A Novel Bio-inspired Algorithm for TSP

A project report submitted in partial fulfillment of the requirements for the Award of the diploma in

Computer Science and Engineering

Submitted by
Aditya Anand
(DIP/14/CS/002)

Under the guidance of
Mr. Aswini Kumar Patra
Assistant Professor

Department of Computer Science and Engineering
North Eastern Regional Institute of Science and Technology
(Deemed university, under MHRD, Govt. of India)
Nirjuli, Arunachal Pradesh, 791109

May 2016

CERTIFICATE OF APPROVAL

Certified that the project entitled “**A novel bio-inspired algorithm for TSP”** is hereby approved as a credible work of an engineering subject, carried out by **Aditya Anand (DIP/14/CS/002)**, presented in a manner satisfactory to warrant its acceptance as a prerequisite to the award of **Diploma in Computer Science and Engineering** submitted to **North EasternRegional Institute ofScienceandTechnology**. It is understood by this approval that the undersigned do not endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it has been submitted.

(Mr. Aswini Kumar Patra)
Project Guide
Assistant Professor,
Computer Science and
Engineering Department

(Mr. Aswini Kumar Patra)
Project Coordinator
Assistant Professor,
Computer Science and
Engineering Department

(Dr. (Mrs) NingrinlaMarchang)
Head of Department
Computer Science and Engineering

DECLARATION BY THE CANDIDATE

The work titled "**A novel bio-inspired algorithm for TSP**" submitted to the Department of Computer science and Engineering, NERIST, contains no materials which has been accepted for the award of any other Degree in any other tertiary institution and to the best of my knowledge and belief, contains no materials previously published or written by another person, **except** where due reference has been made in the text.

ADITYA ANAND
DIP/14/CS/002

ACKNOWLEDGEMENT

This project has helped us to earn a lot of knowledge and skills in solving Travelling Salesman Problem and this would not have been possible without proper guidance and support. So I take this opportunity to thanks those influential persons.

First of all, I would like to have the opportunity to thank the whole family of **"Computer Science and Engineering Department, NERIST"** for the help and support in the completion of this project**.**

In this context, my sincere thanks goes to Sir Aswini Kumar Patra (Coordinator and guide) from whom I got this opportunity to undergo project in this organization. I would like to acknowledge the contribution of all concerned persons, technicians and all other who helped us in one way or the other while progressing with our work.

Aditya Anand
Dip/14/CS/002

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Optimization problem, either single-objective or multi-objective, are generally difficult to solve and many of them are said to be NP-hard and cannot be solved effectively by any known algorithms in acceptable amount of time. The most famous optimization problem is probably “The Travelling Salesman Problem”.

The purpose of this project is to perform research on “Travelling Salesmen Problem (TSP)”.

1.1 Motivation

Much of the work on the TSP is motivated by its use as a platform for the study of general methods that can be applied to a wide range of discrete optimization problems. Indeed, the numerous direct applications of the TSP bring life to the research area and help to direct future work.The TSP naturally arises as a sub-problem in many transportation and logistics applications, for example the problem of arranging school bus routes to pick up the children in a school district. More recent applications involve the scheduling of service calls at cable firms, the delivery of meals to homebound persons, the scheduling of stacker cranes in warehouses, the routing of trucks for parcel post pickup, and a host of others.

Although transportation applications are the most natural setting for the TSP, the simplicity of the model has led to many interesting applications in other areas. A classic example is the scheduling of a machine to drill holes in a circuit board or other object. In this case the holes to be drilled are the cities, and the cost of travel is the time it takes to move the drill head from one hole to the next. The technology for drilling varies from one industry to another, but whenever the travel time of the drilling device is a significant portion of the overall manufacturing process then the TSP can play a role in reducing costs.

1.2 Objective

The TSP problem is NP-hard Problem and the optimal solution takes exponential rate of growth. It has always been challenging to improve the result with polynomial time complexity. The main objective is to further reduce the time consumption for Travelling Salesman Problem using a bio-inspired approach.

