MULTIDEPOT VEHICLE ROUTING PROBLEM USING GENETIC ALGORITHM

By

SOURJALI DAS, UN ROLL-11700114074

SANCHARI DAS, UN ROLL-11700114058

ARNAB MONDAL, UN ROLL-11700114016

SARMISTHA BHATTACHARYYA, UN ROLL-11700114061

UNDER THE GUIDANCE OF

Mr. HARINANDAN TUNGA

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

Session 2015-2016

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

[Affiliated to West Bengal University of Technology]

CANAL SOUTH ROAD, BELIAGHATA, KOLKATA-700015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

श्रमम् बिना न किमपि साध्यम्

TO WHOM IT MAY CONCERN

I hereby recommend that the Project entitled MULTIDEPOT VEHICLE ROUTING PROBLEM USING GENETIC ALGORITHM' prepared under my supervision by Sourjali Das(univ.roll-11700114074,class roll-cse2014/019) Sanchari Das(univ.roll-11700114058,class roll-cse2014/034) Sarmistha Bhattacharyya(univ.roll-11700114061,class roll-cse2014/036) Arnab Mondal(univ.roll-11700114016,class roll-cse2014/043) of B.Tech(8 th Semester), may be accepted in fulfilment for the degree of Bachelor of Technology in Computer Science & Engineering under Maulana Abul Kalam Azad University of Technology.

……………………………………………………………………………………

Harinandan Tunga, Associate Professor
Project Supervisor
Department of Computer Science and Engineering
RCC Institute of information Technology

Countersigned:

………………………………………………………………………………

Head
Department of Computer Sc. & Engineering, RCC Institute of Information Technology Kolkata-700015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

CERTIFICATE OF APPROVAL

The foregoing Project is hereby accepted as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptances a pre requisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in, but approve the project only for the purpose for which it is submitted.

FINAL EXAMINATION FOR EVALUATION OF PROJECT

1.______________________________

2.______________________________

(Signature of Examiners)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCC INSTITUTE OF INFORMATION TECHNOLOGY

We express our sincere gratitude to Harinandan Tunga, Associate Professor of Department of Computer Science & Engineering, RCCIIT for providing his valuable Time for us to take up this topic as a Project.

Last but not the least I would like to express my gratitude to Mr. Harinandan Tunga, Mr. Sk. Mazarul Islam of our department who helped us in their own way when ever needed.

Sourjali Das

Sanchari Das

Sarmistha Bhattyacharyya

Arnab Mondal

MULTIDEPOT VEHICLE ROUTING PROBLEM USING GENETIC ALGORITHM

TABLEOFCONTENTS

Page NO.

1.Introduction

The Multi-Depot Vehicle Routing Problem (MDVRP), an extension of classical VRP, is a NP-hard problem for simultaneously determining the routes for several vehicles from multiple depots to a set of customers and then return to the same depot. The objective of the problem is to find routes for vehicles to service all the customers at a minimal cost in terms of number of routes and total travel distance, without violating the capacity and travel time constraints of the vehicles. The solution to the MDVRP, in this paper, is obtained through Genetic Algorithm (GA). The customers are grouped based on distance to their nearest depots and then routed with Clarke and Wright saving method. Further the routes are scheduled and optimized using GA.

In a MDVRP, the number and locations of the depots are predetermined. Each depot is large enough to store all the products ordered by the customers. Each vehicle starts and finishes at the same depot. The location and demand of each customer is also known in advance and each customer is visited by a vehicle exactly once.

2.REVIEW OF LITERATURE:

This section briefs the existing work related to MDVRP solutions by various heuristic methods. Research on MDVRPs is quite limited compared with the extensive literature on simple VRPs and their variants. Salhi et Al., [5] addressed a multi-level composite heuristic with two reduction tests. The initial feasible solutions were constructed in the first level, while the intra-depot and the inter-depot routes were improved in the second and third levels. Wu et Al., [1] reports a simulated annealing (SA) heuristic for solving the multi-depot location routing problem (MDLRP). Giosa et Al., [4] developed a “cluster first, route second” strategy for the MDVRP with Time Windows (MDVRPTW), an extension of the MDVRP. Considering the operational nature of the MDVRPTW, this paper, focuses more on the computational time. Haghani et Al., [2] presented a formulation for solving the dynamic vehicle routing problem with time-dependent travel times using Genetic Algorithm. Nagy et al. [8] proposed several enhancements to an integrated heuristic method for solving the MDVRP. Lee et al. [6] handled the MDVRP by formulating the problem as deterministic dynamic programming (DP) with finite-state and action spaces, and then using a shortest path heuristic search algorithm. Creviera et Al., [7] proposed a heuristic combining tabu search method, and integer programming for multi-depot vehicle routing problem in which vehicles may be replenished at intermediate depots along their route.

