TRAVELLING SALESMAN PROBLEM

DATA STRUCTURES AND ALGORITHMS

FINAL REPORT

Group members:

1. **Shivam Kumar Verma -16BCB0015**
2. **Shrutisha Joshi-16BCB0066**
3. **Saikat Bhattacharya-16BCB0092**

ABSTRACT

The traveling-salesman problem is one of the classical NP-Complete problems. No current algorithms are available which can solve these problems in polynomial time, that is, the number of steps grows as a polynomial according to the size of the input. The traveling-salesman problem involves a salesman who must make a tour of a number of cities using the shortest path available. For each number of cities n, the number of paths which must be explored is n!, causing this problem to grow exponentially rather than as a polynomial. Three separate algorithms are examined. These are an iterative algorithm, a recursive algorithm, and a branch and bound algorithm.

INTRODUCTION

Traveling salesman problem (TSP) is a classiccombinatorial optimization problem, which is to find a shortesttour route for a traveling salesman in the predefined quantity ofcities. The constraints of TSP require that the traveling

salesman can only visit each city just once and return back tothe starting city. TSP has a wide range of applications in manyareas..

Traveling salesman problem(TSP) is a classiccombinatorial optimization problem. The goal is to find theshortest tour that visits each city in a given list exactly once andthen returns to the starting city[18]. Generally speaking, the

TSP can be described as follows:

$$\min Z = \sum_{i \neq j} d_{ij} \times x_{ij} \tag{1}$$

$$\text{st.} \begin{cases} \sum_{i \neq j} x_{ij} = 1, i \in v \\ \sum_{i \neq j} x_{ij} = 1, j \in v \\ \sum_{i,j \in s} x_{ij} = |s| \\ x_{ij} \in \{0,1\} \end{cases} \tag{2}$$

where $v = 1,2,3\ldots,n$ is defined as the vertex set , $E = \{e_{ij} = (i,j) | i,j \in v, i \neq j\}$ is defined as the frontier set, $d_{ij}(i,j \in v, i \neq j)$ is defined as the distance of vertex i to j, $x_{ij} \in (0,1)$, if the frontier e_{ij} is not on the optimal path, then $x_{ij} = 0$, otherwise, $x_{ij} = 1$.The objective of TSP is to make a traveling salesman to go through every city which can be regarded as one point, and each point just can be through one time, then finally find the shortest cycle path.

ABOUT METHODOLOGY

GREEDY

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

For example, a greedy strategy for the traveling salesman problem (which is of a high computational complexity) is the following heuristic: "At each stage visit an unvisited city nearest to the current city". This heuristic need not find a best solution, but terminates in a reasonable number of steps; finding an optimal solution typically requires unreasonably many steps. In mathematical optimization, greedy algorithms solve combinatorial problems having the properties of matroids**.**

BRANCH AND BOUND

Branch and bound isan algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores *branches* of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated *bounds* on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

The algorithm depends on the efficient estimation of the lower and upper bounds of a region/branch of the search space and approaches exhaustive enumeration as the size (*n*-dimensional volume) of the region tends to zero.

GENETIC ALGORITHM(GA)

In computer science and operations research, a **genetic algorithm** (**GA**) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

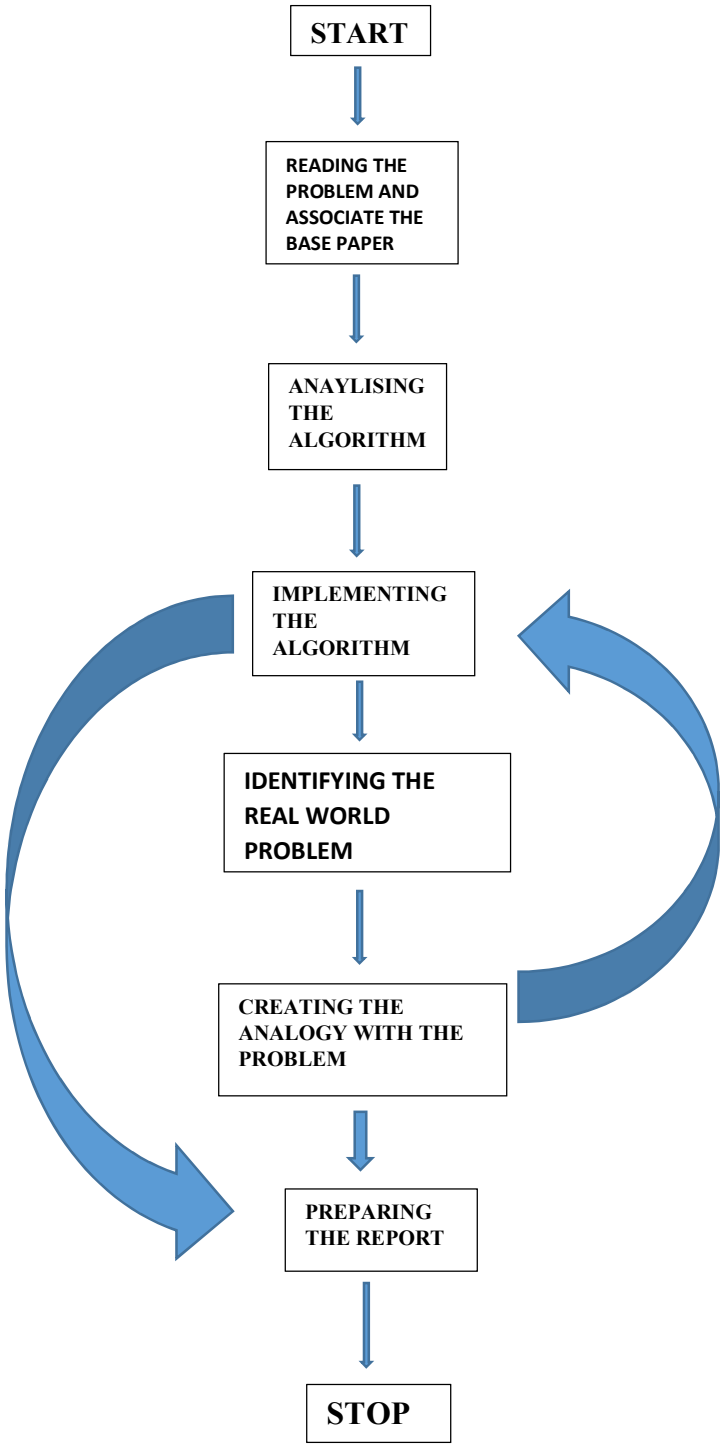
In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.[2]

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

PROCEDURE:


START
READING THE PROBLEM AND ASSOCIATE THE BASE PAPER
ANAYLISING THE ALGORITHM
IMPLEMENTING THE ALGORITHM
IDENTIFYING THE REAL WORLD PROBLEM
CREATING THE ANALOGY WITH THE PROBLEM
PREPARING THE REPORT
STOP

