**Assignment 1 Report**

**Part 1: NP-completeness**

Given:

F = (z​1​ ∨z​2)​ ∧(z​1​ ∨z​2​ ∨z​3​ ∨z​4)​

Solution:

NP completeness: There is no solution in polynomial time. (Not Solvable)

Clauses:

To convert the formula F into 3COL we define clauses.

Above formula(F) will have 4 clauses.

They are as below,

1. (z​1​ ∨z​2 ∨A)
2. (z​1​ ∨z​2 ∨-A)
3. (z​1​ ∨z​2 ∨B)
4. (z​3∨z​4∨-B)

So, our formula F into 3COL is (z​1​ ∨z​2 ∨A) ^ (z​1​ ∨z​2 ∨-A) ^ (z​1​ ∨z​2 ∨B) ^ (z​3∨z​4∨-B)

Description:

3COL is NP-Complete.

Step 1: 3COL is NP

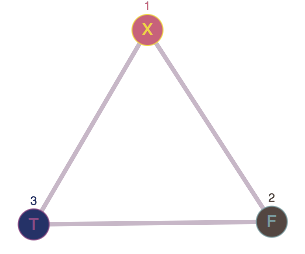
We colour the vertices so that every connected vertices will have a different colour.

Step 2: We reduce to 3COL

Step 3: Reduce 3SAT to 3COL

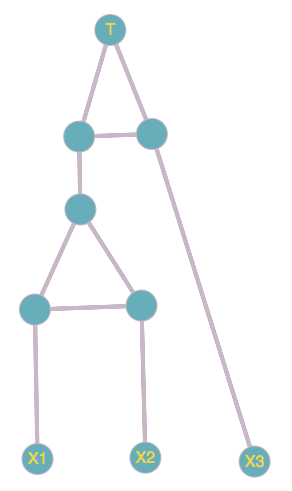
We use following gadgets:

3 special vertices are T(True), F(False) and X



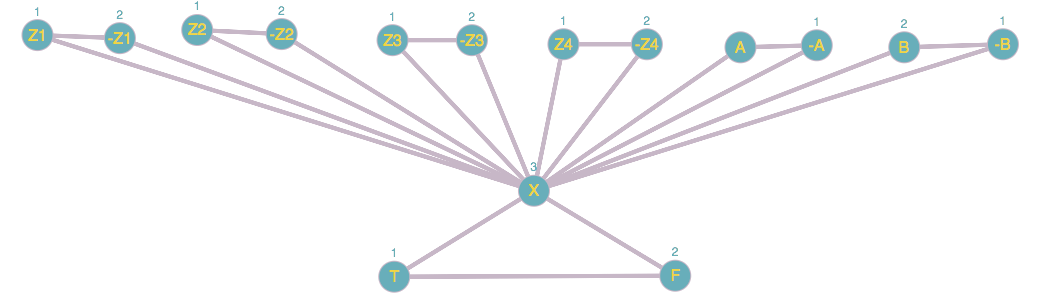
The OR Gadget:

OR Gadgets are used to encode the clauses.