3.OBJECTIVE OF THE PROJECT

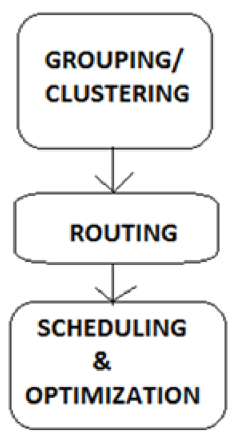
The objective of that project is to find routes for vehicles to give service to all the customers at a minimal cost in terms of number of routes and total travel distance, without violating the capacity of the vehicles. The purpose of the project is to explore a real world vehicle routing problem, that has multiple depot with a heterogeneous group of vehicle that are available to pick-up and deliver with varying location. Both the completion time and the no of delivers utilized, are analyzed here. In general, MDVRP is to minimize the total delivery distance or time spent in serving all customers thus utilizing efficient amount of vehicles. The solution to the MDVRP, is obtained through Genetic Algorithm (GA).The customers are grouped , based on the distance to the nearest depots and then routed. Further the routes are scheduled and optimized using GA.

Genetic Algorithms (GA) is based on a parallel search mechanism, which makes it more efficient than other classical optimization techniques such as branch and bound. The basic idea of GA is to maintain a population of candidate solutions that evolves under selective pressure. The GA can avoid getting trapped in a local optimum by tuning the genetic operators, crossover and mutation. Due to its high potential for global optimization, GA has received great attention in solving multi-depot vehicle routing problems. GA imitates the mechanism of natural selection and the survival of the fittest as witnessed in natural evolution.

The details of the chromosome representation, fitness evaluation, and other GA components used for the MDVRP are provided here. When the GA is initialized, a simple clustering strategy is employed to assign each customer to an initial depot. The clustering strategy assigns the customers one by one to a given depot until all the customers have been assigned.

COMPUTATIONAL PATH OF MDVRP

The decision making stages are classified into grouping, routing, scheduling and optimization



GROUPING/CLUSTERING:

Initially each customer is assigned to the nearest depot in terms of Euclidean distance. This strategy is simple, fast, and viable since no capacity limit is imposed on each depot

ROUTING:

The actual correspondence between a chromosome and the routes is achieved by route scheduler. The scheduler is designed to consider the feasibility of the routes checking the capacity constarints.

SCHEDULING AND OPTIMIZATION:

Routes are scheduled and optimized with the help of Genetic Algorithm.GA is performed by selection, crossover, mutation.

A. Selection
During each generation, the parents are selected for mating and reproduction. In this MDVRP application, we use tournament selection to generate new individuals in the population. This selection strategy is based on fitness evaluation.

B. Crossover

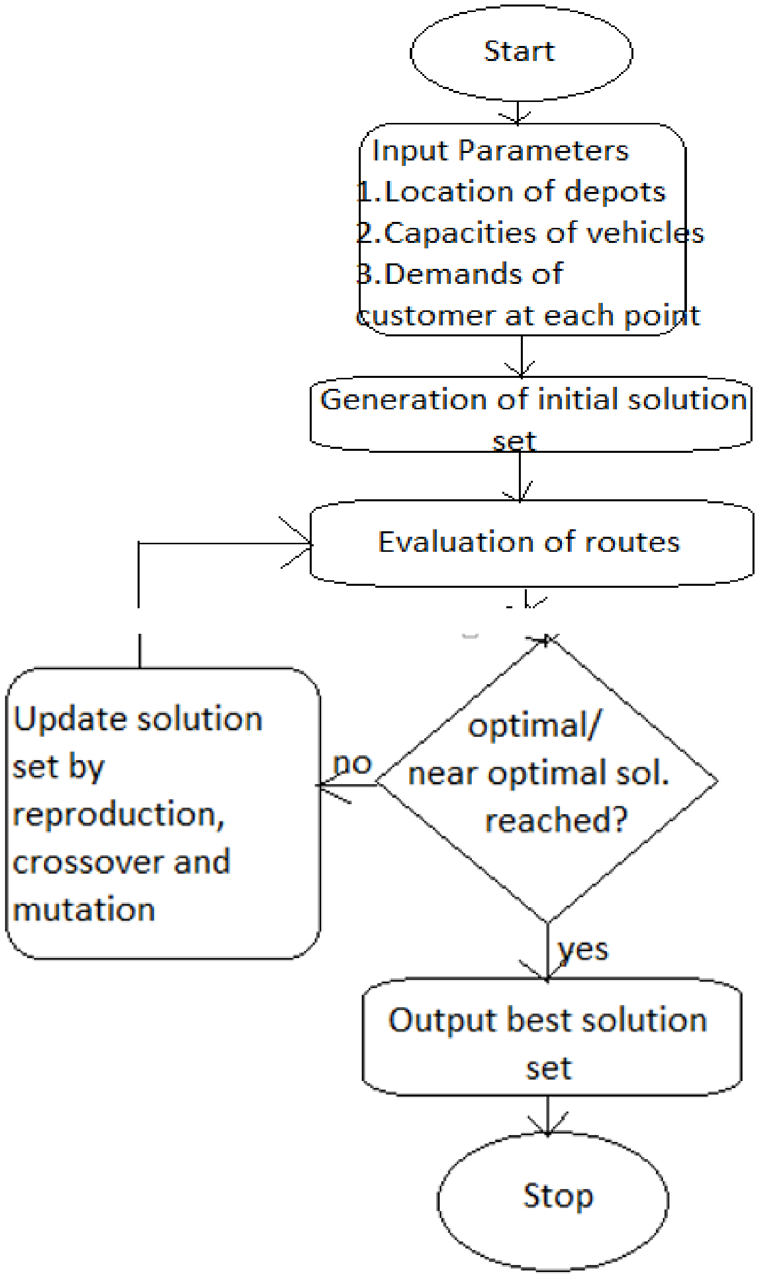
A problem specific crossover technique, the greedy Crossover is used here for multi depot vehicle routing problem .

