RESEARCH ARTICLE OPEN ACCESS

IMPLEMENTATION OF SHORTEST ROUTE FILE DELIVERY ON THE MESSENGER POPULATION AND CIVIL REGISTRATION OF DKI JAKARTA USING TSP BACKTRACKING METHOD

Tikah Sucilawati *, Nia Rahma Kurnianda**
*(Computer Science Faculty, Mercu Buana University, Jakarta
Email: nia.rahma@mercubuana.ac.id)
** (Computer Science Faculty, Mercu Buana University, Jakarta
Email: 41816120068@student.mercubuana.ac.id)

****

Abstract:

Currently, the speed in service is a certain action that is carried out correctly, which can have an impact on meeting community needs. Delivery of old files from messenger can cause complaints against the public in the population and civil registration services. For this reason, efficient route search is needed in order to help ease the work of how to send files or goods more quickly by not repeating the same path. This study aims to find information on determining the closest route from the population and civil registration office to each sub-district and back to the starting point, namely the population and civil registration office. To achieve this goal, researchers used the Backtracking Traveling Salesman Problem (TSP) method. Where this method can make route information faster and more efficient.

Keywords — TSP, Backtracking, Dinas Population and Civil Registration Tribe

****

I. INTRODUCTION

The Department of Population and Civil Registration is a service that emphasizes community satisfaction [1]. In addition, it builds an impression that can provide a positive image in the eyes of the community because the services are good, fast and free of charge. For the community, it can encourage people to work together and play an active role in the implementation of excellent service because the purpose of public service is to satisfy accordingly. With the wishes of society in general. So to avoid this, a quality service is needed that is in accordance with the needs and desires of society. This is what is meant by service quality. It is the conformity between expectations and desires. Public services are becoming an increasingly strategic policy issue because the improvement of public services in Indonesia tends to "run in place" while the implications are very broad in economic, political, socio-cultural and other life [2].

Public service is one of the important services that cannot be ignored by the local government because if this public service is hampered, it can cause all sectors to have an impact too. Therefore, there needs to be good planning and even the need to formulate community service standards according to the authority given by the central government in local government [3].

In this case, there is an important role where the task is as a distributor of goods or documents for population administration and civil society registration. The problem that often occurs in sending goods is that the delivery of goods takes a long time because they have to visit several districts that require goods from the population and civil registration service tribe which are quite far from each district, then have to return to their starting point without repeating the same path so as not to waste time for messenger work.

So based on these problems, the researchers raised the topic of determining the optimal path with the Traveling Salesman Problem method. This method can be used to solve the problem of determining the optimal path, the Traveling Salesman Problem method itself is a method used to minimize distribution costs by for the shortest distance and route, fastest time and minimal cost.

A. *Distribution*

Distribution is the process of sending a product from one party to another [4]. The company that produces the product sells the product to the distributor, then the distributor sells the product to the retailer or customer. The process of distributing products from factories to distributors requires transportation. Transportation in the distribution process will determine the success of product delivery to the distributor's location with the right amount, good product condition and the right time.

On time delivery is one of the goals of the distribution process which can be done by understanding the distribution destination locations. The intended location in the distribution is not in one location only but in many locations. Distribution to many locations requires transportation costs which are quite high and even exceed the budgeted costs. Transportation costs that exceed the budget can be due to the determination of distribution routes being still done manually or randomly, namely, determining distribution routes based on estimates only.

B. *The Route Determination*

Determination of the route is grouped into 2, namely, for arc tracing and the tracing of nodes. For arc tracing it means that all networks are visited only once and for node tracing it means that all points are visited once. Routing error is divided into daily, periodic, and fixed route determination [5].

C. *Graph*

Graph in general can be defined as a collection of points connected by lines, can be written G (V, E). The graph is divided into 3 types, namely undirected graph, meaning that it has no directional orientation, directed graph is a graph which specifically contains a flow, for example, the flow of loads from one point to another, so that it must be arranged sequentially, while weighted graph is a given graph. Price or weight. Graph. The weight can be state the distance between two cities, the cost of travel between the two cities, travel time, production costs, and so on (Munir, 2009: 376) [6].