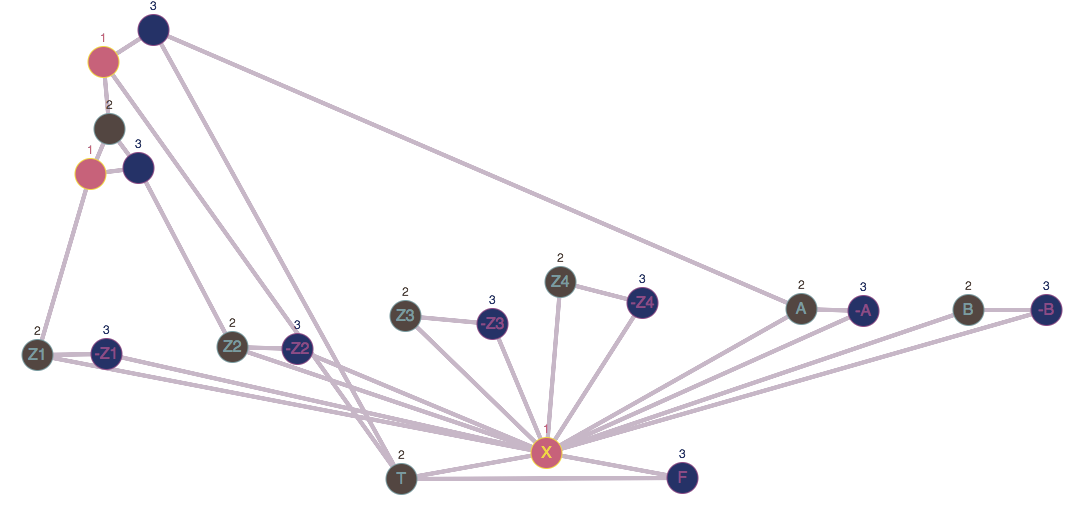


Steps:

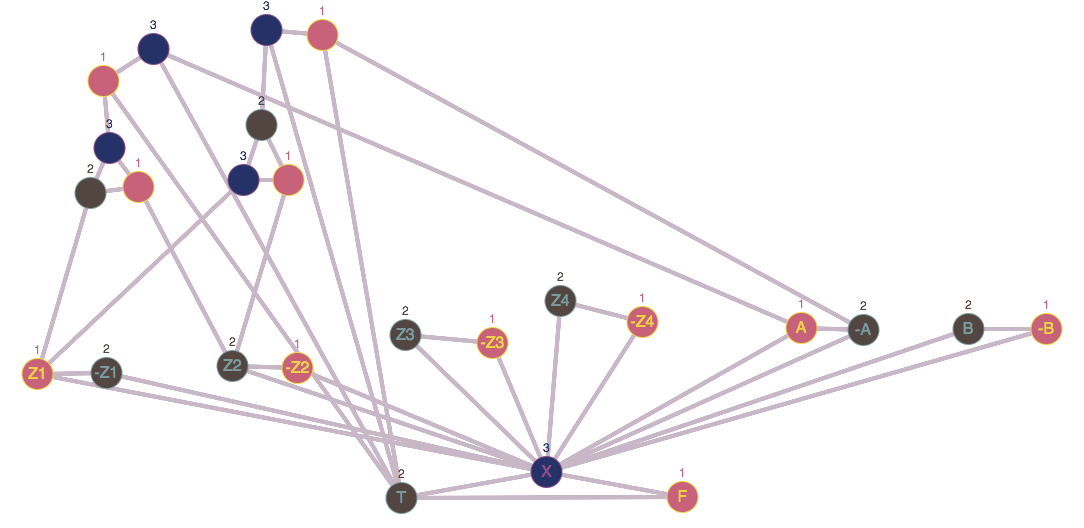
1. Adding all the vertices to our 3 special vertices



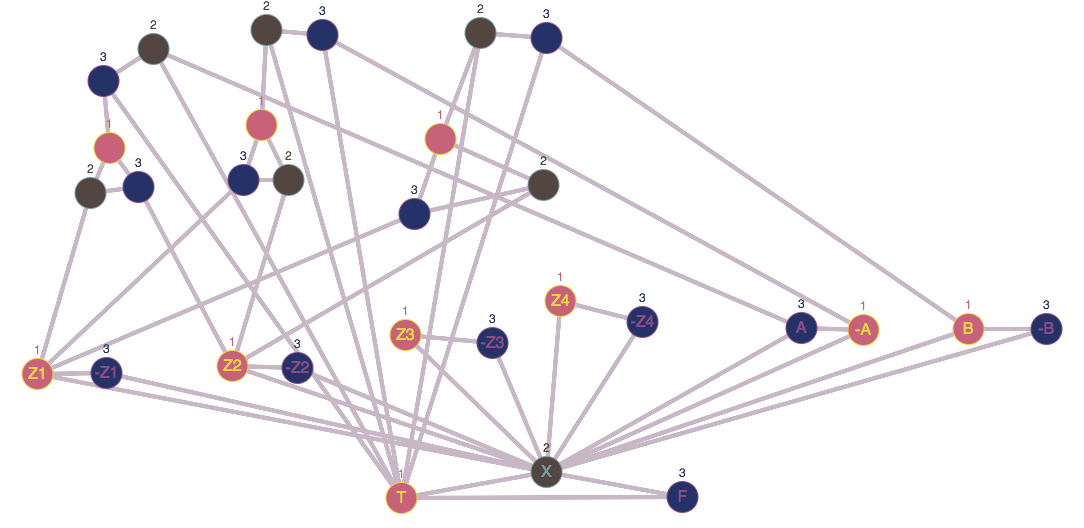
1. Creating a clause in graph for (z​1​ ∨z​2 ∨A)