1.3 Contribution of the project

This project contributes a novel bio-inspired algorithm for Travelling Salesman Problem.

CHAPTER 2: PRELIMINARIES AND BACKGROUND

A Travelling Salesmen may encounter problems is such a way that he/she needs to start from a fixed city, travel each city once and only once, and return back to the city from where he/she started. So, *the solution* comprises of finding the *shortest possible path* (from the starting city, visiting all possible cities and returning back to the city from where the salesman started) and *the distance(rather cost)* of the shortest path.Now, TSP is further classified into **symmetric TSP and asymmetric TSP.** If the distance from 'City 1' to 'City 2' is *equal* to the distance from 'City 2' back to 'City 1', then it is said to be *Symmetric TSP*. Likewise, if the distance from 'City 1' to 'City 2' is *not equal* to the distance from 'City 2' back to 'City 1', then it is said to be *Asymmetric TSP*.

2.1 Cuckoo Search

Cuckoo search (CS) is an optimization algorithm developed by Xin-she Yang and Suash Deb in 2009. It was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (of other species).Some host birds can engage direct conflict with the intruding cuckoos. For example, if a host bird discovers the eggs are not their own, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Cuckoo search idealized such breeding behavior, and thus can be applied for various optimization problems. It seems that it can outperform other metaheuristic algorithms in applications. Cuckoo search (CS) uses the following representations:

Each egg in a nest represents a solution, and a cuckoo egg represents a new solution. The aim is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests. In the simplest form, each nest has one egg. The algorithm can be extended to more complicated cases in which each nest has multiple eggs representing a set of solutions.

CS is based on three idealized rules:

1. Each cuckoo lays one egg at a time, and dumps its egg in a randomly chosen nest;
2. The best nests with high quality of eggs will carry over to the next generation;
3. The number of available hosts' nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability (0 to 1). Discovering operate on some set of worst nests, and discovered solutions dumped from farther calculations.

2.2 Ordered Crossover

Ordered crossover build off springs by choosing a subsequence of a tour from one parent and preserving the relative order of cities from the other parent. Two cut points are selected and the elements between them are copied. The rest elements are copied from the beginning of the second parent respecting their relative order omitting those which already exist in the first parent.

For example, consider two parents p1 and p2 (with two points marked by " | ")

p1 = (1 2 │3 4 5 │ 6 7 8 9) and p2 = (8 5 │ 7 1 2 │ 4 9 3 6)

Produce off springs in the following way. First the segments between the cut points are copied in to off springs.

O1 = (- - │3 4 5 │ - - - -) and O2 = (- - │7 1 2 │ - - - -)

The sequence of cities in the second parent is 4 5 7 1 2 6 8 9 3. Removing 3, 4 and 5 we get

7 1 2 6 8 9

Placing this sequence in the first offspring from left to right according to the order we have

O1 = (7 1 3 4 5 2 6 8 9)

Similarly O2 = (3 4 7 1 2 5 6 8 9)

2.3 Python Programming Language

Python is an open-source object-oriented programming language that offers two to tenfold programmer productivity increase over languages like C, C++, Java, C#, Visual Basic (VB), and Perl.Python is an agile programming language. Agile Programming languages has the following features:

- excellent for beginners, yet superb for experts
- highly scalable, suitable for large projects as well as small ones
- rapid development
- portable, cross-platform
- embeddable
- easily extensible
- object-oriented
- stable and mature

2.4 Distance Matrix

The distance matrix comprises of values of distance from one node to any other node (including itself). The distance matrix is *a list of lists (2D Array*) represented by D[i][j].

D[i][j]==D[j][i].

i j

0	10	8	9	7
10	0	10	5	6
8	10	0	8	9
9	5	8	0	6
7	6	9	6	0

Fig.: Distance Matrix of 5 cities TSP.