D. *Travelling Salesman Problem (TSP)*

Traveling Salesman Problem TSP is a problem that a salesman has in finding the shortest alternative route to visit the specified places, where they just start and return to the same place and visit these places [7].

E. *Python*

Python is a high-level open language that is currently widely used in various fields, especially in the world of scientific computing [8]. Python was developed by Guido van Rossum in 1989 in Amsterdam and was first introduced in 1991 [9] [10]. Python is an advanced development of the ABC programming language. Python is designed to make it easy for programmers both in terms of time efficiency and ease of program development. Python has several features, including

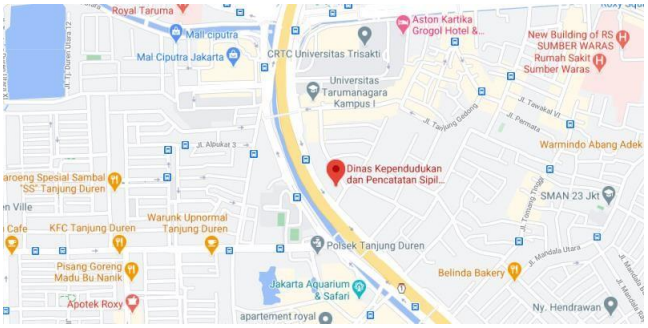
1. Grammar that is easy to understand.
2. It has many libraries. Python provides ready-to-use libraries.
3. Have a layout or layout that makes it easy to check, read back, and rewrite source code.

4. It has a modular design which is easy to develop.

II. RESEARCH METHOD

A. Research Location

The research location used in this study is located in 5 sub-Department Offices of Population and Civil Registry in DKI Jakarta.



Picture Population and Civil Registry in West Jakarta.

Picture Population and Civil Registry Research Locations of North Jakarta.

Picture Population and Civil Registry Research Locations in South Jakarta.

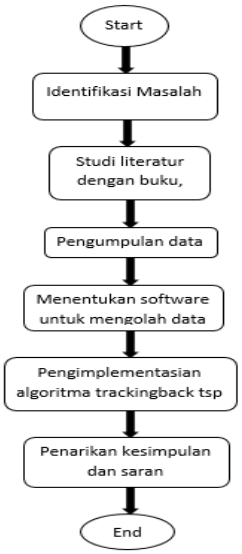
Picture Population and Civil Registry in East Jakarta.

Picture Population and Civil Registry in Central Jakarta.

B. Data Collection Method

The data collection technique used in this research is to study literature with books, journals, and reports related to the research. Then make observations by mapping the distance between the population and civil registry offices and the sub-district offices of each DKI Jakarta area. Then the results are converted into a graph.

C. Research flow diagram



Research flow diagram

The following is an explanation of the flow chart above:

1. The first step is to identify the problem. Problem identification aims to determine whether there is a problem or a phenomenon that occurs.
2. Next, do a literature study with books, journals or reports related to research.
3. The third stage is to collect data on the distance to the locations that will be used in this study.
4. The fourth stage is determining the software or supporting facilities that will be used to perform data processing.
5. Next, implement the TSP tracking back algorithm.
6. After implementation, the results of the implementation are evaluated.
7. And the last is to draw conclusions and provide suggestions for further research.