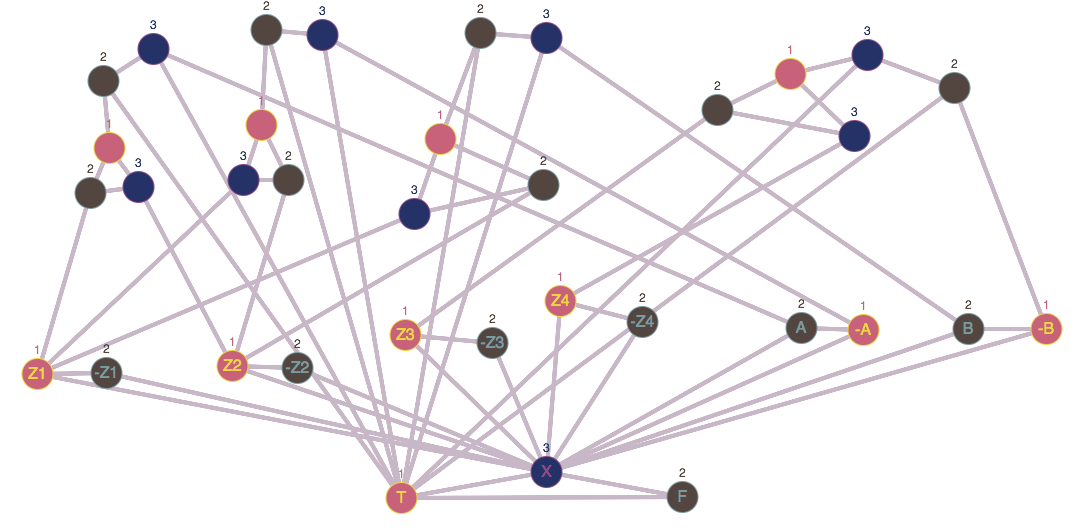
1. Creating a clause in graph for (z​1​ ∨z​2 ∨-A)



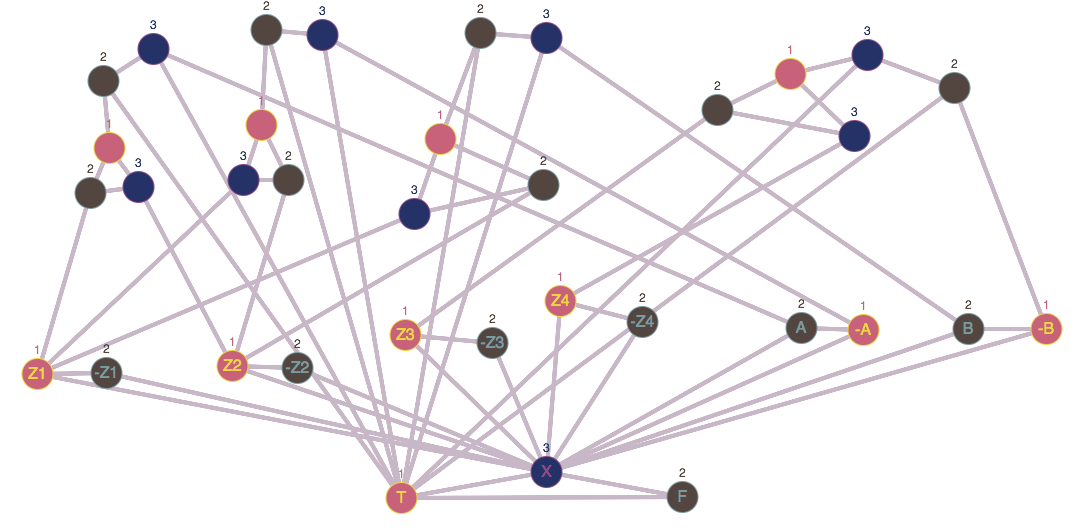
1. Creating a clause in graph for (z​1​ ∨z​2 ∨B)