CHAPTER 3: PROPOSED WORK(s)

Step 1: The distance matrix is taken as input from .txt file stored in HDD.
Step 2: 100 initial populations are generated. Each population is a list of n numbers (cities) such that each number appear only once.
Step 3: The 100 populations, so generated, are further divided into 5 local lists, each containing 20 lists.
Step 4: The 5 local lists are then send for sorting one after other. The sorted list contains populations sorted according to their minimum value.
Step 5: After the 5 list are sorted, each of the list are affected in such a way that 60% of the sorted list are kept and 30% comes from the ordered crossover operation performed over already present 60% list and the rest 10% is the Global Best.
Step 6: After the lists are affected, they are distributed depending upon the distribution probability.
Step 7: After the distribution is over, some population (depending upon user input) is selected for swapping 3 pair of index, (i.e., 25 and 40; 12 and 20; 5 and 35 in case of 48 cities TSP).
Step 8: Steps 5, 6 and 7 are repeated several number of times (decided by user).
Step 9: After several number of iterations, the best population with its value is printed.

3.1 Algorithm with code segments

Step 1: The distance matrix is taken as input from .txt file stored in HDD.

```
#reading 48 cities distance matrix
        f = open("L:/pendrive/48inp.txt","r")
        inputlist=[[int(i) for i in line.split()] for line in f]
```

Step 2: 100 initial populations are generated.

```
defgenerate_initial_population(n):
        list1=[]
        new2=[]
        fori in range(0,100,1):
                a=[]
                j=0
                a=range(0,n)
                random.shuffle(a)
                list1.append(a)
        return list1
```

Step 3: The 100 populations, so generated, are further divided into 5 local lists, each containing 20 lists.

```
#dividing 100 population into 5 local list
        i=0
        ilist=[]
        while(i<100):
                ilist.insert((i/20),listpop[i:i+20])
                i=i+20
```

Step 4: The 5 lists are then send for sorting. The sorted list contains populations sorted according to their minimum value.

```
defget_best(temp,n,inputlist,div):
        ilop=0
        div=int(div)
        whileilop<div:
                a=[]
                t=0
                s=0
                a=temp[ilop]
                j=0
                nm=n-1
                while j<nm:
                        b=a[j]
                        c=a[j+1]
                        s=s+inputlist[b][c]
                        j=j+1
                d=a[j-1]
                e=a[0]
                t=s+inputlist[d][e]
                sumarr.append(t)
                ilop=ilop+1
        sumindx=[]
        sumindx=sorted(range(len(sumarr)), key=lambda k: sumarr[k])
        list=[]
        indx=0
        whileindx<div:
                ind=sumindx[indx]
                list.append(temp[ind])
                indx=indx+1
        listval=[]
        indx=0
        whileindx<div:
                ind=sumindx[indx]
                listval.append(sumarr[ind])
                indx=indx+1
        return (list,listval)
```

Step 5: After the 5 list are sorted, each of the list are affected in such a way that 60% of the sorted list are kept and 30% comes from the ordered crossover operation performed over already present 60% list and the rest 10% is the Global Best.

```
def affect(slist,n):
        ap=0.6 * 20
        li=slist[0:12] #taking 12 best
        i=0
        whilei<ap:
                li.append(ord_cross(li[i],li[i+1],n))
                i=i+2
        li.append(Global1)
        li.append(Global2)
        return li
```

Step 6: After the lists are affected, they are distributed depending upon the distribution probability.

```
def distribute(mainlist,n,dp,div,k):
        i=0
        div=int(div)
        k=int(k)
        dprob=dp*div
        dprob=int(dprob)
        whilei<k:
                li1=mainlist[i]
                Hlist.append(li1[0:dprob])
                Llist.append(li1[dprob:div])
                i=i+1
        j=0
        dlist=[]
        while j<k-1:
                temp1=[]
                temp2=[]
                temp3=[]
                temp1=Hlist[j]
                temp2=Llist[j+1]
                temp3=temp1+temp2
                dlist.append(temp3)
                j=j+1
        temp1=Hlist[j]
        temp2=Llist[0]
        temp3=temp1+temp2
        dlist.append(temp3)
        returndlist
```

