

# Artificial Intelligence Lab – Meta Heuristics

Submission date :20/4/2021

Shadi Halloun – 313552309

Noor Khamaisi – 207076076

קישור ל GitHub שלנו , Source קוד וקובץ ה PDF מופיעים בצורה מסודרת.

<https://github.com/shadihalloun35/ArtificialIntelligenceLab.git>

## Part A:

- 1- We created a class named problem where the fields are the capacity ,dimension, maximum number of trucks allowed ( Shay told us to choose a number of our own so we assumed the maximum number of trucks is 20), coordinates (the cities) , and edges .

```
class Problem
{
public:
    // methods we could use
    void Initialize(std::string fileName);           // initializing the problem

    // operators
    Problem& operator =(const Problem& prob);
    bool operator ==(const Problem &prob) const;

    // setters
    void setCapacity(int capacity);
    void setDimension(int dimension);
    void setNumOfTrucks(int dimension);
    void setCoordinates(std::vector<vec2> coordinates);
    void setEdges(std::vector<Edge> edges);

    // getters
    int getCapacity();
    int getDimension();
    int getNumOfTrucks();
    std::vector<vec2>& getCoordinates();
    std::vector<Edge>& getEdges();

private:
    int capacity;
    int dimension;
    int numOfTrucks;
    std::vector<vec2> coordinates;
    std::vector<Edge> edges;
};
```

To decode the format of the inputs, we created a special class, a controller named Init where the main function is LoadProblem that is responsible for taking the file name and this pointer (Problem object) and setting the fields to its correct values by reading line after line from the file.

```

7
8
9 class Init
10 {
11 public:
12     static void LoadProblem(Problem &myProblem, std::string fileName);
13     static int FindDimension(std::string line);
14     static int FindCapacity(std::string line);
15     static vec2 FindCoordinates(std::string line);
16     static int FindDemands(std::string line);
17     static void GenerateEdges(Problem & myProblem);
18 };

```

- I couldn't include the implementation of the functions in the report here because the cpp file is about a 100 lines , it will be submitted and it exists on the GitHub.

We created another class names vec2 where it have two main fields : X and Y that leads to the position of current city. We also added another field named index for the print function in the future, and another field named capacity which tell us the capacity of the current city.

```

class vec2 {
public:
    int x;
    int y;
    int demand;
    int index;

    // constructors
    vec2();
    vec2(int x, int y);

    // operators
    vec2 operator+(vec2& v);
    vec2 operator-(vec2& v);
    vec2& operator+=(vec2& v);
    vec2& operator-=(vec2& v);
    vec2& operator=(const vec2& v);
    bool operator ==(const vec2 &v) const;

    // setters
    void setDemand(int demand);
    void setIndex(int index);

    // getters
    int getDemand();
    int getIndex();

    // methods we could use
    float distance(vec2 v) const; // distance between two points
}

```

For design patterns purposes, in the main class we created a global problem object, the user can choose which problem to choose by changing the value of the problem that exists as a define, after choosing the problem we initialize the fields by calling the function described above, then we find the best path by the algorithms described in the assignment .

```
#define problem 1 // choose one of the seven problems
#define Algorithm 1 // choose local search algorithm or genetic algorithm
#define MetaHeuristicAlgorithm 3 // choose one of the local search algorithms
// polluting global namespace, but hey...
using namespace std;

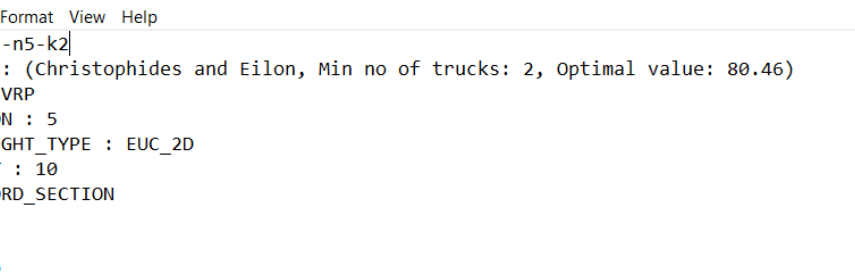
Problem myProblem;

void InitProblem()
{
    switch (problem)
    {
        case 1:
            myProblem.Initialize("problem1.txt");
            break;
        case 2:
            myProblem.Initialize("problem2.txt");
            break;
        case 3:
            myProblem.Initialize("problem3.txt");
            break;
        case 4:
            myProblem.Initialize("problem4.txt");
            break;
        case 5:
            myProblem.Initialize("problem5.txt");
            break;
        case 6:
            myProblem.Initialize("problem6.txt");
            break;
        case 7:
            myProblem.Initialize("problem7.txt");
            break;
        default:
            break;
    }
}
```

Output

```
int main()
{
    using clock = std::chrono::system_clock;
    using sec = std::chrono::duration<double>;
    const auto before = clock::now(); // for elapsed time
    srand(unsigned(time(NULL)));
    clock_t begin = std::clock(); // for clock ticks
    InitProblem();
    FindBestPath();
    clock_t end = std::clock();
    float time_spent = (float)(end - begin) / CLOCKS_PER_SEC;
    std::cout << "Clock Ticks: " << time_spent << "s" << std::endl;
    const sec duration = clock::now() - before;
    std::cout << "Time Elapsed: " << duration.count() << "s" << std::endl;
    getchar();
    return 0;
}
```

2- All the problems are in the GitHub under the name problems.



```
File Edit Format View Help
NAME : E-n5-k2
COMMENT : (Christophides and Eilon, Min no of trucks: 2, Optimal value: 80.46)
TYPE : CVRP
DIMENSION : 5
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 10
NODE_COORD_SECTION
1 0 0
2 0 10
3 -10 10
4 0 -10
5 10 -10
DEMAND_SECTION
1 0
2 3
3 3
4 3
5 3
DEPOT_SECTION
1
-1
EOF
```

And the output we got is :

[illegible]

- 3- Simulated Annealing Algorithm: the heuristic was simply **the minimal cost** . Each state was a solution of its own, we generate a random permutation of N indices where N is the dimension ,then for each truck we calculate its tour till they can no longer visit another city. we would generate a random neighbor , by choosing a 2 random indices of our current permutation and swap them we would have another solution, we calculate it's cost , we move to this state depending on the probability of the simulated annealing algorithm and the temperature by using the cost of the solution.