III. RESULTS AND DISCUSSION

A. Identification of problems

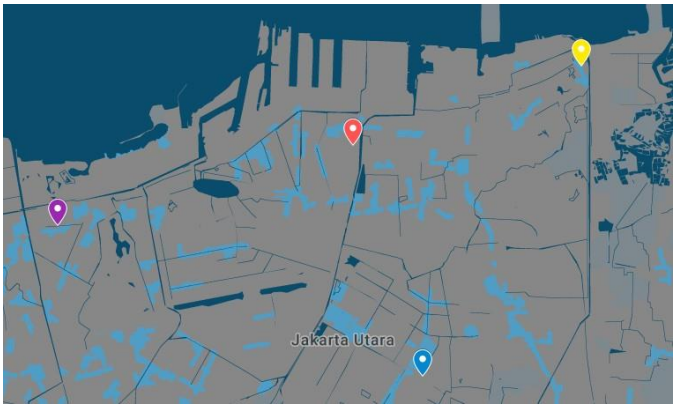
This study uses five illustrations of TSP cases in the population and civil registry offices in DKI Jakarta.

The main problem experienced is that the time period for distributing goods or documents is often delayed because they have to distribute goods or documents to several sub-districts without having to go through to the starting point. After analyzing there are three factors that cause the problem, including from the location, vehicles and technicians.

In terms of the location of the problems found, namely, the intended location has difficult terrain and alternative routes that are less familiar to people outside the installation location. Furthermore, namely from the side of the vehicle. The insufficient number of vehicles often causes queues. And the last is from the technician's side. The insufficient number of technicians also causes the installation time to delay, and sometimes technicians do not come from the installation location.

B. Dataset

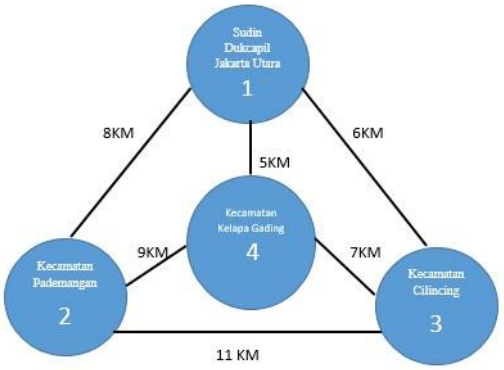
Data collection is done by determining which locations will be used as points.



location point

The red location point is the starting point, namely the North Jakarta Population and Civil Registry Office. The yellow dot is Cilincing District, the blue dot is Kelapa Gading District, and the last purple dot is the Pademangan District Office. The four locations are several sub-districts in North Jakarta.

C. Manual calculation of the tracking back algorithm



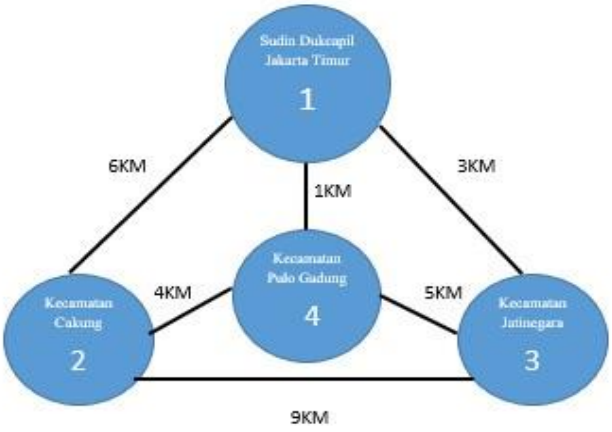
Population and Civil Registry Research Locations of North Jakarta

Manual of Calculation Results for Distance in kilometers:

No	Path	Path Length
1	Dukcapil-Pademangan-Kelapa Gading-Cilincing-Dukcapil	30 km
2	Dukcapil-Cilincing-Kelapa Gading-Pademangan-Dukcapil	30 km
3	Dukcapil-Kelapa Gading-Pademangan-Cilincing-Dukcapil	31 km
4	Dukcapil-Kelapa Gading-Cilincing-Pademnagan-Dukcapil	31 km

The closest route that can be taken by the North Jakarta population and civil registration messenger are 2 routes with a minimum value of 30 km.