Step 7: After the distribution is over, some population (depending upon user input) is selected for swapping 3 pair of index, (i.e., 25 and 40; 12 and 20; 5 and 35 in case of 48 cities TSP).

```
# hard coding used for swapping two elements of the population, only for 48 cities
tsp
inner=0
while inner<swapPPL:
        ALIST[inner][25],ALIST[inner][40]=swapp(ALIST[inner][25],ALIST[inner][40])
        ALIST[inner][12],ALIST[inner][20]=swapp(ALIST[inner][12],ALIST[inner][20])
        ALIST[inner][5],ALIST[inner][35]=swapp(ALIST[inner][5],ALIST[inner][35])
        inner=inner+1
```

```
print "No. of iterations?"
iter=input()
while loop<iter:
        alist1=[]
        alist=[]
        ALIST=[]
        i3=0
        while(i3<5):
                alist1=affect(slcombo[i3],n)
                ALIST=ALIST+alist1
                alist.append(alist1)
                i3=i3+1
        loop=loop+1
        inner=0
        while inner<swapPPL:
                ALIST[inner][25],ALIST[inner][40]=swapp(ALIST[inner][25],ALIST[inner][40])
                ALIST[inner][12],ALIST[inner][20]=swapp(ALIST[inner][12],ALIST[inner][20])
                ALIST[inner][5],ALIST[inner][35]=swapp(ALIST[inner][5],ALIST[inner][35])
                inner=inner+1
        i2=0
        while(i2<5):
                sortedlist,sortedvalue=get_best(alist[i2],n,inputlist)
                slcombo.append(sortedlist)
                svcombo.append(sortedvalue)
                i2=i2+1
```

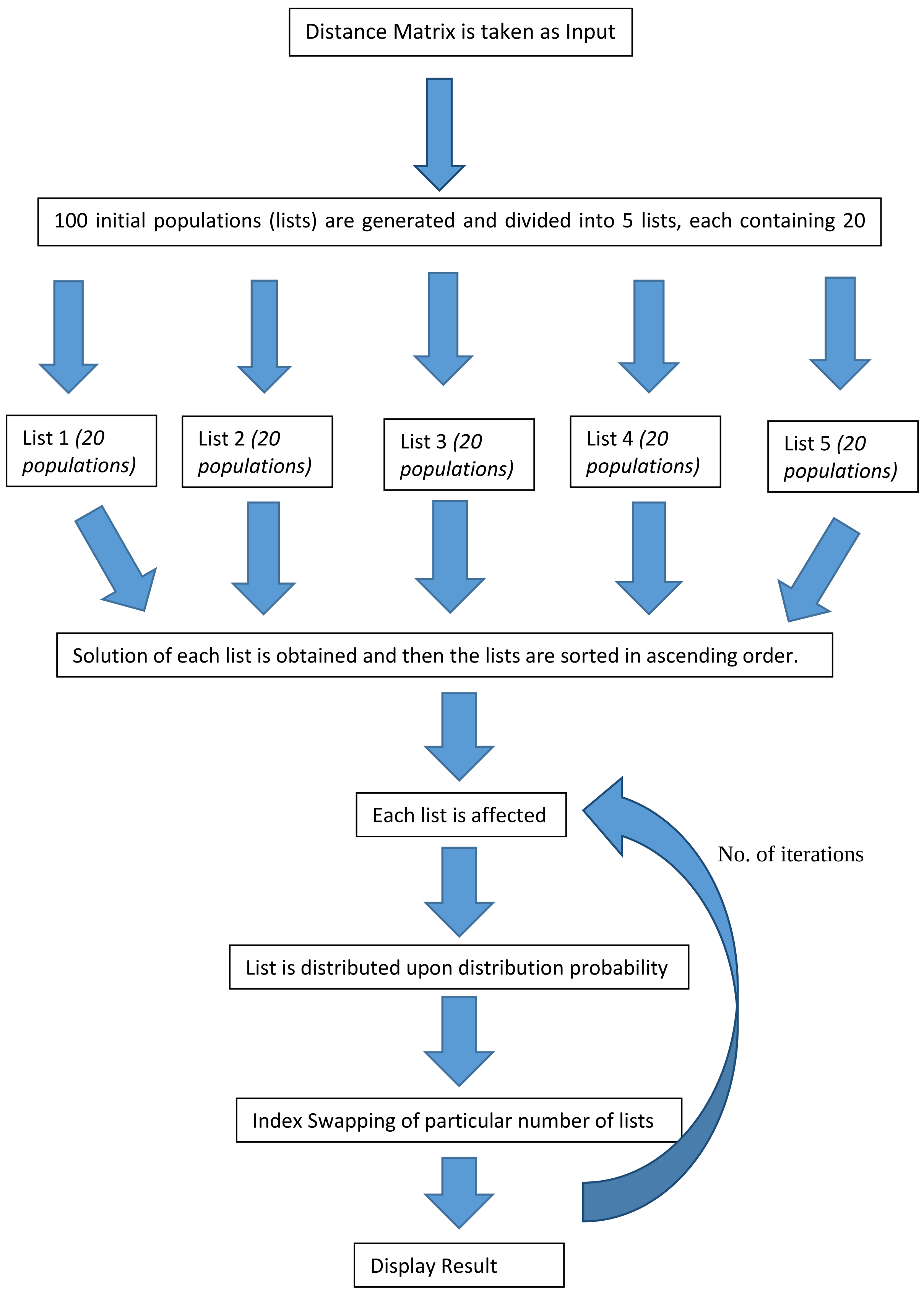
Step 8: Steps 5, 6 and 7 are repeated several number of times (decided by user).