C.Mutation

The 2-opt mutation is used in finding an MDVRP solution using GA. 2 links from a tour is removed & reconnected from opposite direction.

4.FLOW CHART OF MDVRP

In MDVRP, the number and locations of the depots are predetermined. Each depot is large enough to store all the products ordered by the customers. Each vehicle starts and finishes at the same depot. The location and demand of each customer is also known in advance and each customer is visited by a vehicle exactly once.



5.METHODOLOGY FOR IMPLEMENTATION(ALGORITHM)

Step 1: **CLUSTERING/GROUPING**

1.Input the number of depots and their corresponding co-ordinates.

2.For k=1 to no.of.depots

2.1. Store the co-ordinates of the depots in the array depot[k]

3. Input the number of cities and their corresponding co-ordinates.

4. For c=1 to no.of cities

4.1. store the co-ordinates of the cities in the array city[c]

5. For c=1 to no. of. Cities

5.1. For k=1 to no. of depots

5.1.1. Calculate the distance Dck and store the value in an array a[k]

End for

5.2. Select the depot co-ordinate having min(a[k]).

5.3.Store the city co ordinate(co ordinate of city ci) in an array of index k(depot index) group[k]

End for

Step 2: ROUTING

1. **Input no. of. Vehicle in each depot and store the input in vehicle[k]{k=1 to no.of depot}**
2. **Input the capacity of the vehicles(assuming that capacity of all the vehicles are same) and store the value in C.**
3. **Input demand of each city and store the input in demand[c]{c=1 to no.of.cities}**
4. **For i in 1 to (no of depot)**
5. **Sum=0,t=0,p=0,route[t]={co ordinate of the depot i.e depot[i]}**

6.1.For j in 1 to (no of cities)

6.1.1If(j=group[i])

6.1.1.1.sum=sum+demand[j]

6.1.1.2.if(sum> C(capacity of vehicle)

6.1.1.3. p++,t=0;

Break;

6.1.1.3.else

depot[p][++t] =group[j];

end if;

end for;

7.for i in 1 to no of cities

7.1. for j in 1 to no of car

7.1.1. for k in 1

7.1.1.1.

// depot[p][q] is our stored route for each vehicles

Step 3: SCHEDULING AND OPTIMIZATION(USING GA)

1. **Input the no. of generation= g, cost matric c[][] for each depot**
2. **t=0,p[][];**
3. **For i in 1 to no of generation**

- **[tournament selection]**

3.1. For i in 1 to (c[t][]!)

3.1.1.populate the fitness value of each route from the cost matrix;

3.2. divide the population in 2 groups

Randomly

3.3. select one route from each group which have minimum fitness value(i.e minimum cost)

(these routes are parent 1 (p[1][])and parent 2(p[2][]))

[greedy crossover]

```
3.4. for i in 1 to 2
        3.4.1.for j in 1 to (no of nodes in the routes)
        3.4.2. create c[][],t=1; // c[][]=child route
                c[j][t++]=p[i][j];
        3.4.3. search the city c[j][t] in c[j+1][] and let the searched city is c[j+1][p]
        3.4.3. if (dist(c[j][t-1],c[j][t])<dist(c[j+1][p],c[j+1][p+1])
                3.4.3.1. put c[j][t++]=c[j][t+2]
                                Else
                                Put c[j][t++]=c[j+1][p+1]
                        End if
        End for
```

[2 opt mutation]

```
3.5.For i in 1 to 2

3.6. For j in (no of cities in each routes) to 1

    3.6.1.k=1;

    3.6.2.Rev[i][k++]=c[i][j]

    3.6.3. If (fitness(c[i][]>fitness[i][])

        3.6.3.1.c[i][]=rev[i][]

        end for

end for (no of generation calculator)
```

6.MATHEMATICAL ILLUSTRATION

Depot	positon of depots	No of vehicles in depot	capacity of each
Depot A	(0 , 0)	2	50
Depot B	(9 , 9)	2	30

Table 1

customer point	location	demand
1	(1 ,1)	20
2	(2 ,2)	15
3	(3 ,3)	30
4	(4 ,4)	15
5	(5, 5)	15
6	(6, 6)	10
7	(7, 7)	12
8	(8, 8)	23

table 2

Suppose we will have to solve the above problem by MDVRP using GA.