No	Path	Path Length
1	Dukcapil-Pademangan-Kelapa Gading-Cilincing-Dukcapil	30 km
2	Dukcapil-Cilincing-Kelapa Gading-Pademangan-Dukcapil	30 km



Population and Civil Registry in East Jakarta.

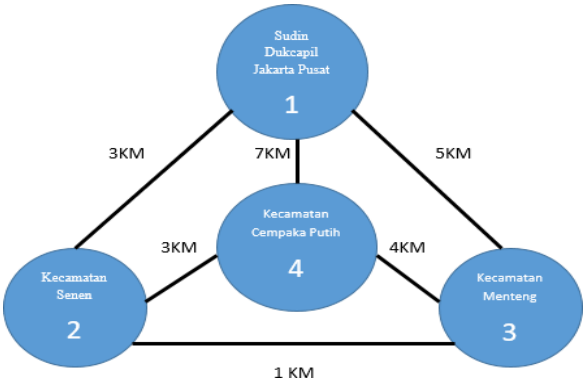
Manual of Calculation Results for Distance in

No	Path	Path Length
1	Dukcapil-Pulo Gadung-Jatinegara-Cakung-Dukcapil	21km
2	Dukcapil-Cakung-Pulo Gadung-Jatinegara-Dukcapil	18km
3	Dukcapil-Jatinegara-Pulo Gadung-Cakung-Dukcapil	18km
4	Dukcapil-Pulo Gadung-Cakung-Jatinegara-Dukcapil	17km

kilometers:

The closest route that can be taken by the population and civil registry messengers of East Jakarta is with a minimum value of 17 km.

No	Path	Path Length
1	Dukcapil-Pulo Gadung-Cakung-Jatinegara-Dukcapil	17 km



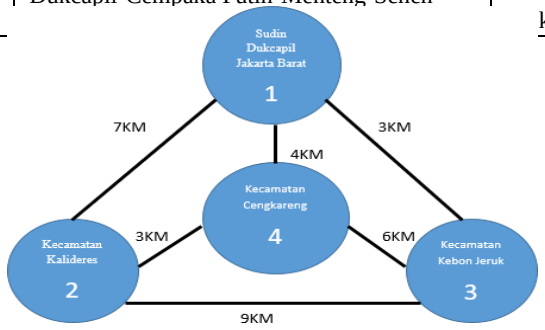
Population and Civil Registry in Central Jakarta.

Manual of Calculation Results for Distance in kilometers:

No	Path	Path Length
1	Dukcapil-Senen-Cempaka Putih- Menteng-Dukcapil	15 km
2	Dukcapil-Menteng-Cempaka Putih-Senen-Dukcapil	15 km
3	Dukcapil-Cempaka Putih-Senen- Menteng-Dukcapil	16 km
4	Dukcapil-Cempaka Putih-Menteng-Senen-Dukcapil	15km

The closest route that can be reached by Central Jakarta population and civil registration messenger are 3 routes, namely:

No	Path	Path Length
1	Dukcapil-Senen-Cempaka Putih-Menteng-Dukcapil	15km
2	Dukcapil-Menteng-Cempaka Putih-Senen-Dukcapil	15km
3	Dukcapil-Cempaka Putih-Menteng-Senen-	km



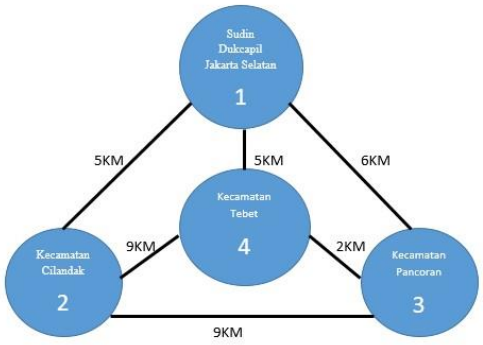
Population and Civil Registry in West Jakarta.