Step 9: After several number of iterations, the best population with its value is printed.

```
print "Minimum Tour Path By Proposed Alogorithm :"
printslcombo[0][0]  #Best Path

print "Minimum Tour length By Proposed Alogorithm :"
printsvcombo[0][0]  #Shortest distance
```

3.2 Flowchart


Distance Matrix is taken as Input
100 initial populations (lists) are generated and divided into 5 lists, each containing 20
List 1 *(20 populations)*
List 2 *(20 populations)*
List 3 *(20 populations)*
List 4 *(20 populations)*
List 5 *(20 populations)*
Solution of each list is obtained and then the lists are sorted in ascending order.
Each list is affected
No. of iterations
List is distributed upon distribution probability
Index Swapping of particular number of lists
Display Result

CHAPTER 4: EXPERIMENTS AND RESULT

4.1 Hardware and Software Used:

All experiments, related to this project, is performed on IntelCore i7 processor with 2.20 GHz and 8GB RAM.

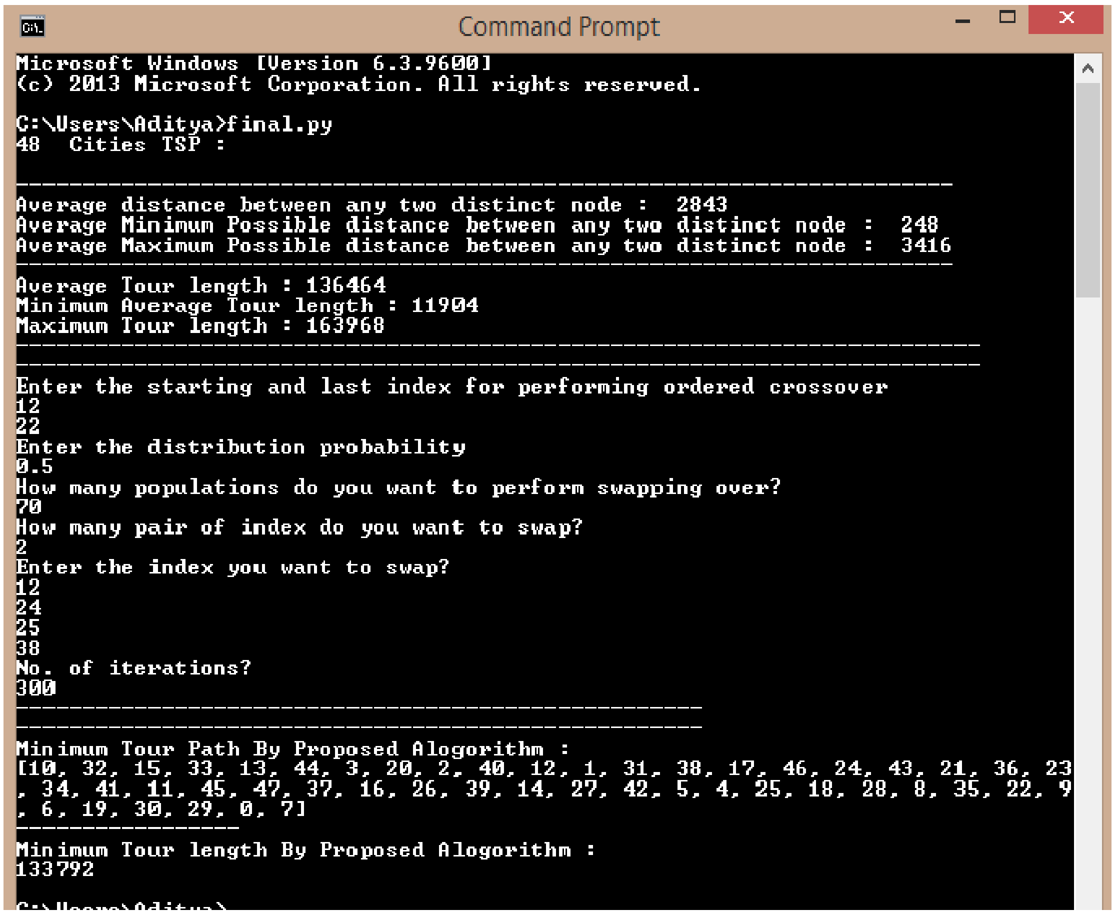
Python 2.7.11 is used as the python compiler. Codes are written in a text editor, in my case I used Notepad++. Execution of code is done in the Kernel Mode Interface, in my case it is *Command prompt* (Microsoft Windows [version 6.3.9600]).

4.2 Implementation Details

This project is implemented using Python Programming Language. The distance Matrix that we use is a *text file (.txt)* where each line is considered to be a row, and the columns inside those rows (or line) are separated using space (alt+255) or tab (\t). The code was implemented and executed in Windows platform, anyways it does not matter.

4.3 Experiment Details with Screenshots of Output Result

Numerous experiments were performed and are plotted on the graph below. These are some screenshots of the experiments performed:

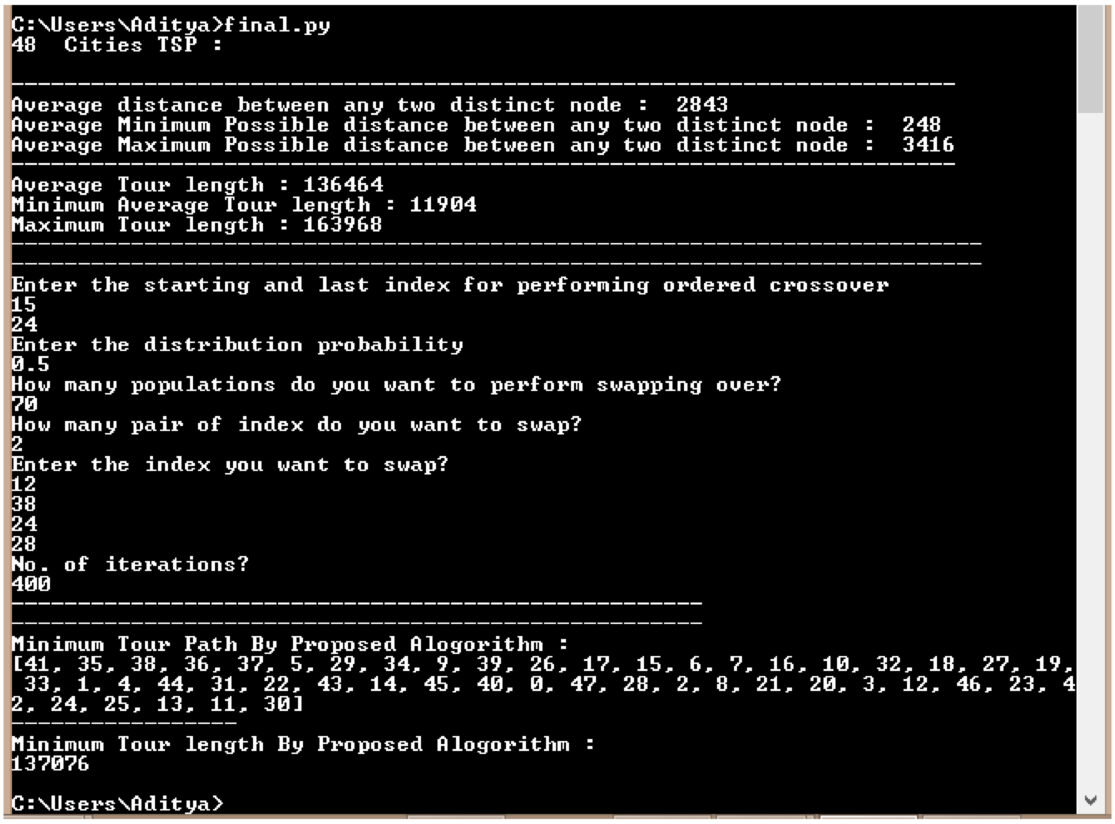


```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Aditya>final.py
48   Cities TSP :

-----------------------------------------------------------------------
Average distance between any two distinct node :  2843
Average Minimum Possible distance between any two distinct node :  248
Average Maximum Possible distance between any two distinct node :  3416
-----------------------------------------------------------------------
Average Tour length : 136464
Minimum Average Tour length : 11904
Maximum Tour length : 163968
-------------------------------------------------------------------------
-------------------------------------------------------------------------
Enter the starting and last index for performing ordered crossover
12
22
Enter the distribution probability
0.5
How many populations do you want to perform swapping over?
70
How many pair of index do you want to swap?
2
Enter the index you want to swap?
12
24
25
38
No. of iterations?
300
----------------------------------------------------
----------------------------------------------------
Minimum Tour Path By Proposed Alogorithm :
[10, 32, 15, 33, 13, 44, 3, 20, 2, 40, 12, 1, 31, 38, 17, 46, 24, 43, 21, 36, 23
, 34, 41, 11, 45, 47, 37, 16, 26, 39, 14, 27, 42, 5, 4, 25, 18, 28, 8, 35, 22, 9
, 6, 19, 30, 29, 0, 7]
-----------------
Minimum Tour length By Proposed Alogorithm :
133792

C:\Users\Aditya>
```