The Code: (for GA with greedy and branch and bound)

```
from random import randrange as rr

from math import sqrt

import time

def distance(l):
   s=0
   for i in range(1,len(l)):
      x1,y1=dic[l[i-1]][0],dic[l[i-1]][1]
      x2,y2=dic[l[i]][0],dic[l[i]][1]
      s+=sqrt((x2-x1)**2+(y2-y1)**2)
   return s
def dist(i,j):
   return float(format(sqrt((i[1]-j[1])**2+(i[0]-j[0])**2),".1f"))

dic={}
siz=int(input())
for _ in range(siz):
   inp=list(map(float,input().strip().split()))
   dic[int(inp[0])]=(inp[1],inp[2])

dic[siz+1]=dic[1]

l=[i+1 for i in range(siz)]
l=l+[siz+1]
#l=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
cities=list(l)
len_c=len(cities)
len_c-=1

population=len_c*6
# permutations

paths=[]
i=0
while i<population-1:
   r1=rr(1,len_c)
   r2=rr(1,len_c)
```

```
    while r1==r2:
        r1=rr(1,len_c)
        r2=rr(1,len_c)
    cities[r1],cities[r2]=cities[r2],cities[r1]
    if cities not in paths:
        paths.append(list(cities))
        i+=1
ll=list(l)

# grredy cell----------------------++++++++++++++++++++++++++++++++++++++++

for i in range(0,len_c-1):
    c=distance([l[i],l[i+1]])
    jj=i+1
    for j in range(i+2,len_c-1):
        if distance([l[i],l[j]])<c:
            jj=j
            c=distance([l[i],l[j]])
    l[i+1],l[jj]=l[jj],l[i+1]


paths.append(list(l))
l=list(ll)
# grredy cell----------------------++++++++++++++++++++++++++++++++++++++++



#b & b ----------------------------------------++++++++++++++++++++++++++++

n=len(l)
n-=1
costm=[]

for i in range(1,n+1):
    te=[]
    for j in range(1,n+1):
        if i==j:
            te.append("inf")
        else:
            te.append(dist(dic[i],dic[j]))
    costm.append(te)


def reduc(matr,n):
```

```
    cost=0
    for i in range(n):
        min_num=10000000
        for j in range(n):
            if matr[i][j]!='inf':
                if min_num>matr[i][j]:
                    min_num=matr[i][j]


        if min_num!=0 and min_num!=10000000:
            for j in range(n):
                if matr[i][j]!='inf':
                    matr[i][j]-=min_num
            cost+=min_num


    for i in range(n):
        min_num=10000000
        for j in range(n):
            if matr[j][i]!='inf':
                if min_num>matr[j][i]:
                    min_num=matr[j][i]
        if min_num!=0 and min_num!=10000000:
            for j in range(n):
                if matr[j][i]!='inf':
                    matr[j][i]-=min_num
            cost+=min_num

    return cost

cost1=reduc(costm,n)



def makeinf(m,n,x,y):
    for i in range(n):
        m[x][i]='inf'
        m[i][y]='inf'
    m[y][0]='inf'


for i in range(0,n-1):
    mij=costm[l[i]-1][l[i+1]-1]
    newmat=[list(costm[_]) for _ in range(n)]
```

```
    makeinf(newmat,n,l[i]-1,l[i+1]-1)
    if mij=='inf':
        mij=100000
    costmin=cost1+reduc(newmat,n)+ mij
    z=i+1

    confmat=[list(newmat[_]) for _ in range(n)]
    for j in range(i+2,n):
        mij=costm[l[i]-1][l[j]-1]
        if mij=='inf':
            mij=100000
        newmat=[list(costm[_]) for _ in range(n)]
        makeinf(newmat,n,l[i]-1,l[j]-1)
        curcost=(cost1+reduc(newmat,n)+ mij)

        if curcost<costmin:
            costmin=curcost
            z=j
            confmat=[list(newmat[_]) for _ in range(n)]
    cost1=costmin
    costm=[list(confmat[_]) for _ in range(n)]
    l[i+1],l[z]=l[z],l[i+1]

print(l)
paths.append(l)
l=list(ll)

#b&b----------------------------------------------------------++++++++++++++++++++


#crossovers
def crossover():
    j=0
    crossovers=[]
    parents=[]
    while j<population//2:
        r1=rr(0,population)
        r2=rr(0,population)
        while r1==r2 and r1 in parents and r2 in parents:
            r1=rr(0,population)
            r2=rr(0,population)
        parents.append(r1)
        parents.append(r2)
        p1=list(paths[r1])
```

```
    p2=list(paths[r2])

    r1=rr(1,len_c)
    r2=rr(1,len_c)
    while r1==r2:
        r1=rr(1,len_c)
        r2=rr(1,len_c)
    p1[r1:r2+1],p2[r1:r2+1]=p2[r1:r2+1],p1[r1:r2+1]

    xtra=list(set(l)-set(p1))
    len_x=len(xtra)
    for i in range(len(p1)):
        if p1.count(p1[i])>1:
            x=0#rr(0,len_x)
            p1[i]=xtra[x]
            del xtra[x]
            len_x-=1

    xtra=list(set(l)-set(p2))
    len_x=len(xtra)
    for i in range(len(p2)):
        if p2.count(p2[i])>1:
            x=0#rr(0,len_x)
            p2[i]=xtra[x]
            del xtra[x]
            len_x-=1

    if p1 not in crossovers and p2 not in crossovers and p1 not in paths and p2 not in paths:
        crossovers.append(list(p1))
        crossovers.append(list(p2))
        j+=1


#mutation
mutations=[]
i=0
while i<(population):
    mutagens=list(crossovers[i])
    r1=rr(1,len_c)
    r2=rr(1,len_c)
    while r1==r2:
        r1=rr(1,len_c)
        r2=rr(1,len_c)
    mutagens[r1],mutagens[r2]=mutagens[r2],mutagens[r1]
```

```
        if mutagens not in paths and mutagens not in crossovers and mutagens not in mutations:
            mutations.append(list(mutagens))
            i+=1

    ultimatelist=paths+crossovers+mutations
    distances={}
    for i in range(len(ultimatelist)):
        distances[distance(ultimatelist[i])]=ultimatelist[i]


    dissort=sorted(distances.keys())
    print(dissort[0])

    for i in range(min(population,len(dissort))):
        paths[i]=distances[dissort[i]]



generations=int(input())
print(time.ctime())
while generations>0:
    crossover()
    generations-=1
print(paths[0])
print(time.ctime())


"""

c=distance(paths[0])

    for i in range(len_c-1,0,-1):
        jj=i
        for j in range(1,len_c):
            paths[0][i],paths[0][j]=paths[0][j],paths[0][i]
            if distance(l)<c:
                jj=j
                c=distance(paths[0])
            paths[0][i],paths[0][j]=paths[0][j],paths[0][i]
        paths[0][i],paths[0][jj]=paths[0][jj],paths[0][i]
"""
```

Testing on different TSP problems:

Problem – 1:

```
NAME: burma14
TYPE: TSP
COMMENT: 14-Staedte in Burma (Zaw Win)
DIMENSION: 14
EDGE_WEIGHT_TYPE: GEO
EDGE_WEIGHT_FORMAT: FUNCTION
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
   1  16.47       96.10
   2  16.47       94.44
   3  20.09       92.54
   4  22.39       93.37
   5  25.23       97.24
   6  22.00       96.05
   7  20.47       97.02
   8  17.20       96.29
   9  16.30       97.38
  10  14.05       98.12
  11  16.53       97.38
  12  21.52       95.59
  13  19.41       97.13
  14  20.09       94.55
EOF
```

Input:

```
14
   1  16.47       96.10
   2  16.47       94.44
   3  20.09       92.54
   4  22.39       93.37
   5  25.23       97.24
   6  22.00       96.05
   7  20.47       97.02
   8  17.20       96.29
   9  16.30       97.38
  10  14.05       98.12
  11  16.53       97.38
  12  21.52       95.59
  13  19.41       97.13
  14  20.09       94.55
500
```

Output:

```
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
30.878503892588
[1, 10, 9, 11, 8, 13, 7, 12, 6, 5, 4, 3, 14, 2, 15]
Tue May  2 22:54:23 2017
```

Solution: [1, 10, 9, 11, 8, 13, 7, 12, 6, 5, 4, 3, 14, 2, 15]

Where 15 is city 1 itself (complete tour).

Generations: 500

Population each generation: 140

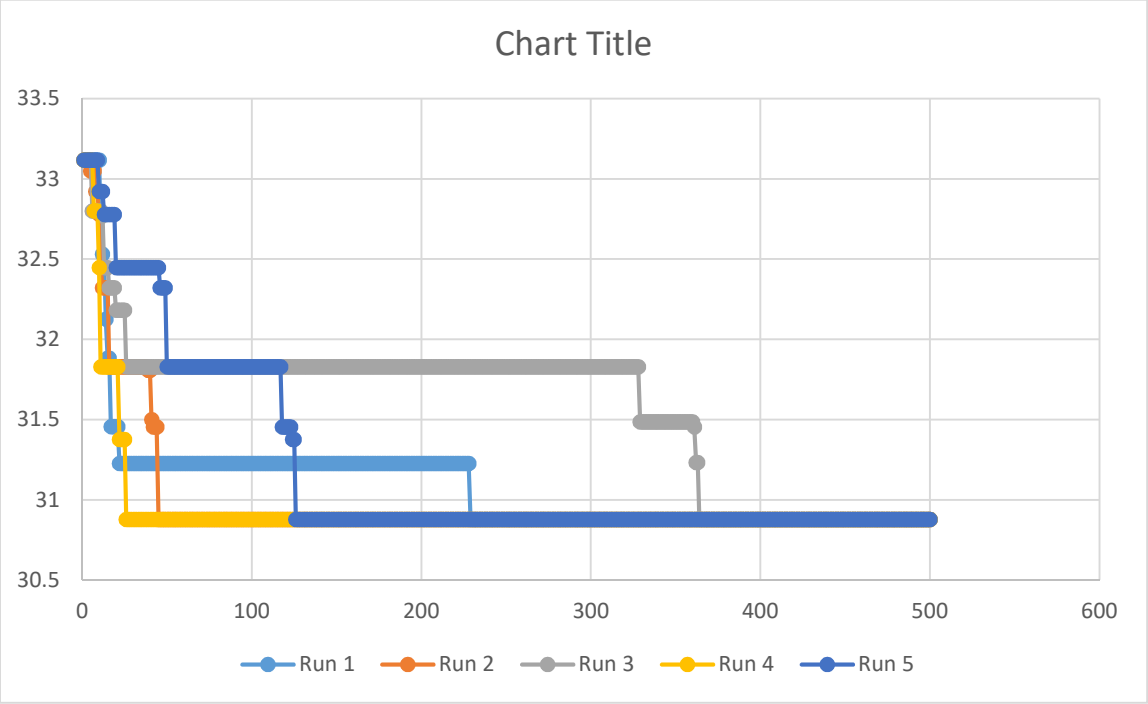
Average Time Taken for execution: 12 seconds