Manual of Calculation Results for Distance in kilometers:

No	Path	Path Length
1	Dukcapil-Kalideres-Cengkareng-Kebon Jeruk-Dukcapil	19 km
2	Dukcapil-Kebon Jeruk-Cengkareng-Kalideres-Dukcapil	19 km
3	Dukcapil-Cengkareng-Kalideres-Kebon Jeruk-Dukcapil	19 km
4	Dukcapil-Cengkareng-Kebon Jeruk-Kalideres-Dukcapil	26 km

The closest route that can be reached by the West Jakarta population and civil registration couriers are 3 routes with a minimum value of 19 km.

No	Path	Path Length
1	Dukcapil-Kalideres-Cengkareng-Kebon Jeruk-Dukcapil	19 km
2	Dukcapil-Kebon Jeruk-Cengkareng-Kalideres-Dukcapil	19 km
3	Dukcapil-Cengkareng-Kalideres-Kebon Jeruk-Dukcapil	19 km



Population and Civil Registry Research Locations in South Jakarta.

Manual of Calculation Results for Distance in kilometers:

No	Path	Path Length
1	Dukcapil-Cilandak-Tebet-Pancoran-Dukcapil	22km
2	Dukcapil-Pancoran-Tebet-Cilandak-Dukcapil	22km
3	Dukcapil-Tebet-Cilandak-Pancoran-Dukcapil	29km
4	Dukcapil-Tebet-Pancoran-Cilandak-Dukcapil	21km

No	Lintasan	Path Length
1	Dukcapil-Tebet-Pancoran-Cilandak-Dukcapil	21km

D. Implementation of TSP Backtracking Algorithm

Following is the results of calculations using Python3 for the DKI Jakarta Population and Civil Registry Office.

```
# Python3 implementation of the approach
V = 4
answer = []

# Function to find the minimum weight
# Hamiltonian Cycle
def tsp(graph, v, currPos, n, count, cost):

    # If last node is reached and it has
    # a link to the starting node i.e
    # the source then keep the minimum
    # value out of the total cost of
    # traversal and "ans"
    # Finally return to check for
    # more possible values
    if (count == n and graph[currPos][0]):
        answer.append(cost + graph[currPos][0])
        return

    # BACKTRACKING STEP
    # Loop to traverse the adjacency list
    # of currPos node and increasing the count
    # by 1 and cost by graph[currPos][i] value
    for i in range(n):
        if (v[i] == False and graph[currPos][i]):

            # Mark as visited
            v[i] = True
            tsp(graph, v, i, n, count + 1,
                cost + graph[currPos][i])

            # Mark ith node as unvisited
            v[i] = False

# Driver code

# n is the number of nodes i.e. V
if __name__ == '__main__':
    n = 4
    graph= [[ 0, 8, 6, 5 ],
            [ 8, 0, 11, 9 ],
            [ 6, 11, 0, 7 ],
            [ 5, 9, 7, 0 ]]

    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)

    # ans is the minimum weight Hamiltonian Cycle
    print(min(answer))
```

```
30
```

Counting the north Jakarta population and civil registry count using Python.

```
# Python3 implementation of the approach
V = 4
answer = []

# Function to find the minimum weight
# Hamiltonian Cycle
def tsp(graph, v, currPos, n, count, cost):

    # If last node is reached and it has
    # a link to the starting node i.e
    # the source then keep the minimum
    # value out of the total cost of
    # traversal and "ans"
    # Finally return to check for
    # more possible values
    if (count == n and graph[currPos][0]):
        answer.append(cost + graph[currPos][0])
        return

    # BACKTRACKING STEP
    # Loop to traverse the adjacency list
    # of currPos node and increasing the count
    # by 1 and cost by graph[currPos][i] value
    for i in range(n):
        if (v[i] == False and graph[currPos][i]):

            # Mark as visited
            v[i] = True
            tsp(graph, v, i, n, count + 1,
                cost + graph[currPos][i])

            # Mark ith node as unvisited
            v[i] = False
```