1.Grouping/Clustering:

We will have to find out the Euclidian distance of each node from each depot.

$D(c_i,k)=\sqrt{(x_{c_i}-x_k)^2+(y_{c_i}-y_k)^2}$ where ci represent cutomer and k represent depot.

Now

-If D(Ci,A)<D(Ci,B), then customer Ci is assigned to depot A
-If D(Ci,A)>D(Ci,B), then customer Ci is assigned to depot B
-If D(Ci,A)=D(Ci,B), then customer Ci is assigned to a depot chosen arbitrarily between A and B

So in the above problem,

D(1,A)=√2, D(1,B)=8√2.

As D(1,A)<D(1,B), so 1 is assigned to depot A.

D(2,A)=2√2, D(2,B)=7√2.

As D(2,A)<D(2,B), so 2 is assigned to depot A.

D(3,A)=3√2, D(3,B)=6√2.

As D(3,A)<D(3,B), so 3 is assigned to depot A.

D(4,A)=4√2, D(4,B)=5√2.

As D(4,A)<D(4,B), so 4 is assigned to depot A.

D(5,A)=5√2, D(5,B)=4√2.

As D(5,A)>D(5,B), so 5 is assigned to depot **B**.

D(6,A)=6√2, D(6,B)=3√2.

As D(6,A)>D(6,B), so 6 is assigned to depot **B**.

D(7,A)=7√2, D(7,B)=2√2.

As D(7,A)>D(7,B), so 7 is assigned to depot **B**.

D(8,A)=8√2, D(8,B)=√2.

As D(8,A)>D(8,B), so 8 is assigned to depot **B**.

So point 1,2,3,4 is assigned to Depot A and point 5,6,7,8 is assigned to depot B.

2.Routing

	Depot A	1	2	3	4
Depot A		√2	2√2	3√2	4√2
1	√2		√2	2√2	3√2
2	2√2	√2		√2	2√2
3	3√2	2√2	√2		√2
4	4√2	3√2	2√2	√2	

distance matrix of Depot A with the clusterder points

Table 3

in this stage, routes are determined from the distance matrix . routes always start from depots and end to that particular depot, considering the demands of customers at each point.

From depot A, we will go to that point having mimimun distance without violating the capacity constraints of the vehicle(capacity of each vehicle is 50)

So depot A->point 1(capacity=20<50) so accepted

Depot A->1->2(capacity=20+15=35<50) so accepted

Depot 1->1->2->3(capacity=20+15+30=65>50) rejected as its violating the capacity constraint

depot A ->1->2->4(capacity=20+15+15=50<=50) accepted

so the first route is

A->1->2->4->A

2 nd route will be

A->3->A(capacity=30<=50)accepted

	B Depot	5	6	7	8
B Depot		$4\sqrt{2}$	$3\sqrt{2}$	$2\sqrt{2}$	$\sqrt{2}$
5	$4\sqrt{2}$		$3\sqrt{2}$	$2\sqrt{2}$	$\sqrt{2}$
6	$3\sqrt{2}$	$\sqrt{2}$		$\sqrt{2}$	$2\sqrt{2}$
7	$2\sqrt{2}$	$2\sqrt{2}$	$\sqrt{2}$		$\sqrt{2}$
8	$\sqrt{2}$	$3\sqrt{2}$	$2\sqrt{2}$	$\sqrt{2}$	

distance matrix of Depot B with the clusterder points

Table 4

From depot B, we will go to that point having mimimun distance without violating the capacity constraints of the vehicle(capacity of each vehicle is 35)

So in this case,from depot B we will go to point 8 first

B->8(capacity=23)accepted

B->8->7(capacity=23+12=37>35) rejected

B->8->6(capacity=23+10=33<=35)accepted

So Route *B->8->6->B*

Another route

B->7(because next shortest distance is between depot B and point 7)

B->7(capacity=12<=35)accepted

B->7->5(capacity=12+15=27<=35)accepted

So next route is *B->7->5->B*

3. Scheduling and optimization
   For Depot A, vehicle 1
   Route:A->1->2->4->A
   Initial population 3!=6
   Now we will calculate the fitness of each chromosomes from the distance matrix(table 3)

chromosomes	fitness value
A->1->2->4->A	$\sqrt{2}+\sqrt{2}+2\sqrt{2}+4\sqrt{2}=8\sqrt{2}$
A->1->4->2->A	$\sqrt{2}+3\sqrt{2}+3\sqrt{2}+2\sqrt{2}=9\sqrt{2}$
A->2->1->4->A	$2\sqrt{2}+2\sqrt{2}+3\sqrt{2}+4\sqrt{2}=11\sqrt{2}$
A->2->4->1->A	$2\sqrt{2}+2\sqrt{2}+4\sqrt{2}+\sqrt{2}=9\sqrt{2}$
A->4->1->2->A	$4\sqrt{2}+4\sqrt{2}+\sqrt{2}+2\sqrt{2}=11\sqrt{2}$
A->4->2->1->A	$4\sqrt{2}+3\sqrt{2}+2\sqrt{2}+\sqrt{2}=10\sqrt{2}$

Tournament selection:
In this process 6 chromosomes are divided into 2 groups randomly.
Suppose 1st team have (1,3,6)th chromosomes and other team have (2,4,5)th chromosomes.
From the 1st team/group , we will select that chromosome which have best fitnessvalue(i.e minimum route cost) which will be the parent of the next generation.
So here from team 1,chromosome 1 will be selected as it have less route cost

So *P1:A->1->2->4->A*

Similarly from 2nd team,either 2 or 4 will be selected as both of them have same route cost.We are selecting 4.