Distance for 5 runs:

	Run 1	Run 2	Run 3	Run 4	Run 5
1	33.11767	33.11767	33.11767	33.11767	33.11767
2	33.11767	33.11767	33.11767	33.11767	33.11767
3	33.11767	33.11767	33.11767	33.11767	33.11767
4	33.11767	33.11767	33.11767	33.11767	33.11767
5	33.11767	33.04998	33.11767	33.11767	33.11767
6	33.11767	33.04998	32.80042	33.11767	33.11767
7	33.11767	33.04998	32.80042	32.80042	33.11767
8	33.11767	32.92138	32.80042	32.80042	33.11767
9	33.11767	32.92138	32.80042	32.80042	33.11767
10	33.11767	32.77711	32.80042	32.44685	32.92138
11	32.77711	32.44685	32.80042	31.82832	32.92138
12	32.53164	32.32049	32.76917	31.82832	32.92138
13	32.44685	32.32049	32.44685	31.82832	32.77711
14	32.12547	32.32049	32.44685	31.82832	32.77711
15	31.88253	32.32049	32.44685	31.82832	32.77711
16	31.88253	31.82832	32.32049	31.82832	32.77711
17	31.45623	31.82832	32.32049	31.82832	32.77711
18	31.45623	31.82832	32.32049	31.82832	32.77711
19	31.45623	31.82832	32.32049	31.82832	32.77711
20	31.45623	31.82832	32.18189	31.82832	32.44685
21	31.45623	31.82832	32.18189	31.82832	32.44685
22	31.22692	31.82832	32.18189	31.37552	32.44685

483	30.8785	30.8785	30.8785	30.8785	30.8785
484	30.8785	30.8785	30.8785	30.8785	30.8785
485	30.8785	30.8785	30.8785	30.8785	30.8785
486	30.8785	30.8785	30.8785	30.8785	30.8785
487	30.8785	30.8785	30.8785	30.8785	30.8785
488	30.8785	30.8785	30.8785	30.8785	30.8785
489	30.8785	30.8785	30.8785	30.8785	30.8785
490	30.8785	30.8785	30.8785	30.8785	30.8785
491	30.8785	30.8785	30.8785	30.8785	30.8785
492	30.8785	30.8785	30.8785	30.8785	30.8785
493	30.8785	30.8785	30.8785	30.8785	30.8785
494	30.8785	30.8785	30.8785	30.8785	30.8785
495	30.8785	30.8785	30.8785	30.8785	30.8785
496	30.8785	30.8785	30.8785	30.8785	30.8785
497	30.8785	30.8785	30.8785	30.8785	30.8785
498	30.8785	30.8785	30.8785	30.8785	30.8785
499	30.8785	30.8785	30.8785	30.8785	30.8785
500	30.8785	30.8785	30.8785	30.8785	30.8785

Graph:



X axis – generations, y axis – distance

Local optima at :[1, 2, 8, 11, 9, 10, 13, 7, 12, 6, 5, 4, 3, 14, 15]

With distance at 30.87

Problem – 2:

```
 NAME: bays29
 TYPE: TSP
 COMMENT: 29 cities in Bavaria, street distances (Groetschel,Juenger,Reinelt)
 DIMENSION: 29
 EDGE_WEIGHT_TYPE: EXPLICIT
 EDGE_WEIGHT_FORMAT: FULL_MATRIX
 DISPLAY_DATA_TYPE: TWOD_DISPLAY

DISPLAY_DATA_SECTION
   1    1150.0  1760.0
   2     630.0  1660.0
   3      40.0  2090.0
   4     750.0  1100.0
   5     750.0  2030.0
   6    1030.0  2070.0
   7    1650.0   650.0
   8    1490.0  1630.0
   9     790.0  2260.0
  10     710.0  1310.0
  11     840.0   550.0
  12    1170.0  2300.0
  13     970.0  1340.0
  14     510.0   700.0
  15     750.0   900.0
  16    1280.0  1200.0
  17     230.0   590.0
  18     460.0   860.0
  19    1040.0   950.0
  20     590.0  1390.0
  21     830.0  1770.0
  22     490.0   500.0
  23    1840.0  1240.0
  24    1260.0  1500.0
  25    1280.0   790.0
  26     490.0  2130.0
  27    1460.0  1420.0
  28    1260.0  1910.0
  29     360.0  1980.0
EOF
```

INPUT:

```
29
   1    1150.0  1760.0
   2     630.0  1660.0
   3      40.0  2090.0
   4     750.0  1100.0
   5     750.0  2030.0
   6    1030.0  2070.0
   7    1650.0   650.0
   8    1490.0  1630.0
   9     790.0  2260.0
  10     710.0  1310.0
  11     840.0   550.0
  12    1170.0  2300.0
  13     970.0  1340.0
  14     510.0   700.0
  15     750.0   900.0
  16    1280.0  1200.0
  17     230.0   590.0
  18     460.0   860.0
  19    1040.0   950.0
  20     590.0  1390.0
  21     830.0  1770.0
  22     490.0   500.0
  23    1840.0  1240.0
  24    1260.0  1500.0
  25    1280.0   790.0
  26     490.0  2130.0
  27    1460.0  1420.0
  28    1260.0  1910.0
  29    360.0   1980.0
[1, 24, 16, 13, 4, 15, 11,
500
```

OUTPUT:

```
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
9353.21886025498
[1, 28, 12, 9, 26, 3, 29, 2, 20, 10, 4, 15, 18, 14, 17, 22, 11, 19, 25, 7, 23, 27, 8, 24, 16, 13, 21, 5, 6, 30]
Tue May  2 23:23:05 2017
```

Solution:

[1, 24, 16, 13, 4, 15, 11, 22, 17, 14, 18, 19, 25, 7, 23, 27, 8, 28, 12, 6, 9, 5, 26, 2, 20, 10, 3, 29, 21, 30] where 30 is city 1

Average Time taken for execution: 2 min 22s

Distance for 5 runs:

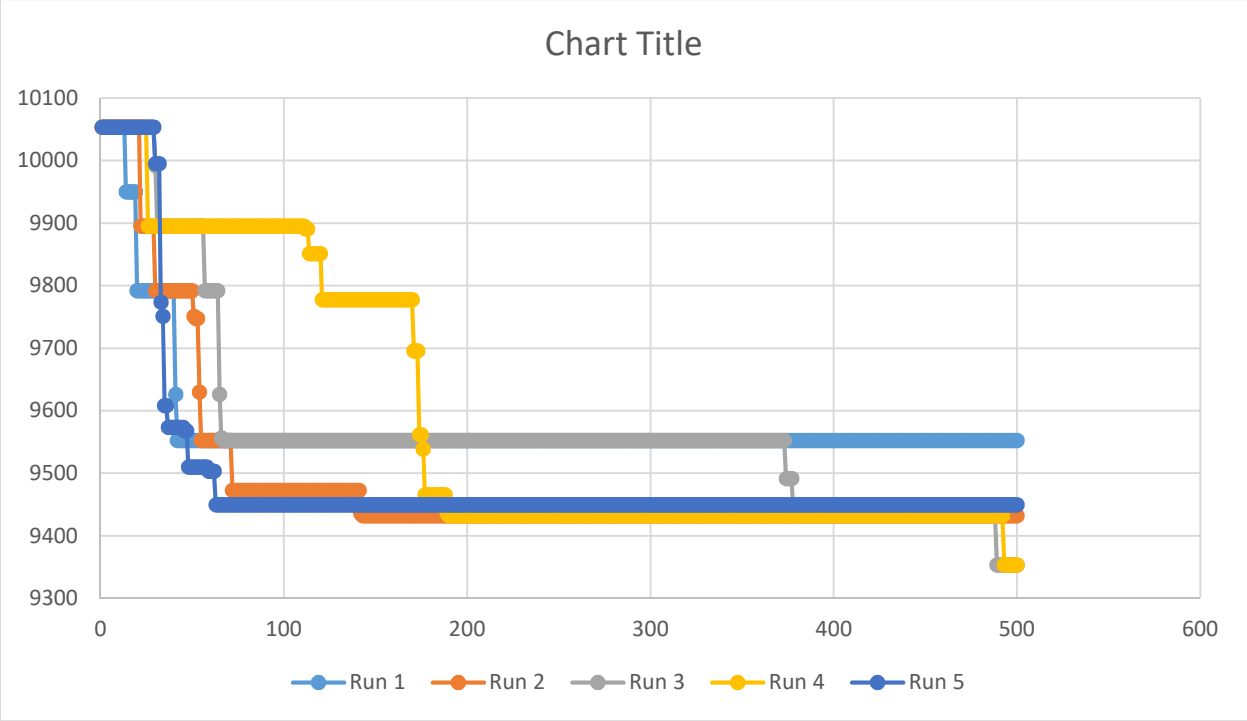
	Run 1	Run 2	Run 3	Run 4	Run 5
1	10053.37	10053.37	10053.37	10053.37	10053.37
2	10053.37	10053.37	10053.37	10053.37	10053.37
3	10053.37	10053.37	10053.37	10053.37	10053.37
4	10053.37	10053.37	10053.37	10053.37	10053.37
5	10053.37	10053.37	10053.37	10053.37	10053.37
6	10053.37	10053.37	10053.37	10053.37	10053.37
7	10053.37	10053.37	10053.37	10053.37	10053.37
8	10053.37	10053.37	10053.37	10053.37	10053.37
9	10053.37	10053.37	10053.37	10053.37	10053.37
10	10053.37	10053.37	10053.37	10053.37	10053.37
11	10053.37	10053.37	10053.37	10053.37	10053.37
12	10053.37	10053.37	10053.37	10053.37	10053.37
13	10053.37	10053.37	10053.37	10053.37	10053.37
14	9949.691	10053.37	10053.37	10053.37	10053.37
15	9949.691	10053.37	10053.37	10053.37	10053.37
16	9949.691	10053.37	10053.37	10053.37	10053.37
17	9949.691	10053.37	10053.37	10053.37	10053.37
18	9949.691	10053.37	10053.37	10053.37	10053.37
19	9949.691	10053.37	10053.37	10053.37	10053.37
20	9791.769	10053.37	10053.37	10053.37	10053.37

481	9552.575	9431.95	9431.95	9431.95	9449.775
482	9552.575	9431.95	9431.95	9431.95	9449.775
483	9552.575	9431.95	9431.95	9431.95	9449.775
484	9552.575	9431.95	9431.95	9431.95	9449.775
485	9552.575	9431.95	9431.95	9431.95	9449.775
486	9552.575	9431.95	9431.95	9431.95	9449.775
487	9552.575	9431.95	9431.95	9431.95	9449.775
488	9552.575	9431.95	9431.95	9431.95	9449.775
489	9552.575	9431.95	9353.219	9431.95	9449.775
490	9552.575	9431.95	9353.219	9431.95	9449.775
491	9552.575	9431.95	9353.219	9431.95	9449.775
492	9552.575	9431.95	9353.219	9431.95	9449.775
493	9552.575	9431.95	9353.219	9353.219	9449.775
494	9552.575	9431.95	9353.219	9353.219	9449.775
495	9552.575	9431.95	9353.219	9353.219	9449.775
496	9552.575	9431.95	9353.219	9353.219	9449.775
497	9552.575	9431.95	9353.219	9353.219	9449.775
498	9552.575	9431.95	9353.219	9353.219	9449.775
499	9552.575	9431.95	9353.219	9353.219	9449.775
500	9552.575	9431.95	9353.219	9353.219	9449.775

Generations: 500

Population each generation : 290

Graph:



X axis – generations, y axis – distance

Local optima at:[1, 24, 16, 13, 4, 15, 11, 22, 17, 14, 18, 19, 25, 7, 23, 27, 8, 28, 12, 6, 9, 5, 26, 2, 20, 10, 3, 29, 21, 30]

With distance at 9353.22

Problem – 3:

```
NAME: ulysses22.tsp
TYPE: TSP
COMMENT: Odyssey of Ulysses (Groetschel/Padberg)
DIMENSION: 22
EDGE_WEIGHT_TYPE: GEO
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
 1 38.24 20.42
 2 39.57 26.15
 3 40.56 25.32
 4 36.26 23.12
 5 33.48 10.54
 6 37.56 12.19
 7 38.42 13.11
 8 37.52 20.44
 9 41.23 9.10
 10 41.17 13.05
 11 36.08 -5.21
 12 38.47 15.13
 13 38.15 15.35
 14 37.51 15.17
 15 35.49 14.32
 16 39.36 19.56
 17 38.09 24.36
 18 36.09 23.00
 19 40.44 13.57
 20 40.33 14.15
 21 40.37 14.23
 22 37.57 22.56
```

INPUT:

```
22
 1 38.24 20.42
 2 39.57 26.15
 3 40.56 25.32
 4 36.26 23.12
 5 33.48 10.54
 6 37.56 12.19
 7 38.42 13.11
 8 37.52 20.44
 9 41.23 9.10
 10 41.17 13.05
 11 36.08 -5.21
 12 38.47 15.13
 13 38.15 15.35
 14 37.51 15.17
 15 35.49 14.32
 16 39.36 19.56
 17 38.09 24.36
 18 36.09 23.00
 19 40.44 13.57
 20 40.33 14.15
 21 40.37 14.23
 22 37.57 22.56
[1, 8, 22, 17, 4, 18, 2,
23]
500
```

OUTPUT:

```
76.44090878732085
76.44090878732085
76.44090878732085
76.44090878732085
76.44090878732085
76.44090878732085
76.44090878732085
76.44090878732085
76.44090878732085
[1, 8, 22, 18, 4, 17, 2, 3, 16, 14, 13, 12, 21, 20, 19, 10, 9, 11, 5, 6, 7, 15, 23]
Wed May  3 01:06:52 2017
```

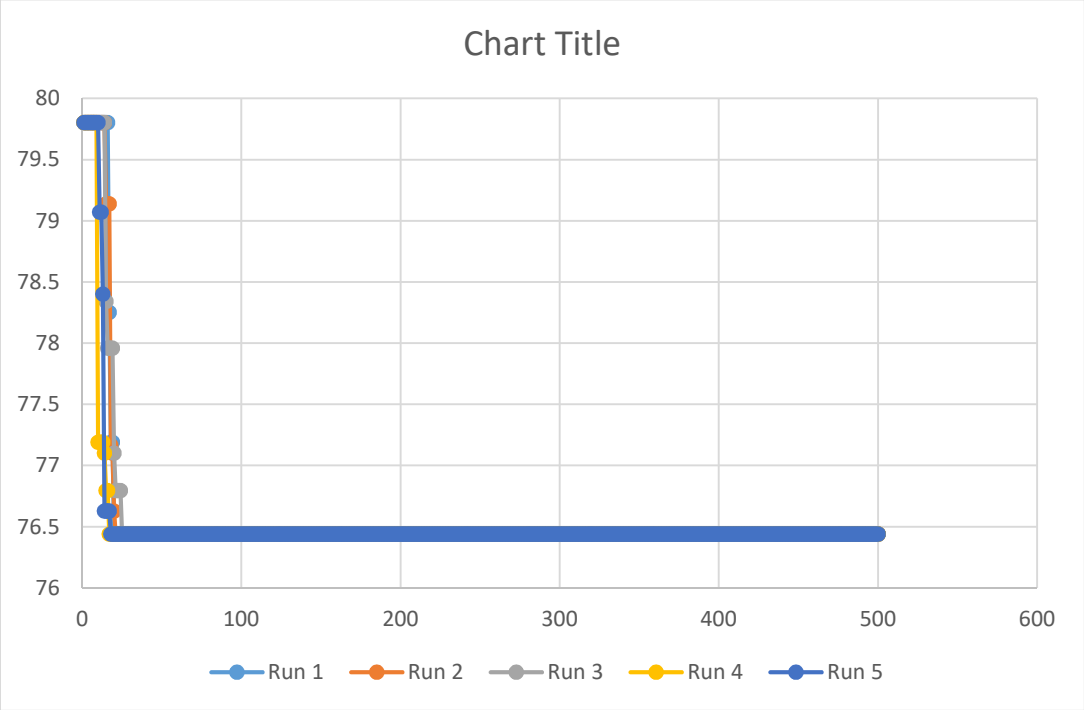
Solution: [1,8,22,4,17,2,3,16,14,13,12,21,20,19,10,9,11,5,6,7,15,23]