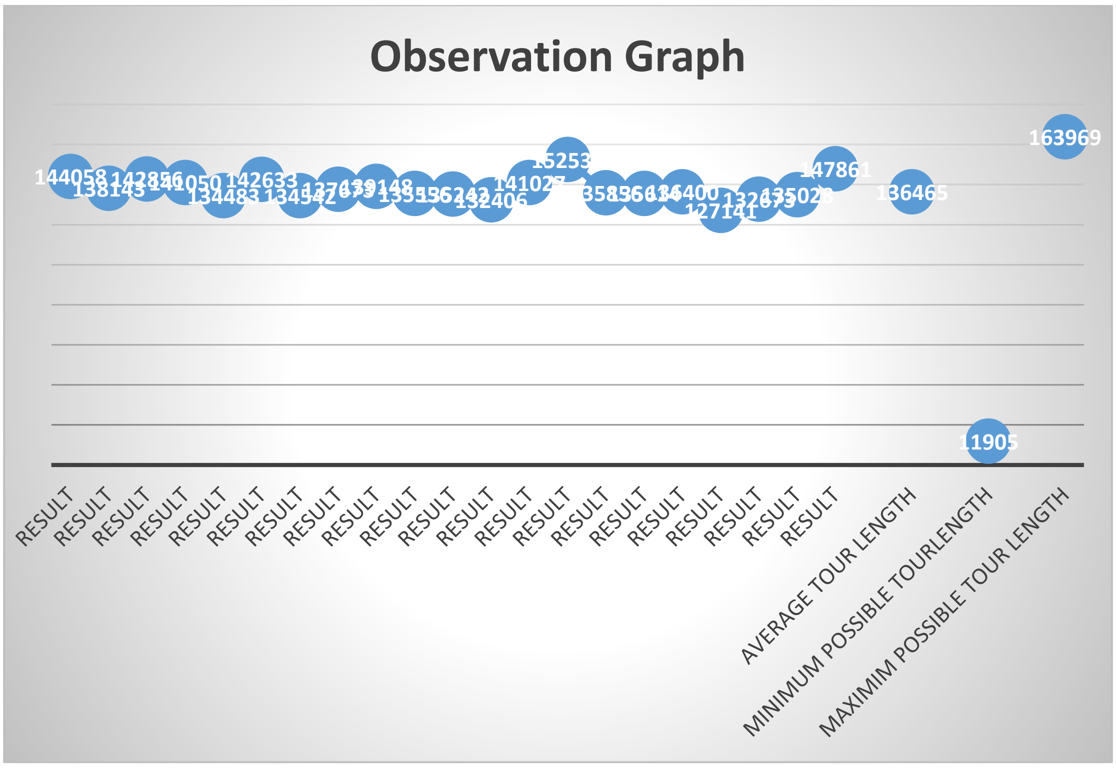


```
C:\Users\Aditya>final.py
48   Cities TSP :

-----------------------------------------------------------------------
Average distance between any two distinct node :  2843
Average Minimum Possible distance between any two distinct node :  248
Average Maximum Possible distance between any two distinct node :  3416
-----------------------------------------------------------------------
Average Tour length : 136464
Minimum Average Tour length : 11904
Maximum Tour length : 163968
-------------------------------------------------------------------------
-------------------------------------------------------------------------
Enter the starting and last index for performing ordered crossover
15
24
Enter the distribution probability
0.5
How many populations do you want to perform swapping over?
70
How many pair of index do you want to swap?
2
Enter the index you want to swap?
12
38
24
28
No. of iterations?
400
----------------------------------------------------
----------------------------------------------------
Minimum Tour Path By Proposed Alogorithm :
[41, 35, 38, 36, 37, 5, 29, 34, 9, 39, 26, 17, 15, 6, 7, 16, 10, 32, 18, 27, 19,
 33, 1, 4, 44, 31, 22, 43, 14, 45, 40, 0, 47, 28, 2, 8, 21, 20, 3, 12, 46, 23, 4
2, 24, 25, 13, 11, 30]
-----------------
Minimum Tour length By Proposed Alogorithm :
137076

C:\Users\Aditya>
```

4.4 Graphs



CHAPTER 5: CONCLUSIONand FUTURE WORK

We implemented the proposed algorithm and it is shown that the result obtained is roughly equal to the Average Tour Length, sometimes, even better values are obtained. Further, it can be used to minimize the result. The conclusion derived from this project is a novel bio-inspired algorithm for TSP (n-city TSP). Many approaches were made to minimize the tour length. The ordered crossover operations may give us a value somewhat better than the lists send as operands, populations were distributed among themselves, and some of the list index were also changed to observe the effect.

This project can be used to study the different approaches made in solving n-city TSP and study the drawbacks/limitations while approaching towards solving an n-city TSP.

REFERENCES

1. Discrete cuckoo search algorithm for the travelling salesman problem- Aziz Ouaarab, Belaı¨dAhiod, Xin-She Yang -© Springer-Verlag London 2013.
2. Fundamentals ofPython: - First Programs - Kenneth A. Lambert, Martin Osborne (Contributing Author).
3. Wikipedia – TSP problems, Cuckoo Search, Ordered Crossover.