So *P2: A->2->4->1->A*

Greedy crossover:

P1: A->1->2->4->A and P2: A->2->4->1->A

For P1:

A->1, now we will check if the distance between point 1->2 is miminum or 1->A is minimum . From distance matrix (table 3)

So 1->2 (V2) and 1->A(2V2) so the route is A->1->2

Similar step will be followed for determining the next node

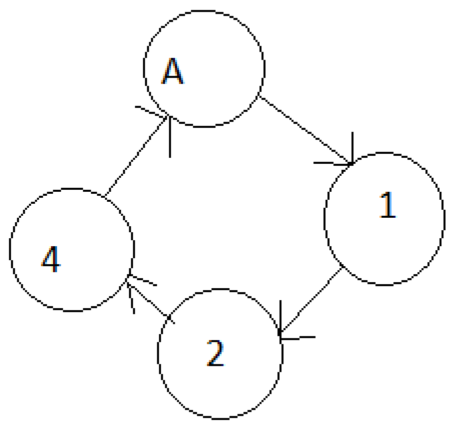
So after crossover

Child1 : A->1->2->4->A

Child2 :A->2->4->1->A

2-opt mutation:

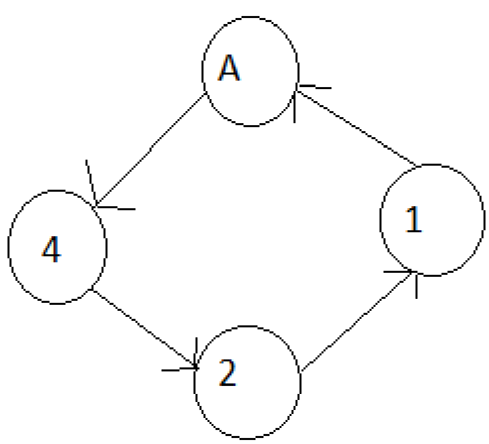
Our child1: A->1->2->4->A



So the route is like

Whose fitness=8√2

Now we will change the direction of the path



Now its fitness=$9\sqrt{2}$
As the previous one have better fitness so previous one will be selected.

Similarly, mutation will be applied to child2:A->2->4->1->A
And it will be converted into A->1->4->2->A.

So after end of the 1st generation,
We will get
P1: A->1->2->4->A
P2: A->1->4->2->A.
In this way,GA will be followed unless and until we get optimal solution set.

7. **IMPLEMENTATION DETAILS**

In this project, 3 depots are taken and 41 customer location points are taken as input. Also capacity of each vehicle and customer demands at each location, is taken as input. At first we have created chromosomes (here routes).Then all the chromosomes are divided into 2 groups randomely and from each groups, the chromosome having highest fitness value is being selected (Tournamnet Selection) as parent for the crossover. We have used greedy crossover in our project. After doing crossover of parent 1 and 2, we get child 1 and child 2. Then we have done 2-opt mutation between child 1 and child2. Then fitness value is calculated for child1 and child 2. If the fitness value of the childs are improved , then they are accepted as parents for the next generation and else, if the fitness value of the childs are not better than the parents, that means we reached the optimal solution and those routes(chromosomes) will be shown as our desired output.

8.RESULT/SAMPLE OUTPUT

Depot Name	Vehicle No.	Route	Loads of vehicles
Depot 0	vehicle 1	0->27->26->25->0	47
Depot 1	vehicle 1	1->30->29->5->28->1	75
	vehicle 2	1->4->3->12->1	70
	vehicle 3	1->16->42->1	64
	vehicle 4	1->17->21->36->23->1	70
	vehicle 5	1->13->1	37
	vehicle 6	1->14->24->34->37->31->1	79
	vehicle 7	1->22->15->18->35->1	80
Depot 2	vehicle 1	2->41->40->38->39->2	66
	vehicle 2	2->33->20->19->32->2	80
	vehicle 3	2->43->8->10->11->2	79
	vehicle 4	2->9->7->6->2	66

9.CONCLUSION

The problem instance were initially grouped to assign the customers to their corresponding depots based on the Euclidean distance. The customers of the same depot are assigned to several routes in the routing phase and each route is sequenced in scheduling phase.