1. Creating a clause in graph for (z​1​ ∨z​2 ∨-B)



Final Graph:



**Following table is the proof that a solution for the 3COL instance is a solution for F.**

**All the clauses are “True” based on the following table:**

|  |  |
| --- | --- |
| **Vertexes** | **Truth Value** |
| Z1 | T |
| Z2 | T |
| Z3 | F |
| Z4 | T |
| A | T |
| -A | T |
| B | T |
| -B | F |

**Explanation:**

**A clause is True when one of the elements is true.**

(z​1​ ∨z​2 ∨A) is true because all the elements are T.

(z​1​ ∨z​2 ∨-A) is true because all the elements are T.

(z​1​ ∨z​2 ∨B) is true because all the elements are T.

(z​3∨z​4∨-B) is true because Z4 is T.

Since, All the clauses are True we say that the solution for 3COL is indeed a solution for formula F.

**Part 2: Genetic Algorithm**

**Description:**

Genetic Algorithms are used to find a good and robust solution rated against fitness criteria.

It avoids local optima and searches for global fitness.

It’s an optimization technique based on Darwin’s principle of natural selection.

**TSP(Travelling Salesman Problem):**

We find the shortest possible route that visits each city and returns to the origin city based on a list of cities and distance between each pair of cities.

Basically, We have different orders of many cities and we want to find the best route.

**Rules:**

1. We can visit each city exactly once.
2. To calculate total distance we need to return to the first city.

**Methodology:**

Few Terminologies used in the program:

Gene: a city represented as (x, y) coordinates

Individual(Chromosome): a single route satisfying the above conditions

Population: Collection of individuals

Mating Pool: a collection of parents that are used to create our next generation routes

Fitness: a function that tells us how short the distance is

Mutation: randomly swapping two cities in a route to find variations in the population

Elitism: passing the best individuals into the next generation

**Steps of Genetic Algorithm:**

1. Population Creation
2. Calculate Fitness
3. Select Mating Pool
4. Breed
5. Mutate
6. Repeat

**Components of Genetic Algorithm:**

1. Selection

Selects the individuals(Parents) that contribute to population of next generation.

1. Crossover

Combines two parents to form children for the next generation.

1. Mutation

Apply random changes to individual parents to form children.

**Algorithms used to implement Selection, Crossover and Mutation:**

**Roulette Wheel:**

* Parents are selected according to their fitness
* The better the chromosomes, the more chances to be selected

**Algorithm implemented:**

* Calculate the sum of fitness
* Find maximum fitness value by looping through the mating pool and keeping only the highest fitness value
* Select the Random number between (0,1)
* Calculate the probability using the following formula

 Where N is the number of individuals in the population.

* Calculate the partial sum
* Normalize by

1. Adding 1 to the maximum fitness and subtracting the probability value from it.
2. Dividing the result by the sum of fitness
3. Adding that result to the total fitness variable

* Compare if total fitness is greater than random number and return that individual

**Uniform-Based crossover:**We treat each gene separately rather than dividing chromosomes into segments.

* We randomly select which chromosomes to include in the off-spring.