Time taken for execution: 1 min 25 s

Distance for 5 runs:

	Run 1	Run 2	Run 3	Run 4	Run 5
1	79.80458	79.80458	79.80458	79.80458	79.80458
2	79.80458	79.80458	79.80458	79.80458	79.80458
3	79.80458	79.80458	79.80458	79.80458	79.80458
4	79.80458	79.80458	79.80458	79.80458	79.80458
5	79.80458	79.80458	79.80458	79.80458	79.80458
6	79.80458	79.80458	79.80458	79.80458	79.80458
7	79.80458	79.80458	79.80458	79.80458	79.80458
8	79.80458	79.80458	79.80458	79.80458	79.80458
9	79.80458	79.80458	79.80458	79.80458	79.80458
10	79.80458	79.80458	79.80458	77.19238	79.80458
11	79.80458	79.80458	79.80458	77.19238	79.07329
12	79.80458	79.80458	79.80458	77.19238	79.07329
13	79.80458	79.80458	79.80458	77.19238	78.40393
14	79.80458	79.80458	79.80458	77.10368	76.62996
15	79.80458	79.14181	78.34229	76.79636	76.62996
16	79.80458	79.14181	77.96081	76.79636	76.62996
17	78.25442	79.14181	77.96081	76.44091	76.62996
18	77.19238	77.16532	77.96081	76.44091	76.44091
19	77.19238	77.10368	77.96081	76.44091	76.44091
20	76.44091	76.62996	77.10368	76.44091	76.44091
21	76.44091	76.44091	76.79636	76.44091	76.44091
479	76.44091	76.44091	76.44091	76.44091	76.44091
480	76.44091	76.44091	76.44091	76.44091	76.44091
481	76.44091	76.44091	76.44091	76.44091	76.44091
482	76.44091	76.44091	76.44091	76.44091	76.44091
483	76.44091	76.44091	76.44091	76.44091	76.44091
484	76.44091	76.44091	76.44091	76.44091	76.44091
485	76.44091	76.44091	76.44091	76.44091	76.44091
486	76.44091	76.44091	76.44091	76.44091	76.44091
487	76.44091	76.44091	76.44091	76.44091	76.44091
488	76.44091	76.44091	76.44091	76.44091	76.44091
489	76.44091	76.44091	76.44091	76.44091	76.44091
490	76.44091	76.44091	76.44091	76.44091	76.44091
491	76.44091	76.44091	76.44091	76.44091	76.44091
492	76.44091	76.44091	76.44091	76.44091	76.44091
493	76.44091	76.44091	76.44091	76.44091	76.44091
494	76.44091	76.44091	76.44091	76.44091	76.44091
495	76.44091	76.44091	76.44091	76.44091	76.44091
496	76.44091	76.44091	76.44091	76.44091	76.44091
497	76.44091	76.44091	76.44091	76.44091	76.44091
498	76.44091	76.44091	76.44091	76.44091	76.44091
499	76.44091	76.44091	76.44091	76.44091	76.44091
500	76.44091	76.44091	76.44091	76.44091	76.44091

Graph:



X axis – generations, y axis – distance

Local optima at: [1, 8, 22, 18, 4, 17, 2, 3, 16, 14, 13, 12, 21, 20, 19, 10, 9, 11, 5, 6, 7, 15, 23]

With distance 76.44091

Generations: 500

Population each generation : 220

Problem – 4:

```
NAME: berlin52
TYPE: TSP
COMMENT: 52 locations in Berlin (Groetschel)
DIMENSION: 52
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
```

```
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
5 845.0 655.0
6 880.0 660.0
7 25.0 230.0
8 525.0 1000.0
9 580.0 1175.0
10 650.0 1130.0
11 1605.0 620.0
12 1220.0 580.0
13 1465.0 200.0
14 1530.0 5.0
15 845.0 680.0
16 725.0 370.0
17 145.0 665.0
18 415.0 635.0
19 510.0 875.0
20 560.0 365.0
21 300.0 465.0
22 520.0 585.0
23 480.0 415.0
```

24 835.0 625.0
25 975.0 580.0
26 1215.0 245.0
27 1320.0 315.0
28 1250.0 400.0
29 660.0 180.0
30 410.0 250.0
31 420.0 555.0
32 575.0 665.0
33 1150.0 1160.0
34 700.0 580.0
35 685.0 595.0
36 685.0 610.0
37 770.0 610.0
38 795.0 645.0
39 720.0 635.0
40 760.0 650.0
41 475.0 960.0
42 95.0 260.0
43 875.0 920.0
44 700.0 500.0
45 555.0 815.0
46 830.0 485.0
47 1170.0 65.0
48 830.0 610.0
49 605.0 625.0
50 595.0 360.0
51 1340.0 725.0
52 1740.0 245.0

INPUT:

```
22 520.0 585.0
23 480.0 415.0
24 835.0 625.0
25 975.0 580.0
26 1215.0 245.0
27 1320.0 315.0
28 1250.0 400.0
29 660.0 180.0
30 410.0 250.0
31 420.0 555.0
32 575.0 665.0
33 1150.0 1160.0
34 700.0 580.0
35 685.0 595.0
36 685.0 610.0
37 770.0 610.0
38 795.0 645.0
39 720.0 635.0
40 760.0 650.0
41 475.0 960.0
42 95.0 260.0
43 875.0 920.0
44 700.0 500.0
45 555.0 815.0
46 830.0 485.0
47 1170.0 65.0
48 830.0 610.0
49 605.0 625.0
50 595.0 360.0
51 1340.0 725.0
52 1740.0 245.0
[1, 22, 32, 49, 36
 25, 12, 28, 27, 1
500
```

OUTPUT:

```
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
8258.435303854205
[1, 22, 32, 49, 36, 35, 34, 39, 40, 37, 38, 48, 24, 5, 15, 6, 4, 25, 46, 44, 16
 12, 28, 27, 11, 52, 14, 13, 26, 47, 29, 30, 2, 7, 42, 21, 31, 53]
Wed May  3 09:16:42 2017
```

Generations: 500

Population each generation: 520

Average time for execution: 25 minutes

Local optima: [1, 22, 32, 49, 36,35,34,39,40,37,38,48,24,5,15,6,4,25,46,44,16,12,28,27,11,52,14,13, 26,47,29,30,2,7,42,21,31,53]

Minimum Distance: 8258.435

Distance:

	Run 1	Run 2	Run 3	Run 4
1	10217.82	10217.82	10217.82	10217.82
2	10217.82	10217.82	10217.82	10217.82
3	10217.82	10217.82	10217.82	10217.82
4	10217.82	10217.82	10217.82	10217.82
5	10217.82	10217.82	10217.82	10217.82
6	10217.82	10217.82	10217.82	10217.82
7	10217.82	10217.82	10217.82	10217.82
8	10217.82	10217.82	10217.82	10217.82
9	10217.82	10217.82	10217.82	10217.82
10	10217.82	10217.82	10217.82	10217.82
11	10217.82	10217.82	10217.82	10217.82
12	10217.82	10217.82	10217.82	10217.82
13	10217.82	10217.82	10217.82	10217.82
14	10217.82	10217.82	10217.82	10217.82
15	10217.82	10217.82	10217.82	10217.82
16	10217.82	10217.82	10217.82	10217.82
17	10217.82	10217.82	10217.82	10217.82
18	10217.82	10217.82	10217.82	10217.82
19	10217.82	10217.82	10217.82	10217.82
20	10217.82	10217.82	10217.82	10217.82
21	10217.82	10217.82	10217.82	10217.82
22	10217.82	10217.82	10217.82	10217.82
23	10217.82	10217.82	10208.5	10200.91
24	10217.82	10217.82	10172.82	10168.62

