE(1,3);
  endif

  clear('INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY');

  SORTED_SAVINGS(CANDIDATE_NODE(1,1),5)=1;

  clear('CANDIDATE_NODE');

  %ELIMINATED NODE

SORTED_SAVINGS(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVING
S(:,5)==0),:)(:,2)==ELIMINATED_NODE),1),5)=1;

SORTED_SAVINGS(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVING
S(:,5)==0),:)(:,3)==ELIMINATED_NODE),1),5)=1;

  clear('ELIMINATED_NODE');
```

```
   %ELIMINATED SAVINGS BASED ON MOST LEFT NODE AND MOST RIGHT NODE
   if
size(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),:)(:,3)==INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2))),1),1)>0
     INDEX_SORTED_SAVINGS=...

SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),:)(:,3)==INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2))),1);
   else
     INDEX_SORTED_SAVINGS=...

SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2))),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(find(SORTED_SAVINGS(find(SORTED_SAVINGS(:,5)==0),:)(:,2)==INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2))),:)(:,3)==INDIVIDUAL_SAVINGS_ALGORITHM(1,1)),1);
   endif

   SORTED_SAVINGS(INDEX_SORTED_SAVINGS,5)=1;

  endwhile

  %INSERT START FINISH NODE
  INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY=INDIVIDUAL_SAVINGS_ALGORITHM;
  clear('INDIVIDUAL_SAVINGS_ALGORITHM');
  INDIVIDUAL_SAVINGS_ALGORITHM(1,1)=START_FINISH_NODE;

INDIVIDUAL_SAVINGS_ALGORITHM(1,2:size(INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY,2)+1)=INDIVIDUAL_SAVINGS_ALGORITHM_DUMMY;
  INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2)+1)=START_FINISH_NODE;

  %cd ("Instances (Converted)");
  %DISTANCE=load(strcat(INSTANCE,"_Distance.txt"));
  %cd ("..");

  INDIVIDUAL_SAVINGS_ALGORITHM(1,size(INDIVIDUAL_SAVINGS_ALGORITHM,2)+1)=...
  FUNCTION_LENGTH(INDIVIDUAL_SAVINGS_ALGORITHM(1,1:size(INDIVIDUAL_SAVINGS_ALGORITHM,2)));

  clear('DISTANCE');

  cd ("Instances (Result)");
  dlmwrite(strcat(INSTANCE,"_Savings_Algorithm.txt"),INDIVIDUAL_SAVINGS_ALGORITHM,"delimiter","\t");

  FINISH_TIME=cputime;

  LENGTH_TIME=FINISH_TIME-START_TIME

  dlmwrite(strcat(INSTANCE,"_Savings_Algorithm_CPUTIME.txt"),LENGTH_TIME,"delimiter","\t");

  cd ("..");

  clear('SORTED_SAVINGS');

endfor
```

LAMPIRAN 4

Listing Program *Nearest Neighbour*

```
clc;

clear;

%INSTANCES DIRECTORY
cd ("Instances (Original)");
INSTANCES=readdir(".");
cd ("..");

global DISTANCE;

for i=3:size(INSTANCES,1)
  INSTANCE=char(INSTANCES(i))(1,1:size(char(INSTANCES(i)),2)-4)

  %INSTANCES (CONVERTED) DIRECTORY
  cd ("Instances (Converted)");
  START_TIME=cputime;
  DISTANCE=load(strcat(INSTANCE,"_Distance.txt"));
  cd ("..");

  %INDIVIDUAL NEAREST NEIGHBOUR
  DISTANCE_DUMMY=DISTANCE;
  j=1;

  START_FINISH_PLACE=1;

  NEXT_PLACE=START_FINISH_PLACE;
  INDIVIDUAL_NEAREST_NEIGHBOUR(1,j)=NEXT_PLACE;
  for j=2:size(DISTANCE,2)
    [DISTANCE_DUMMY_SORT, PLACE_NUMBER_DUMMY_SORT]=sort(DISTANCE_DUMMY(NEXT_PLACE,:));
    NEXT_PLACE=PLACE_NUMBER_DUMMY_SORT(1,2);
    INDIVIDUAL_NEAREST_NEIGHBOUR(1,j)=NEXT_PLACE;
    DISTANCE_DUMMY(INDIVIDUAL_NEAREST_NEIGHBOUR(1,j-1),:)=999999999;
    DISTANCE_DUMMY(:,INDIVIDUAL_NEAREST_NEIGHBOUR(1,j-1))=999999999;
  endfor
  j=j+1;
  NEXT_PLACE=START_FINISH_PLACE;
  INDIVIDUAL_NEAREST_NEIGHBOUR(1,j)=NEXT_PLACE;

  INDIVIDUAL_NEAREST_NEIGHBOUR(1,size(INDIVIDUAL_NEAREST_NEIGHBOUR,2)+1)=...
  FUNCTION_LENGTH(INDIVIDUAL_NEAREST_NEIGHBOUR(1,1:size(INDIVIDUAL_NEAREST_NEIGHBOUR,2)));

  %SOLUTIONS NEAREST NEIGHBOUR DIRECTORY
  cd ("Instances (Result)");
  dlmwrite(strcat(INSTANCE,"_Nearest_Neighbour.txt"),INDIVIDUAL_NEAREST_NEIGHBOUR,"delimiter","\t");

  FINISH_TIME=cputime;

  LENGTH_TIME=FINISH_TIME-START_TIME

  dlmwrite(strcat(INSTANCE,"_Nearest_Neighbour_CPUTIME.txt"),LENGTH_TIME,"delimiter","\t");

  cd ("..");

  clear('DISTANCE');
  clear('INDIVIDUAL_NEAREST_NEIGHBOUR');
endfor
```