The scheduled routes are optimized using GA. The simulation results of the proposed heuristic algorithms were compared in terms of depot's route length, optimal route, optimal distance, computational time, average distance, and number of vehicles. It was observed from the conducted experiments, the performance of GA was exceptional and the computational time proved that GA is much faster for solving of multi-depot vehicle routing problem. In future, efforts will be taken to impose more realistic constraints on the problem structure and large size real-time problems would be attempted by the proposed methodology. The GA studied in this paper has an advantage of its simplicity and yet gives efficient solution quality with respect to existing GA and non-GA techniques. A possible future work is to extend this work to the multi-depot vehicle routing problem with time windows and to better tune the GA parameters.

APPENDIX(program source code)

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <ctime>
#include <algorithm>
#include <cmath>

#define depot_num 3
#define N 43  // no of points including 3 depots(0-43)
#define MAXN 0x7fffffff
#define L 80 //distance limit for each vehicle
using namespace std;

double dis[N+1][N+1];
double c[N+1];
double total_fitness;
const int worst_dis=1000000;
int index;
class chromosome
{
public:
        int path[N+1];
        int pre[N+1];
        double fitness;
        void assign(int arr[])
        {
```

```
        for (int i=0;i<=N;i++)
            path[i]=arr[i];
    }
    double split(int s,int e)
    {
        double v[N+1];
        v[path[s]]=0;
        pre[path[s]]=path[s];
        for (int i=s+1;i<=e;i++) v[path[i]]=MAXN;
        for (int i=s+1;i<=e;i++)
        {
            double cost=0;
            double load=0;
            int j=i;
            while (j<=e && load<=L)
            {
                load+=c[path[j]];
                if (i==j) cost=dis[path[s]][path[j]]+dis[path[j]][path[s]];
                else cost=cost-dis[path[s]][path[j-1]]+dis[path[j-
1]][path[j]]+dis[path[j]][path[s]];
                if (load<=L)
                {
                    if (v[path[i-1]]+cost<v[path[j]])
                    {
                        v[path[j]]=v[path[i-1]]+cost;
                        pre[path[j]]=path[i-1];
                    }
                    j++;
                }
            }
        }
        return v[path[e]];
```

```
	}
	void cal_fitness()
	{
		int pre=0;
		fitness=0;

		for (int i=1;i<=N;i++)
		{
			if (path[i]<depot_num)
			{
				fitness+=split(pre,i-1);
				pre=i;
			}
		}
		fitness+=split(pre,N);
		fitness=worst_dis-fitness;
	}
};
vector<chromosome> population[2];
vector<chromosome> temp;

void random_init(int arr[])
{
	int A[N];
	for (int i=0;i<N;i++)
		A[i]=i+1;
	int length=N;
	arr[0]=0;
	for (int i=1;i<=N;i++)
	{
		int r=rand()%length;
```

```
            arr[i]=A[r];
            int temp=A[r];
            A[r]=A[length-1];
            A[length-1]=temp;
            length--;
        }
        arr[N]=A[0];
}

const int p_size=30;
void init_population()
{
        srand((unsigned)time(NULL));
        chromosome chr;
        int p[N+1];
  index=0;
        //population size is 4
        for (int i=0;i<p_size;i++)
        {
            random_init(p);
            //printf("Initial Population:");
            //for (int j=0;j<=N;j++) printf("%d ",p[j]);printf("\n");
            chr.assign(p);
            population[index].push_back(chr);
        }
}

void update_fitness()
{
        total_fitness=0;
        for (int i=0;i<(int)population[index].size();i++)
```

```
    {
        population[index][i].cal_fitness();
        total_fitness+=population[index][i].fitness;

    }

}

bool cmp(chromosome a,chromosome b)
{
    return a.fitness<b.fitness;
}

//generate a number in range from 0 to end
int random(int end)
{
  return rand() % (end + 1);
}

void convert(int &a,int &b)
{
    if (a>b)
    {
        int temp=a;
        a=b;
        b=temp;
    }
}

void crossover(chromosome &a,chromosome &b,int index1,int index2)
{
```

```
int first_pos=rand()%(N-1)+1; //generate a number in range from 1 to N-1
int second_pos;
//generate a number which is different from first number
while ((second_pos=rand()%(N-1)+1)==first_pos);
convert(first_pos,second_pos);
chromosome new1,new2;
new1.path[0]=new2.path[0]=0;
bool hash1[N+1]={false};
bool hash2[N+1]={false};
for (int i=first_pos+1;i<=second_pos;i++)
{
	new1.path[i]=a.path[i];
	hash1[a.path[i]]=true;
	new2.path[i]=b.path[i];
	hash2[b.path[i]]=true;
}
int hashapos=1;
int hashbpos=1;
for (int i=1;i<=first_pos;i++)
{
	while (hash1[b.path[hashbpos]])
		hashbpos++;
	new1.path[i]=b.path[hashbpos++];
	while (hash2[a.path[hashapos]])
		hashapos++;
	new2.path[i]=a.path[hashapos++];
}
for (int i=second_pos+1;i<=N;i++)
{
	while (hash1[b.path[hashbpos]])
		hashbpos++;
```