486	8258.43	8258.43	9769.24	8258.43
487	8258.43	8258.43	9769.24	8258.43
488	8258.43	8258.43	9769.24	8258.43
489	8258.43	8258.43	9769.24	8258.43
490	8258.43	8258.43	9769.24	8258.43
491	8258.43	8258.43	9769.24	8258.43
492	8258.43	8258.43	9769.24	8258.43
493	8258.43	8258.43	9769.24	8258.43
494	8258.43	8258.43	9769.24	8258.43
495	8258.43	8258.43	9769.24	8258.43
496	8258.43	8258.43	9769.24	8258.43
497	8258.43	8258.43	9769.24	8258.43
498	8258.43	8258.43	9769.24	8258.43
499	8258.43	8258.43	9769.24	8258.43
500	8258.43	8258.43	9769.24	8258.43

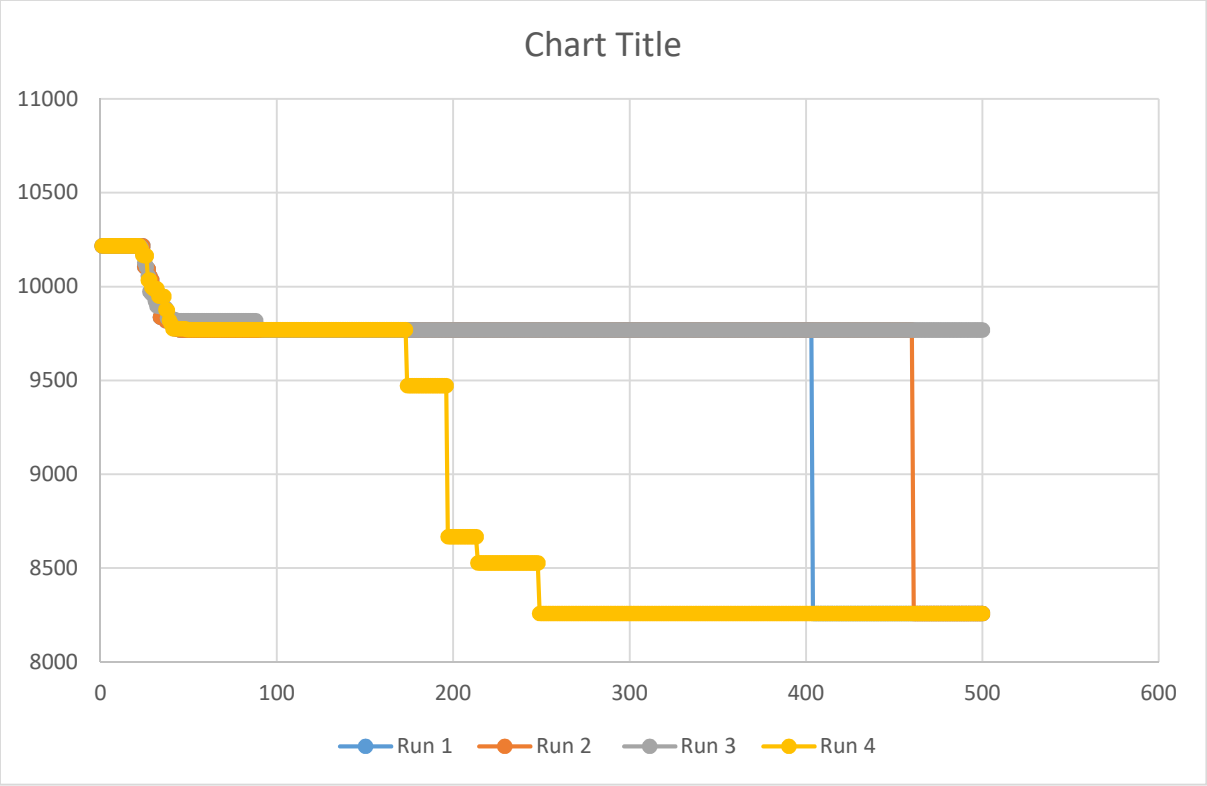
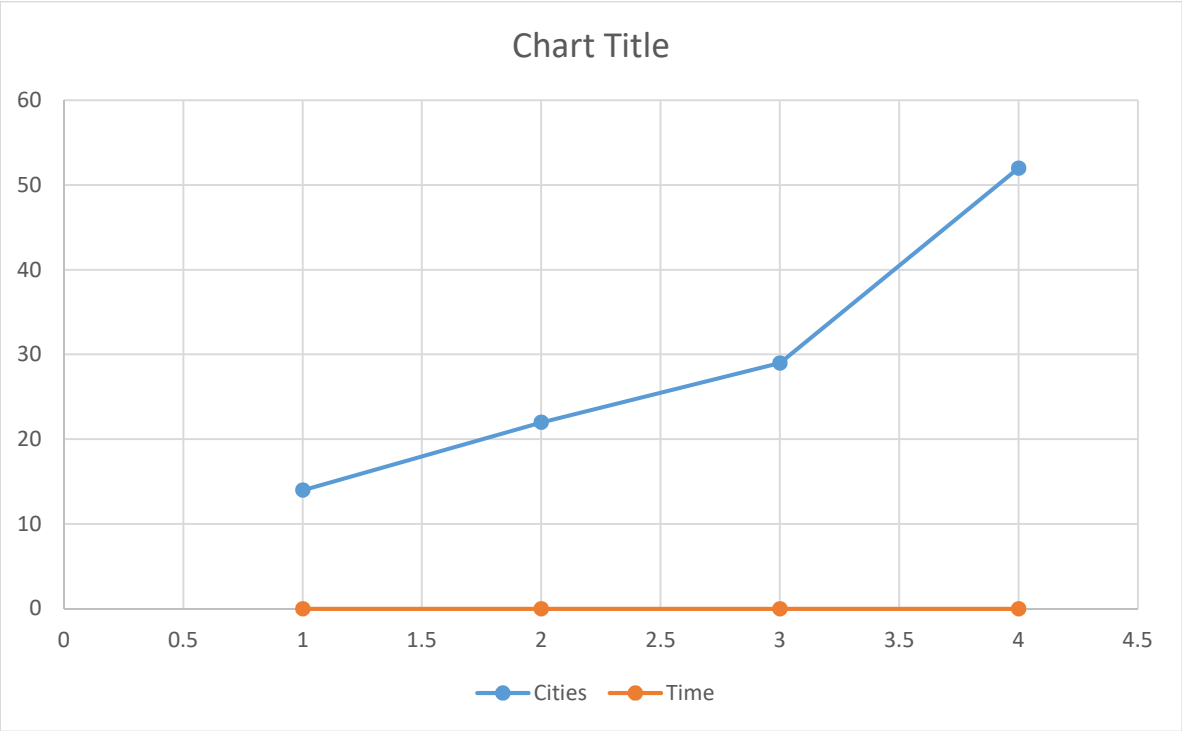


Chart Title
11000
10500
10000
9500
9000
8500
8000
0
100
200
300
400
500
600
Run 1
Run 2
Run 3
Run 4

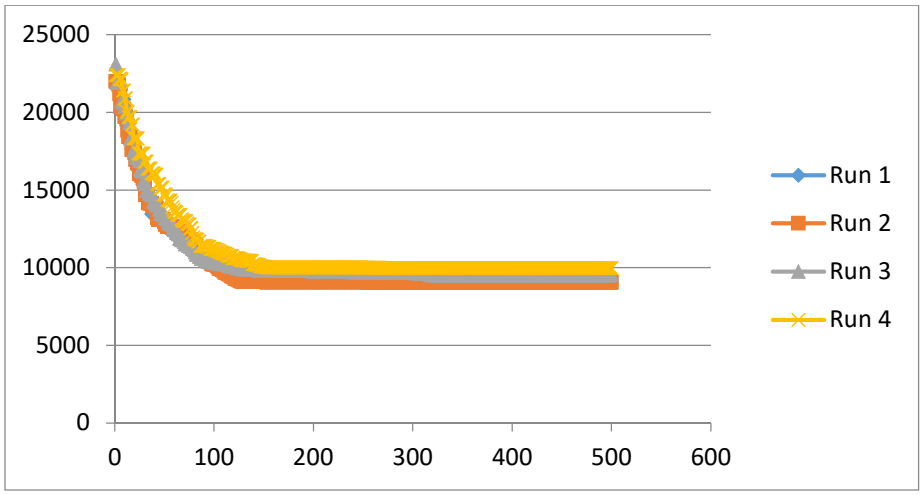
Run-time for increasing number of cities:

	Time
14	12s
22	142s
29	85s
52	1500s



Comparison with normal GA:

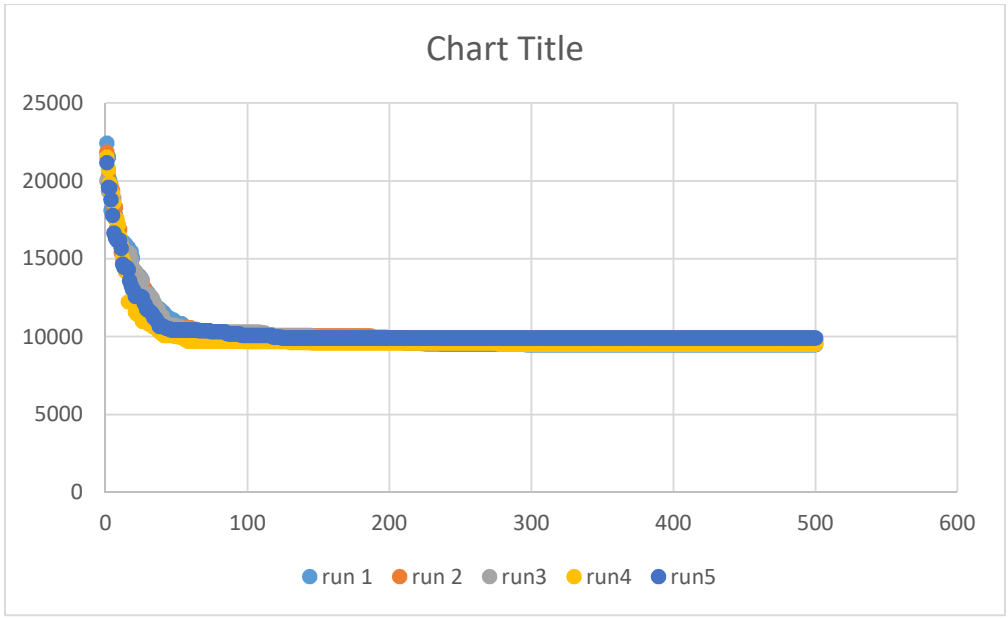
Berlin 52



Minimum distance: 9055.937 compared to 8258.43 in GA+Greedy+BnB

Average time: 250.24 s

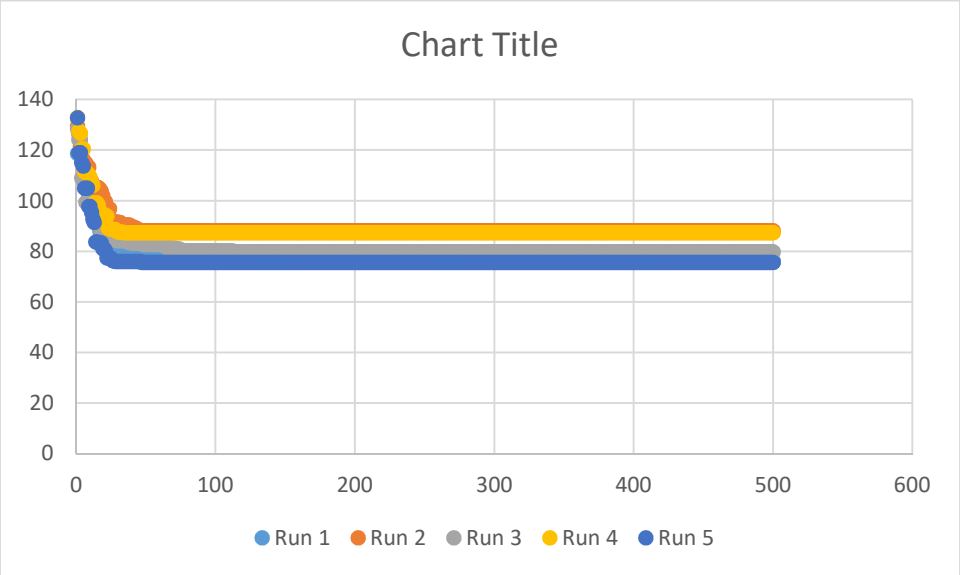
Bays29



Minimum distance: 9931.008 compared to 9353.22

Average time: 80.44s

Ulysses22

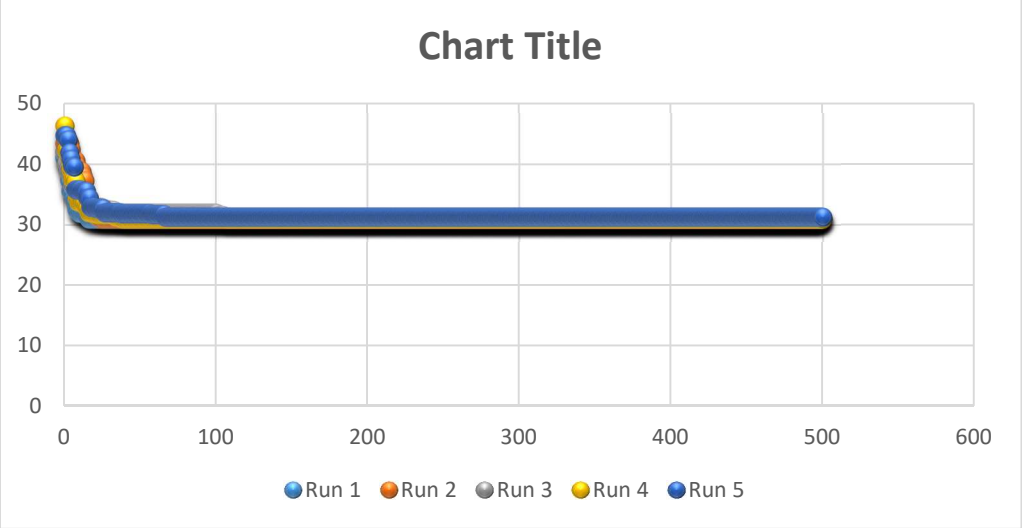


Average time: 48.21s

Minimum Distance: 75.66 compared to 76.44

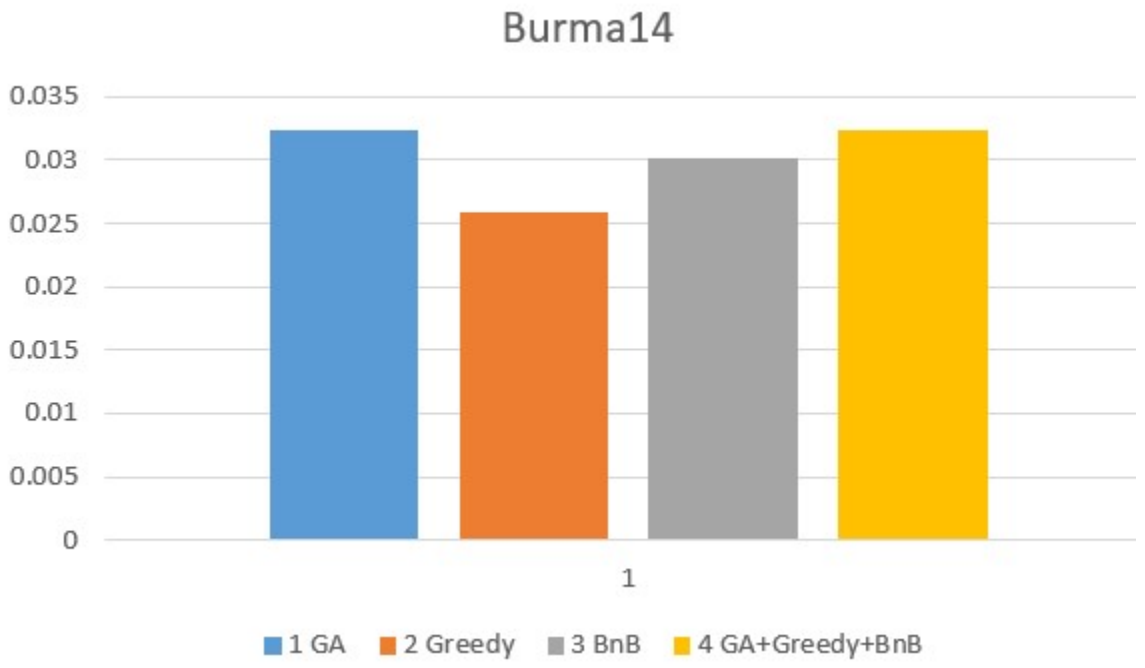
Better solution: [1, 16, 21, 20, 19, 10, 9, 11, 5, 15, 6, 7, 12, 13, 14, 8, 18, 4, 17, 2, 3, 22, 23]