Counting Central Jakarta population and civil registry count using Python.

```
# Driver code

# n is the number of nodes i.e. V
if __name__ == '__main__':
    n = 4
    graph= [[ 0, 6, 3, 1 ],
            [ 6, 0, 9, 4 ],
            [ 3, 9, 0, 5 ],
            [ 1, 4, 5, 0 ]]

    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)

    # ans is the minimum weight Hamiltonian Cycle
    print(min(answer))
```

17

Population and civil registration Python calculations in East Jakarta.

```
# Python3 implementation of the approach
V = 4
answer = []

# Function to find the minimum weight
# Hamiltonian Cycle
def tsp(graph, v, currPos, n, count, cost):

    # If last node is reached and it has
    # a link to the starting node i.e
    # the source then keep the minimum
    # value out of the total cost of
    # traversal and "ans"
    # Finally return to check for
    # more possible values
    if (count == n and graph[currPos][0]):
        answer.append(cost + graph[currPos][0])
        return

    # BACKTRACKING STEP
    # Loop to traverse the adjacency list
    # of currPos node and increasing the count
    # by 1 and cost by graph[currPos][i] value
    for i in range(n):
        if (v[i] == False and graph[currPos][i]):

            # Mark as visited
            v[i] = True
            tsp(graph, v, i, n, count + 1,
                cost + graph[currPos][i])

            # Mark ith node as unvisited
            v[i] = False
```

```
# Driver code

# n is the number of nodes i.e. V
if __name__ == '__main__':
    n = 4
    graph= [[ 0, 3, 5, 7 ],
            [ 3, 0, 1, 3 ],
            [ 5, 1, 0, 4 ],
            [ 7, 3, 4, 0 ]]

    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)

    # ans is the minimum weight Hamiltonian Cycle
    print(min(answer))
```

15

```
# Python3 implementation of the approach
V = 4
answer = []

# Function to find the minimum weight
# Hamiltonian Cycle
def tsp(graph, v, currPos, n, count, cost):

    # If last node is reached and it has
    # a link to the starting node i.e
    # the source then keep the minimum
    # value out of the total cost of
    # traversal and "ans"
    # Finally return to check for
    # more possible values
    if (count == n and graph[currPos][0]):
        answer.append(cost + graph[currPos][0])
        return

    # BACKTRACKING STEP
    # Loop to traverse the adjacency list
    # of currPos node and increasing the count
    # by 1 and cost by graph[currPos][i] value
    for i in range(n):
        if (v[i] == False and graph[currPos][i]):

            # Mark as visited
            v[i] = True
            tsp(graph, v, i, n, count + 1,
                cost + graph[currPos][i])

            # Mark ith node as unvisited
            v[i] = False
```

```
# Driver code

# n is the number of nodes i.e. V
if __name__ == '__main__':
    n = 4
    graph= [[ 0, 7, 3, 4 ],
            [ 7, 0, 9, 3 ],
            [ 3, 9, 0, 6 ],
            [ 4, 3, 6, 0 ]]

    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)

    # ans is the minimum weight Hamiltonian Cycle
    print(min(answer))
```

19

Counting West Jakarta population and civil registry count using Python.

```
# Python3 implementation of the approach
V = 4
answer = []

# Function to find the minimum weight
# Hamiltonian Cycle
def tsp(graph, v, currPos, n, count, cost):

    # If last node is reached and it has
    # a link to the starting node i.e
    # the source then keep the minimum
    # value out of the total cost of
    # traversal and "ans"
    # Finally return to check for
    # more possible values
    if (count == n and graph[currPos][0]):
        answer.append(cost + graph[currPos][0])
        return

    # BACKTRACKING STEP
    # Loop to traverse the adjacency list
    # of currPos node and increasing the count
    # by 1 and cost by graph[currPos][i] value
    for i in range(n):
        if (v[i] == False and graph[currPos][i]):
```

```
# Driver code

# n is the number of nodes i.e. V
if __name__ == '__main__':
    n = 4
    graph= [[ 0, 5, 6, 5 ],
            [ 5, 0, 9, 9 ],
            [ 6, 9, 0, 2 ],
            [ 5, 9, 2, 0 ]]

    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)

    # ans is the minimum weight Hamiltonian Cycle
    print(min(answer))
```

21

Counting South Jakarta population and civil registry count using Python.