```
            new1.path[i]=b.path[hashbpos++];
            while (hash2[a.path[hashapos]])
                hashapos++;
            new2.path[i]=a.path[hashapos++];
        }
        new1.cal_fitness();
        new2.cal_fitness();
        if (new1.fitness>temp[0].fitness)
            temp[index1]=new1;
        if (new2.fitness>temp[1].fitness)
            temp[index2]=new2;

}

void copulation()
{
  //int remainder = 4;
  //int r1,r2;
  int new_index = (index + 1) % 2;
  temp=population[index];
  //crossover(temp[p_size-1],temp[0],8,9);
  crossover(temp[p_size-1],temp[p_size-8],0,1);
  crossover(temp[p_size-2],temp[p_size-7],2,3);
  crossover(temp[p_size-3],temp[p_size-6],4,5);
  crossover(temp[p_size-4],temp[p_size-5],6,7);
  population[new_index] = temp;
  temp.clear();
}

//2-opt
void mutation()
```

```
{
	int new_index=(index+1)%2;
	for (int i=15;i<(int)population[new_index].size();i++)
	{
		int r;
		chromosome maxn,ttt,minn;
		int first_pos,second_pos;
		maxn.fitness=0;
		minn.fitness=MAXN;
		r=random(99)+1;

		//Local Search 1 (used as mutation) : 2-opt
		bool improved_flag = true;
		while (improved_flag)
		{
			improved_flag = false;
			for (first_pos=1;first_pos<=N-1;first_pos++)
				for (second_pos=first_pos+1;second_pos<=N-1;second_pos++)
				{
					ttt=population[new_index][i];
					for (int j=first_pos;j<=(first_pos+second_pos)/2;j++)
					{
						int t=ttt.path[j];
						ttt.path[j]=ttt.path[second_pos-(j-first_pos)];
						ttt.path[second_pos-(j-first_pos)]=t;
					}
					ttt.cal_fitness();
					if (ttt.fitness>maxn.fitness)
					{
						maxn=ttt;
						improved_flag = true;
```

```
        }
        if (ttt.fitness<minn.fitness)
            minn=ttt;
    }


//Local Search 2 : swap(u,v)
for (first_pos=1;first_pos<=N;first_pos++)
    for (second_pos=first_pos+1;second_pos<=N;second_pos++)
    {
        ttt=population[new_index][i];
        int t=ttt.path[first_pos];
        ttt.path[first_pos]=ttt.path[second_pos];
        ttt.path[second_pos]=t;
        ttt.cal_fitness();
        if (ttt.fitness>maxn.fitness)
        {
            maxn=ttt;
            improved_flag = true;
        }
        if (ttt.fitness<minn.fitness)
            minn=ttt;
    }

//Local Search 3 : insert(u,v) after x
for (first_pos=1;first_pos<=N;first_pos++)
    for (second_pos=first_pos+1;second_pos<=N;second_pos++)
    {
        int temp_arr[N+1];
        int arr_counter=0;
        for (int j=0;j<first_pos;j++)
```

```
                temp_arr[arr_counter++]=population[new_index][i].path[j];
            for (int j=second_pos+1;j<=N;j++)
                temp_arr[arr_counter++]=population[new_index][i].path[j];
            int insert_pos;
            insert_pos=rand()%arr_counter; // 0 to length (length+1==arr_counter)
            for (int j=0;j<=insert_pos;j++)
                ttt.path[j]=temp_arr[j];
            int ccc=0;
            while (ccc<=second_pos-first_pos)
            {
ttt.path[insert_pos+1+ccc]=population[new_index][i].path[first_pos+ccc];
                ccc++;
            }
            for (int j=insert_pos+1;j<arr_counter;j++)
            {
                ttt.path[insert_pos+1+ccc]=temp_arr[j];
                ccc++;
            }
            ttt.cal_fitness();
            if (ttt.fitness>maxn.fitness)
            {
                maxn=ttt;
                improved_flag = true;
            }
            if (ttt.fitness<minn.fitness)
                minn=ttt;
        }
    }


    if (maxn.fitness>population[new_index][i].fitness)
```

```
                    population[new_index][i]=maxn;
            //mutation : accept bad solution at rate 2%
            else if (r<=20)
                    population[new_index][i]=minn;
    }
}

double x[N+1],y[N+1];

double GetDistance1(double x1,double y1,double x2,double y2)
{
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

void input()
{
    FILE *fp=fopen("actual_pos.txt","r");//position of the depots and customers
    FILE *fpin=fopen("actual_cap.txt","r");//demand of customers at each node
    for (int i=0;i<=N;i++)
            fscanf(fp,"%lf%lf",&x[i],&y[i]);
    for (int i=0;i<=N;i++)
            for (int j=0;j<=N;j++)
                    if (i==j)
                            dis[i][j]=0;
                    else
                            dis[i][j]=GetDistance1(x[i],y[i],x[j],y[j]);
    for (int i=0;i<=N;i++)
            fscanf(fpin,"%lf",&c[i]);
    fclose(fp);
}
```