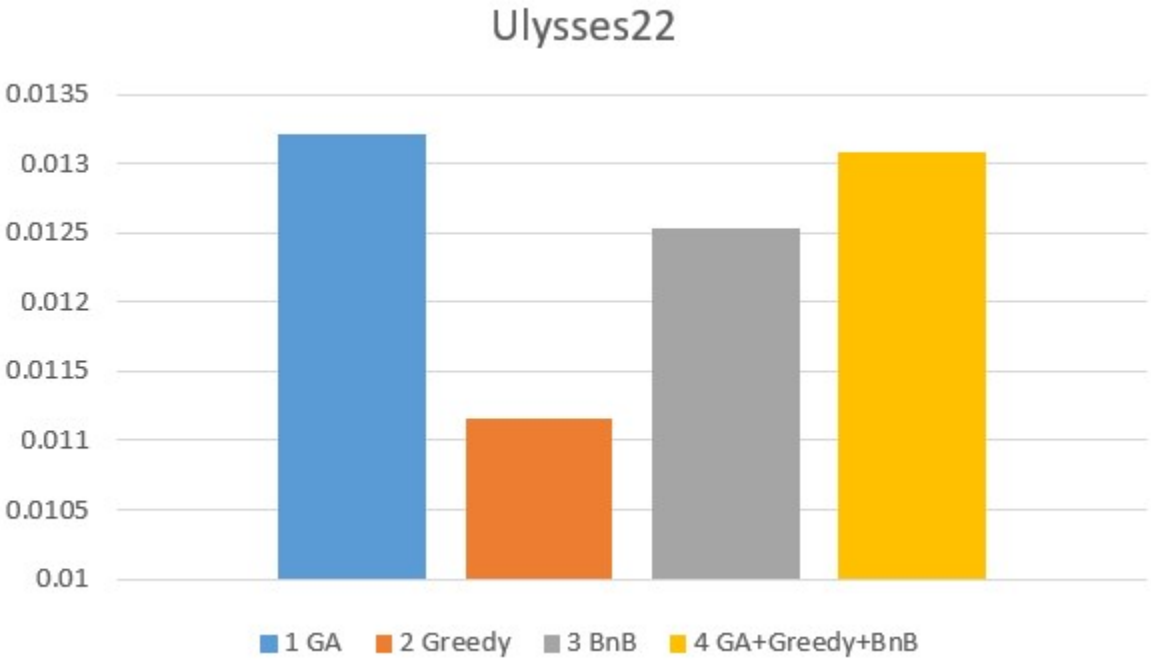
Burma14

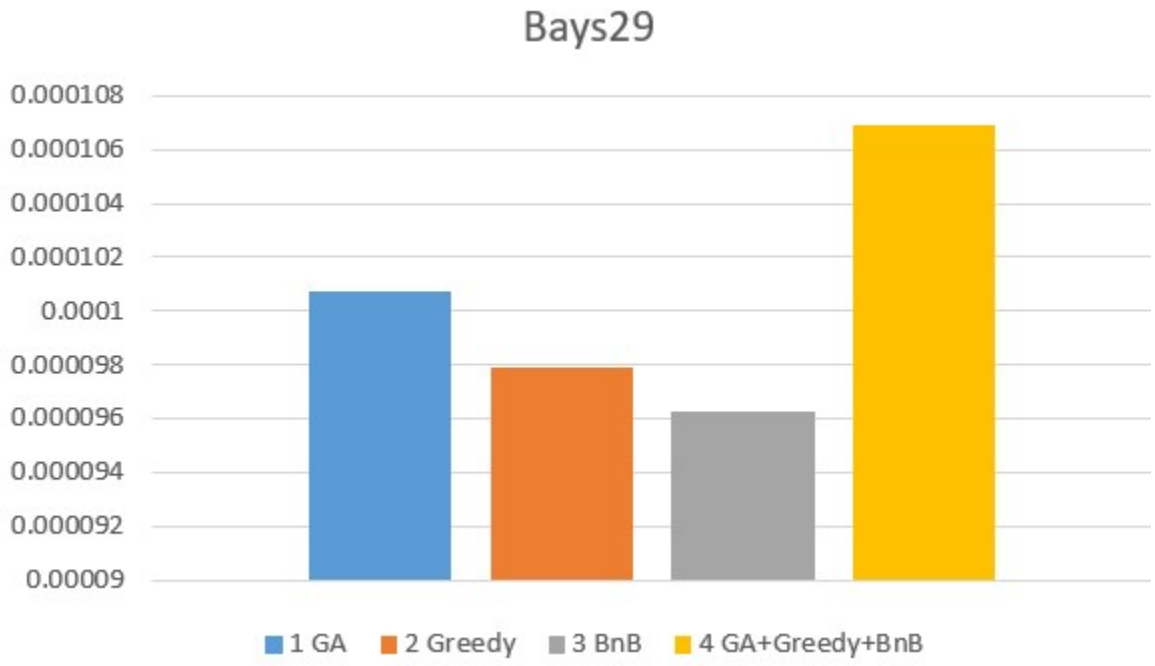


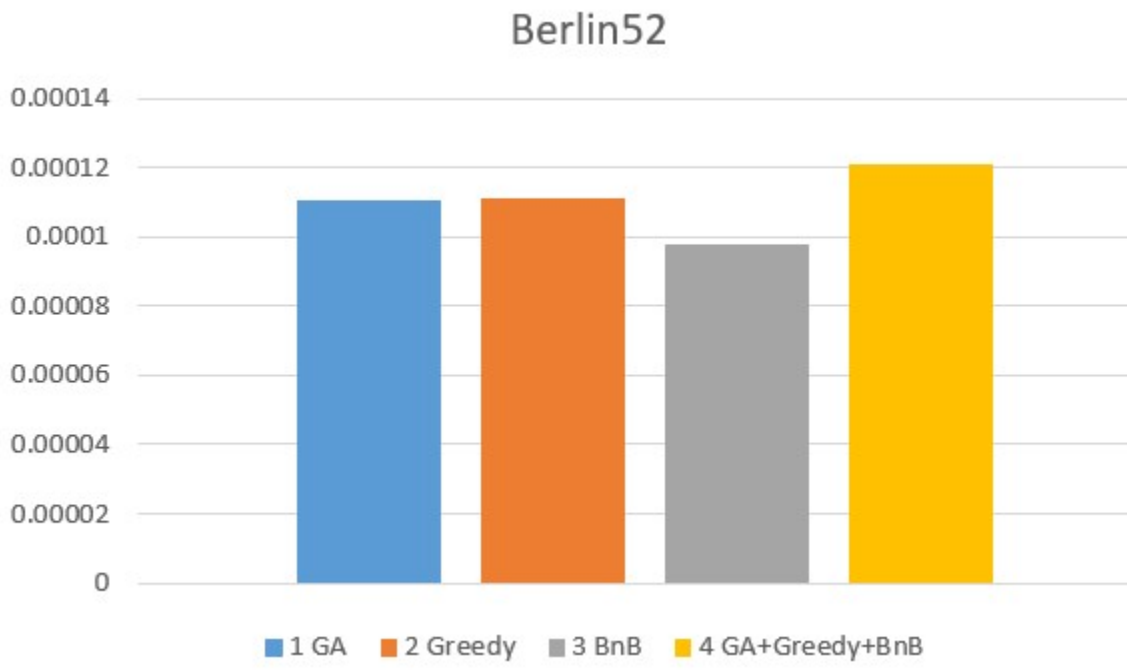
Minimum distance 30.8785 same without greedy and branch and bound seed.

Y axis- fitness value









Where fitness = 1/minimum distance obtained, so more optimal the distance, higher the fitness.

As we can see, GA and GA with Greedy and BnB are more efficient than Greedy or Bnb alone.

Conclusion:

From the graphs we can observe that although the normal Genetic Algorithm code for TSP works faster than the code for greedy and branch and bound, it is way far from the optimum solution. In some cases like Ulysses22 cities problem, the normal GA algorithm provided a better solution. For some problems like Burma14 with very less number of cities, both the algorithms provided optimum solutions in the number of generations specified.

We can hereby conclude that for problems with larger number of cities, the GA with greedy and branch and bound seed, code for TSP works way better than the normal GA code.