**Algorithm implemented:**

* Select a random number between 0 and length of parent – 1
* Select random index based on the above random number selected
* Copy the values of selected indexes in the off-spring
* Now, copy the remaining values in alternate off-springs
* Return the off-spring

**Cycle Crossover:**

* Each element comes from one parent together with its position

**Algorithm implemented:**

* Identify a cycle in parent
* Copy the elements of first cycle as it is in the off-spring
* Check for the empty values in the off-spring
* Identify the second cycle
* Copy the remaining values of parents in the alternate off-spring

**Scramble Mutation:**

* A small random twist in the chromosome to get a new solution
* Applied with a low probability
* Used to maintain and introduce diversity in the genetic population

**Algorithm implemented:**

* Select a subset of genes
* Use shuffle operator to shuffle the values
* Return parent

**Reciprocal Exchange Mutation:**

* Swapping two random genes in the chromosome

**Basic Testing:**

**Configuration 1:**

Initial Solution: Random

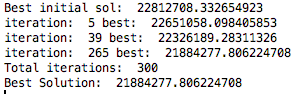
Crossover: Uniform Crossover

Mutation: Reciprocal Exchange

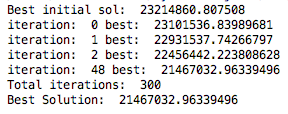
Selection: Random Selection

Result for inst-0.tsp:

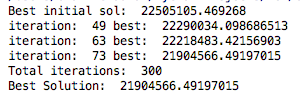
1st Run:



2nd Run:

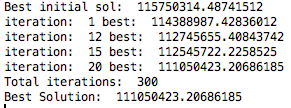


3rd Run:

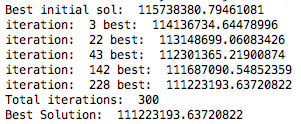


Result for inst-13.tsp:

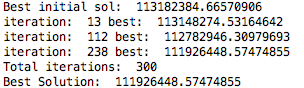
1st Run:



2nd Run:

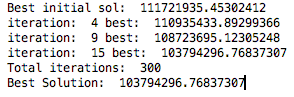


3rd Run:

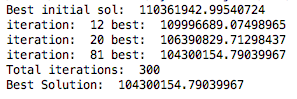


Result for inst-16.tsp:

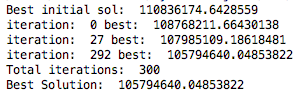
1st Run:



2nd Run:



3rd Run:



**Configuration 2:**

Initial Solution: Random

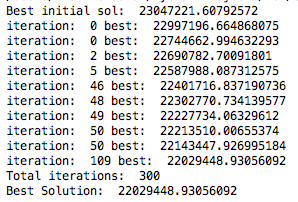
Crossover: Cycle Crossover

Mutation: Scramble Mutation

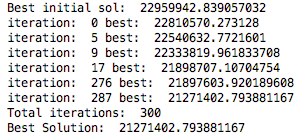
Selection: Random Selection

Result for inst-0.tsp:

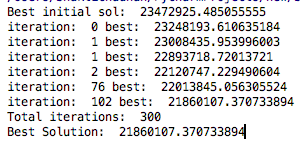
1st Run:



2nd Run:

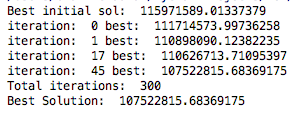


3rd Run:

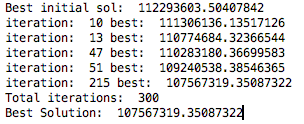


Result for inst-13.tsp:

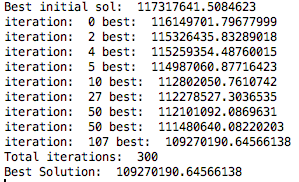
1st Run:



2nd Run:

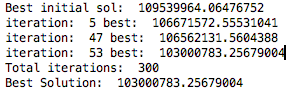


3rd Run:

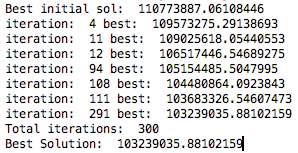


Result for inst-16.tsp:

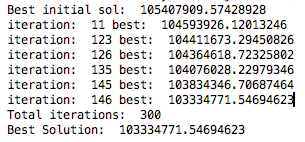
1st Run:



2nd Run:



3rd Run:



**Configuration 3:**

Initial Solution: Random

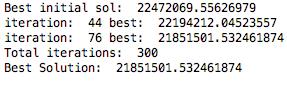
Crossover: Uniform Crossover

Mutation: Reciprocal Exchange

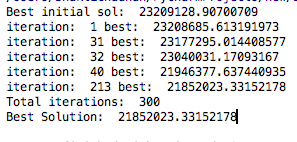
Selection: Roulette Wheel

Result for inst-0.tsp:

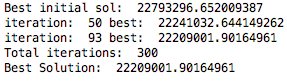
1st Run:



2nd Run:

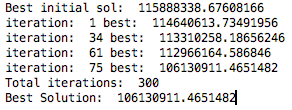


3rd Run:

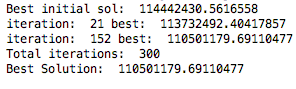


Result for inst-13.tsp:

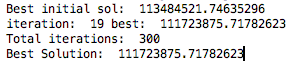
1st Run:



2nd Run:

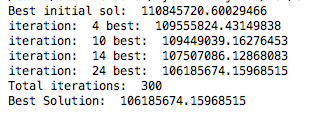


3rd Run:

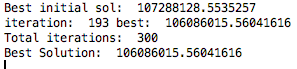


Result for inst-16.tsp:

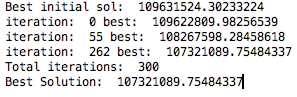
1st Run:



2nd Run:



3rd Run:



**Configuration 4:**

Initial Solution: Random

Crossover: Cycle Crossover

Mutation: Reciprocal Exchange

Selection: Roulette Wheel

Result for inst-0.tsp:

1st Run:



2nd Run:



3rd Run:

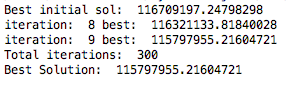


Result for inst-13.tsp:

1st Run:



2nd Run:

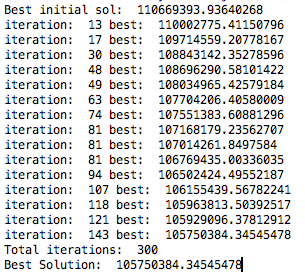


3rd Run:



Result for inst-16.tsp:

1st Run:



2nd Run:



3rd Run:



**Configuration 5:**

Initial Solution: Random

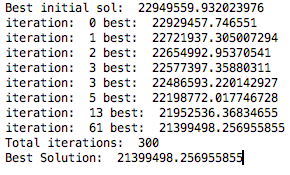
Crossover: Cycle Crossover

Mutation: Scramble Mutation

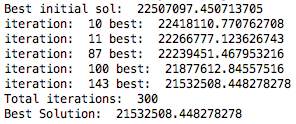
Selection: Roulette Wheel

Result for inst-0.tsp:

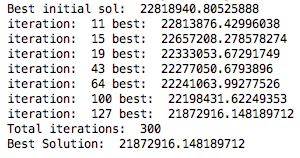
1st Run:



2nd Run:

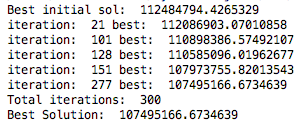


3rd Run:

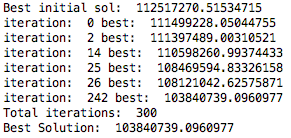


Result for inst-13.tsp:

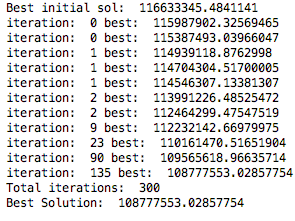
1st Run:



2nd Run:

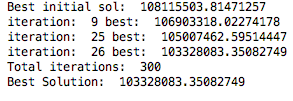


3rd Run:

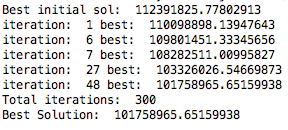


Result for inst-16.tsp:

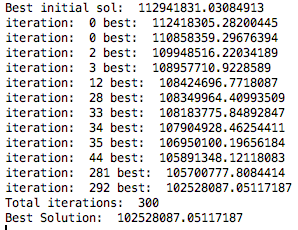
1st Run:



2nd Run:



3rd Run:



**Configuration 6:**

Initial Solution: Random

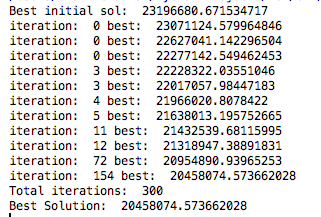
Crossover: Uniform Crossover

Mutation: Scramble Mutation

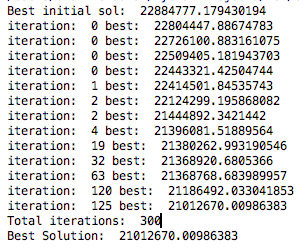
Selection: Best and Second best candidates

Result for inst-0.tsp:

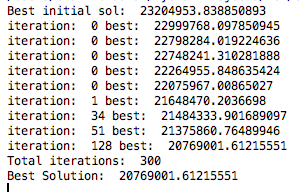
1st Run:



2nd Run:

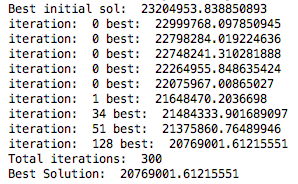


3rd Run:

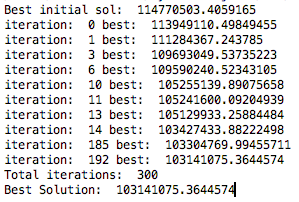


Result for inst-13.tsp:

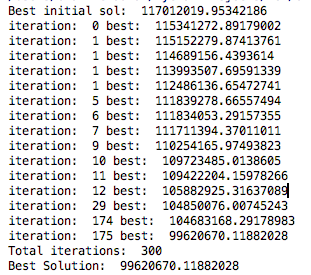
1st Run:



2nd Run:

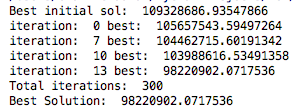


3rd Run:

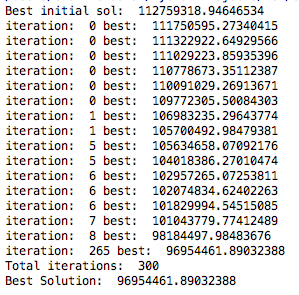


Result for inst-16.tsp:

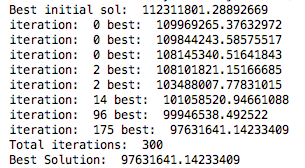
1st Run:



2nd Run:



3rd Run:

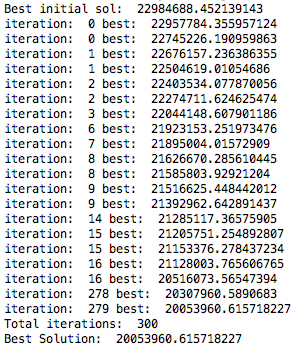


**Advanced Testing:**

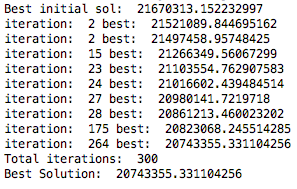
**Configuration 6 with Population Size=150, Mutation rate=0.5**

Result for inst-0.tsp:

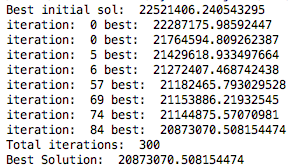
1st Run:



2nd Run:

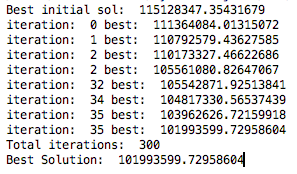


3rd Run:

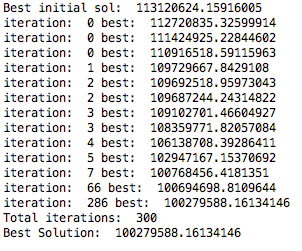


Result for inst-13.tsp:

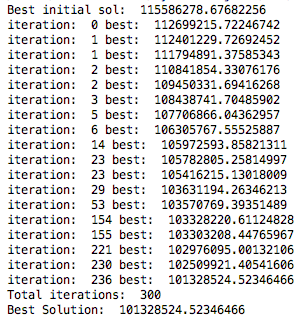
1st Run:



2nd Run:

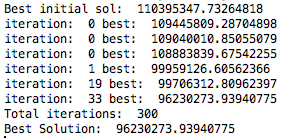


3rd Run:

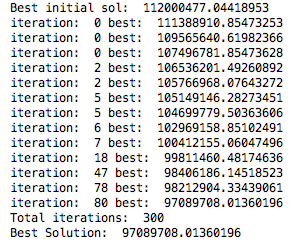


Result for inst-16.tsp:

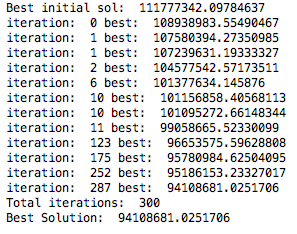
1st Run:



2nd Run:



3rd Run:



**Overall Performance with inst-0.tsp:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Configuration 1** | **Configuration 2** | **Configuration 3** | **Configuration 4** | **Configuration 5** | **Configuration 6** |
| **1st run** | 21884277.806224708 | 22029448.93056092 | 21851501.532461874 | 22970355.71555768 | 21399498.256955855 | 20458074.573662028 |
| **2nd run** | 21467032.96339496 | 21271402.793881167 | 21852023.33152178 | 22992203.940749954 | 21532508.448278278 | 21012670.00986383 |
| **3rd run** | 21904566.49197015 | 21860107.370733894 | 22209001.90164961 | 22360155.06935821 | 21872916.148189712 | 20769001.61215551 |
| **Mean Fitness** | 21751959.09 | 21720319.07 | 21970842.30 | 22774238.24 | 21601640.95 | 20746582.06 |

**Overall Performance with inst-13.tsp:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Configuration 1** | **Configuration 2** | **Configuration 3** | **Configuration 4** | **Configuration 5** | **Configuration 6** |
| **1st run** | 111050423.20686185 | 107522815.68369175 | 106130911.4651482 | 114823783.7832361 | 107495166.6734639 | 20769001.61215551 |
| **2nd run** | 111223193.63720822 | 107567319.35087322 | 110501179.69110477 | 115797955.21604721 | 103840739.0960977 | 103141075.3644574 |
| **3rd run** | 111926448.57474855 | 109270190.64566138 | 111723875.71782623 | 116325309.85577917 | 108777553.02857754 | 99620670.11882028 |
| **Mean Fitness** | 111400021.80 | 108120108.56007545 | 109451988.9580264 | 115649016.28 | 106704486.3 | 74510249.03 |

**Overall Performance with inst-16.tsp:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Configuration 1** | **Configuration 2** | **Configuration 3** | **Configuration 4** | **Configuration 5** | **Configuration 6** |
| **1st run** | 103794296.76837307 | 103000783.25679004 | 106185674.15968515 | 105750384.34545478 | 103328083.35082749 | 98220902.0717536 |
| **2nd run** | 104300154.79039967 | 103239035.88102159 | 106086015.56041616 | 111609893.89907968 | 101758965.65159938 | 96954461.89032388 |
| **3rd run** | 105794640.04853822 | 103334771.54694623 | 107321089.75484337 | 109038665.85721199 | 102528087.05117187 | 97631641.14233409 |
| **Mean Fitness** | 104629697.20 | 103191530.20 | 106530926.50 | 108799648.03 | 102538378.7 | 97602335.03 |

**Conclusion:**

Solved Travelling Salesman problem by implementing Genetic Algorithm to find the optimal solution and Configuration 6 is the fastest since it uses the best and next best candidates, uniform crossover and Scramble mutation.