```
int findpos(int city,chromosome &best_chr)
{
	int pos;
	for (pos=0;pos<=N;pos++)
		if (best_chr.path[pos]==city)
			break;
	return pos;
}

int main()
{
	clock_t start_time=clock();
	input();
	int counter=1;
	int gen;

	double max_fitness=0;
	chromosome best_chr;
	init_population();
	FILE *fit=fopen("best_dis.txt","w");
	while (counter<=200)
	{
		cout<<counter<<"  generation\n";
		update_fitness();
		sort(population[index].begin(),population[index].end(),cmp);
		if (population[index][population[index].size()-1].fitness>max_fitness)
		{
			max_fitness=population[index][population[index].size()-1].fitness;
			best_chr=population[index][population[index].size()-1];
		}
		fprintf(fit,"%lf\n",worst_dis-max_fitness);
```

```
		copulation();
		mutation();
		index=(index+1)%2;
		counter++;
	}
	printf("The best dis is : %f\n",worst_dis-best_chr.fitness);
	int city=best_chr.path[N];


	int depotcounter=0;
	char filename[100],pathname[100];
	strcpy(filename,"matlab_data1.txt");
	strcpy(pathname,"matlab_path1.txt");
	while (1)
	{
		depotcounter++;
		filename[11]=depotcounter+'0';
		pathname[11]=depotcounter+'0';
		FILE *fout=fopen(filename,"w");
		FILE *fstr=fopen(pathname,"w");

		int ans[N];
		int ansc=0;
		while (1)//here
		{
			ans[ansc++]=city;
			if (city==best_chr.pre[city])
				break;
			city=best_chr.pre[city];
		}
		printf("---The depot %d---\n",city);
```

```
ansc--;
int pos=findpos(city,best_chr);
int record_pos=pos;//depot position
pos++;
int vehicle_counter=0;
for (int i=ansc-1;i>=0;i--)
{
	int v_load=0;
	printf("The route of vehicle %d : %d -> ",++vehicle_counter,best_chr.path[record_pos]);
	fprintf(fout,"%f %f\n",x[best_chr.path[record_pos]],y[best_chr.path[record_pos]]);
	fprintf(fstr,"%d\n",best_chr.path[record_pos]);
	while (best_chr.path[pos]!=ans[i])
	{
		v_load+=c[best_chr.path[pos]];
		fprintf(fout,"%f %f\n",x[best_chr.path[pos]],y[best_chr.path[pos]]);
		fprintf(fstr,"%d\n",best_chr.path[pos]);
		printf("%d -> ",best_chr.path[pos]);
		pos++;
	}
	v_load+=c[best_chr.path[pos]];
	fprintf(fout,"%f %f\n",x[best_chr.path[pos]],y[best_chr.path[pos]]);
	fprintf(fstr,"%d\n",best_chr.path[pos]);
	printf("%d -> %d.\n",best_chr.path[pos++],best_chr.path[record_pos]);
	fprintf(fout,"%f %f\n",x[best_chr.path[record_pos]],y[best_chr.path[record_pos]]);
	fprintf(fstr,"%d\n",best_chr.path[record_pos]);

	printf("The load of vehicle %d : %d.\n",vehicle_counter,v_load);
}
if (!city) break;
//find the position of city (pos == 0 is impossible)
pos=findpos(city,best_chr);
```

```
            city=best_chr.path[pos-1];
            fclose(fout);
            fclose(fstr);
        }
        fclose(fit);
        clock_t end_time=clock();
        printf("Time :%lds\n",(end_time-start_time)/CLOCKS_PER_SEC);
        //system("pause");
}
```

REFERENCES

[1] T. H. Wu, C. Low, and J. W. Bai, "Heuristic solutions to multi-depot location-routing problem," Computers & Operations Research, vol. 29, pp. 1393–1415, 2002

[2] A. Haghani and S. Jung, "A dynamic vehicle routing problem with time-dependent travel times," Computers and Operations Research, vol. 32, pp. 2959 - 2986, November 2005.

[3] N. Yoshiike and Y. Takefuji, "Solving vehicle routing problems by maximum neuron mode," Advanced Engineering Informatics, vol. 16, pp. 99-105, April 2002.

[4] I. D. Giosa, I. L. Tansini, and I. O. Viera, "New assignment algorithms for the multi-depot vehicle routing problem," Journal of the Operational Research Society, vol. 53, pp. 977-984, 2002.

[5] S. Salhi and M. Sari, "A multi-level composite heuristic for the multidepot vehicle fleet mix problem," European Journal of Operational Research, vol. 103, pp. 95-112, 16 November 1997.

[6] C.-G. Lee, M. A. Epelman, C. C. White, and Y. A. Bozer, "A shortest path approach to the multiple-vehicle routing problem with split pick-ups," Transportation Research Part B: Methodological, vol. 40, pp. 265–284, May 2006.

[7] B. Creviera, J.-F. Cordeaua, and G. Laporte, "The multi-depot vehicle routing problem with inter-depot routes," European Journal of Operational Research, vol. 176, pp. 756-773, 16 January 2007.

[8] G. Nagy and S. Salhi, "Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries," European Journal of Operational Research, vol. 162, pp. 126-141, 1 April 2005.