**Lampiran 6 Perhitungan Rute Optimisasi Dengan *Ant Colony Optimization* menggunakan *Matlab***

Dalam mengoptimisasi rute menggunakan *software Matlab* digunakan pendekatan *Vechile Routing Problem* (VRP), berikut dapat dilihat langkah-langkah untuk mendapatkan hasil optimisasi rute dari *software Matlab* :

1. *In*i*tial Solution* berupa matriks jarak dari rute yang diperoleh dari metode sebelumnya *insertion heuristic*
2. Input Matriks Jarak ke dalam software seperti formulasi di bawah. Lalu jalankan program matlab. (Matriks Jarak Kendaraan atau 1 Rute ke 1)

% generating distace between cities matrix.

%for i=1:n

% for j=1:n

% d(i,j)=sqrt((x(i)-x(j))^2+(y(i)-y(j))^2);

% end

%end

d=[0 5 10;

5 0 7;

10 7 0];

1. Input Parameter-Parameter yang di gunakan dalam proses optimisasi *ant colony optimization*

% function myaco(num\_of\_nodes,num\_of\_ants, max\_iteration)

function myaco()

% inputs

miter=10; % banyak iterasi

m=10; % banyak semut

n=3; %jumlah kota

% parameters

e=.5; % evaporation coefficient.

alpha=1; % effect of ants' sight.

beta=3; % trace's effect.

t=0.01\*ones(n); % primary tracing.

Formulasi program utama ACO

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Solving VRP Problem Using ACO

% -------------------------------------------------------------------------

% By : Wahyudin

% Contact: wahyudinftunpas@gmail.com Last update: juli 15, 2019

% -------------------------------------------------------------------------

% This program is developed to find shortest path (minimum cost)between

% some cities.

%

% There are 4 parts in this program:

% 1.Main program of ACO (myaco.m)

% 2.Function to generate solution (ant\_tour.m)

% 3.Function to calculate the cost (distance) (calculate\_cost.m)

% 4.Function to update the traces (update\_the\_trace.m)

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% The Program Start Here

%\*------------------------------------------------------------------------

% function myaco(num\_of\_nodes,num\_of\_ants, max\_iteration)

function myaco()

% inputs

miter=10; % banyak iterasi

m=10; % banyak semut

n=3; %jumlah kota

% parameters

e=.5; % evaporation coefficient.

alpha=1; % effect of ants' sight.

beta=3; % trace's effect.

t=0.01\*ones(n); % primary tracing.

el=.97; % common cost elimination.

% -------------------------------------------------------------------------

% Generate coordinates of cities and plot

for i=1:n

x(i)=rand\*20;

y(i)=rand\*20;

end

subplot(3,1,1);

plot(x,y,'o','MarkerFaceColor','k','MarkerEdgeColor','b','MarkerSize',10);

title('Titik Koordinat Kota (Gudang Lini 3)');

xlabel('x (km)');

ylabel('y (km)');

% generating distace between cities matrix.

%for i=1:n

% for j=1:n

% d(i,j)=sqrt((x(i)-x(j))^2+(y(i)-y(j))^2);

% end

%end

d=[0 5 10;

5 0 7;

10 7 0];

% generating sight matrix.

for i=1:n

for j=1:n

if d(i,j)==0

h(i,j)=0;

else

h(i,j)=1/d(i,j);

end

end

end

h=h

% ------------------------------------------------------------------------

% Main Algorithm: ACO Meta heuristic procedure

% a. Probabilistic solution construction biased by

% pheromone trails, without forward pheromone

% updating

% b. Deterministic backward path with loop elimination

% and with pheromone updating--> update\_the\_trace

% c. Evaluation of the quality of the solutions

% generated and use of the solution quality in

% determining the quantity of pheromone to deposit-->calculate\_cost

% -------------------------------------------------------------------------

for i=1:miter

% Step 1: Forward ants and solution construction

% Generate places for each ant

for j=1:m

start\_places(j,1)=fix(1+rand\*(n-1));

end

% Step 2:probabilistic solution contruction

[tour]=ant\_tour(start\_places,m,n,h,t,alpha,beta);

tour=horzcat(tour,tour(:,1));

% Step 3: Calculate the cost --> total distace

[cost,f]=calculate\_cost(m,n,d,tour,el);

[t]=update\_the\_trace(m,n,t,tour,f,e);

average\_cost(i)=mean(cost);

% Step 4: Determine the best route

[min\_cost(i),best\_index]=min(cost);

besttour(i,:)=tour(best\_index,:);

iteration(i)=i;

end

% -------------------------------------------------------------------------

% Plot Average of tour distance vs Number of Iterations

subplot(3,1,2);plot(iteration,average\_cost);

title('Rata-Rata Jarak vs Jumlah Iterasi');

xlabel('iteration');

ylabel('Jarak(Km)');

% Plot the best route

[k,l]=min(min\_cost); % jarak

for i=1:n+1

X(i)=x(besttour(l,i));

Y(i)=y(besttour(l,i));

end

subplot(3,1,3);plot(X,Y,'--o',...

'MarkerEdgeColor','k',...

'MarkerFaceColor','g',...

'MarkerSize',10)

xlabel('x (km)');ylabel('y (km)');

title(['Total Jarak(Km)= ',num2str(k)]);

fprintf('\nTotal Jarak = %d\n',k);

fprintf('\nRute Optimasi ACO = 0, 3, 2, 0\n');

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Ending of Program

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Formulasi Ant Tour ACO

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Function to calculate ants tour matrix during one cycle

%--------------------------------------------------------------------------

% The function Start Here

%--------------------------------------------------------------------------

function [new\_places]=ant\_tour(start\_places,m,n,h,t,alpha,beta);

for i=1:m

mh=h;

for j=1:n-1

c=start\_places(i,j);

mh(:,c)=0;

temp=(t(c,:).^beta).\*(mh(c,:).^alpha);

s=(sum(temp));

p=(1/s).\*temp;

r=rand;

s=0;

for k=1:n

s=s+p(k);

if r<=s

start\_places(i,j+1)=k;

break

end

end

end

end

new\_places=start\_places;

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Ending of Function

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Formulasi *update trace pheromone*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Function to update the traces

%--------------------------------------------------------------------------

% The function Start Here

%--------------------------------------------------------------------------

function [t]=update\_the\_trace(m,n,t,tour,f,e);

for i=1:m

for j=1:n

dt=1/f(i);

t(tour(i,j),tour(i,j+1))=(1-e)\*t(tour(i,j),tour(i,j+1))+dt;

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Ending of Function

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Formulasi *Calculate Cost* ACO

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Function to calculate cost (distance) of ants' touring

%--------------------------------------------------------------------------

% The function Start Here

%--------------------------------------------------------------------------

function [cost,f]=calculate\_cost(m,n,d,at,el);

for i=1:m

s=0;

for j=1:n

s=s+d(at(i,j),at(i,j+1));

end

f(i)=s;

end

cost=f;

f=f-el\*min(f);

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Ending of Function

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*