E. Algorithm Perfomance Evaluation

Based on the results of manual calculations and calculations using Python, the same results are obtained, namely:

Implementasi Shortest Route Pengiriman Berkas pada Caraka Dukcapil DKI Jakarta dengan Metode Backtracking TSP				
NO	NILAI ACUAN	NILAI PENGUKURAN	Error (nilai acuan-nilai pengukuran)	Error % (error/ nilai acuan x 100%)
1	30	30	0	0
2	17	17	0	0
3	19	19	0	0
4	21	21	0	0
5	15	15	0	0
			Total error %	0%
			Rata-rata error %	0%

The result is an accuracy of 100% - 0% = 100%

The acceptance rate of the algorithm performance test results is 100%. We will convert this result into a value like a table with an ordinal scale as follows:

No	Result	Scale	Description
1	0%-20%	1	Not very good
2	21%-40%	2	Not good
3	41%-60%	3	Enough
4	61%-80%	4	Good
5	81%-100%	5	Very good

Based on the table above, the test results fall into the very good performance category. So that it is suitable to be applied in the process of optimizing file distribution by the population messenger and civil registration of DKI Jakarta.

IV. CONCLUSIONS AND SUGGESTIONS

Suggestions for further research development are that this algorithm can be applied to a wider scope such as population and civil registration in several provinces or population and civil registration throughout Indonesia.

REFERENCES

[1] Hidayatulloh, Syarif, and Cisde Mulyadi. "Sistem pelayanan administrasi kependudukan desa candigatak berbasis web." *IT CIDA* 1.1 (2015).

[2] Siti Maryam, Neneng. "Mewujudkan good governance melalui pelayanan publik." *JIPSI-Jurnal Ilmu Politik Dan Komunikasi UNIKOM* 6 (2017).

[3] Meliantari, Kadek. "Optimasi Distribusi Produk Menggunakan Metode Cheapest Insertion Heuristic Berbasis Web ." Universitas Udayana (2018).

[4] Vanrika, A. R. (2019). *Sistem Pencarian Rute Distribusi Terpendek Menggunakan Algoritma Genetika (Studi Kasus Distributor Sari Roti Yogyakarta)* (Doctoral dissertation, Universitas Mercu Buana Yogyakarta).

[5] Sihotang, Niko Saputra. *ANALISIS PENENTUAN RUTE DENGAN MEMPERTIMBANGKAN PENGGUNAAN TOL CIKOPO–PALIMANAN (Studi Kasus Di Pt Pos Indonesia (Persero))*. Diss. Universitas Widyatama, 2016.

[6] Paillin, D. B., and Filinda Sosebeko. "Penentuan Rute Optimal Distribusi Produk Nestle Dengan Metode Traveling Salesman Problem (TSP)(Studi Kasus: PT. Paris Jaya Mandiri)." *Arika* 11.1 (2017): 35-44.

[7] Dicky, Moriza. Rute Pendistribusian Air Mineral Dalam Kemasan Menggunakan Metode Nearest Neightbour dan Branch and Bound di PT. Agronesia BMC. Itenas, Bandung, 2016

[8] S. H. . Herho, "Tutorial Pemrograman Python 2 Untuk Pemula."

[9] Y. Supardi and Y. Syarief, *Tip dan Trik Program Database Python*. Jakarta: Elex Media Komputindo, 2020.

[10] S. A. Qutsiah, M. K. Sophan, and Y. F. Hendrawan, "DATAR MENGGUNAKAN PYTHON PADA PERANGKAT," vol. XI, 2016.