Tabu Search Algorithm: the heuristic we used is a mix between **KNN and minimal cost** , for each solution we generate neighbors and take the nearest one to the optimal, then adding it to the tabu list.

Ant Colony Optimization Algorithm: the heuristic function we used was the **visibility** where visibility is equal to  $1/(\text{distance between 2 cities})$ . Each state here is a city.

Genetic Algorithm: the heuristic we used was **the cost of the solution** where in the genetic algorithms we used the term fitness and it equals to the cost of the solution.

- 4- The algorithms implementation is in the source files that will be submitted.

**Simulated Annealing Algorithm** implementation exists in SimulatedAnnealing.cpp file.

The pseudo code we used is:

```
• Let  $s = s_0$ 
• For  $k = 0$  through  $k_{\max}$  (exclusive):
  •  $T \leftarrow \text{temperature}((k+1)/k_{\max})$ 
  • Pick a random neighbour,  $s_{\text{new}} \leftarrow \text{neighbour}(s)$ 
  • If  $P(E(s), E(s_{\text{new}}), T) \geq \text{random}(0, 1)$ :
    •  $s \leftarrow s_{\text{new}}$ 
• Output: the final state  $s$ 
```

**Tabu search Algorithm** implementation exists in TabuSearch.cpp file.  
The pseudo code we used is:

```
1 sBest ← s0
2 bestCandidate ← s0
3 tabuList ← []
4 tabuList.push(s0)
5 while (not stoppingCondition())
6     sNeighborhood ← getNeighbors(bestCandidate)
7     bestCandidate ← sNeighborhood[0]
8     for (sCandidate in sNeighborhood)
9         if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
10             bestCandidate ← sCandidate
11     end
12 end
13 if (fitness(bestCandidate) > fitness(sBest))
14     sBest ← bestCandidate
15 end
16 tabuList.push(bestCandidate)
17 if (tabuList.size > maxTabuSize)
18     tabuList.removeFirst()
19 end
20 end
21 return sBest
```

**Ant Colony Optimization Algorithm** implementation exists in  
AntColonyOptimization.cpp file.  
The pseudo code we used is described in the lecture's slides.

**Genetic Algorithm** implementation exists in GeneticAlgorithm.cpp file.  
We changed the ga\_struct that it contains a permutation and an object  
of solution and we removed the string field, the crossovers we used is  
PMX and OX , the mutation we used is the exchange(swap) mutation and  
insertion mutation. We implemented all the select methods we learnt.

```
struct ga_struct
{
    std::vector<vec2> permutation;    // the permutation of the whole tour
    Soultion soulution;             // the trucks tour
    float fitness;                   // its fitness;
    unsigned int age;
};
```

### 5- The Minimal Cost Heuristic used in Simulated Annealing Algorithm:

In this way we can estimate the closest solution to the optimal every time we are moving forward another state. In the CVRP problem we want the lowest cost so this heuristic help us to get closer to our optimal solution.

### KNN and minimal cost Heuristic used in Tabu Search Algorithm:

Using this heuristic helped us to take from the many states we produce, the best state possible we could find and that is the nearest one to the optimal, this helped us to get to the optimal in the most efficient way.

### Visibility Heuristic used in Ant Colony Optimization Algorithm:

Using this heuristic helps us to choose the best possible edge that can get us to the optimal solution, the higher the distance between two cities give us lower visibility and then the probability to choose this edge will be low.

The cost of the solution Heuristic used in Genetic Algorithm:

This heuristic helps us to choose the best possible gene of the current generation we produce that has the best fitness .

6- The output using the simulated annealing algorithm :

A screenshot of a Windows command prompt window. The title bar at the top shows the file path "C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe". The main area of the window has a black background with white text. It displays several lines of numbers: "80.645 0", "0 1 2 3 0", "0 4 0", followed by multiple pairs of "0 0". At the bottom left, it shows "Clock Ticks: 6.47s" and "Time Elapsed: 6.4709s". The standard Windows window controls (minimize, maximize, close) are visible in the top right corner.



The output using the tabu search algorithm :

A screenshot of a Windows command prompt window titled "C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the following output:  

```
80.645 0  
0 1 2 3 0  
0 4 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 4.142s  
Time Elapsed: 4.14268s
```

  
The output consists of several lines of space-separated integers, followed by two lines of timing information. A small white cursor is visible at the bottom left of the terminal area.

The output using the Ant Colony Optimization algorithm :

The screenshot shows a Windows command prompt window titled "C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe". The window has standard minimize, maximize, and close buttons in the top right corner. On the left side, there is a vertical scrollbar. The output displayed in white text on a black background includes:  

```
80.645 0  
0 1 2 3 0  
0 4 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 0.016s  
Time Elapsed: 0.0158516s
```

  
A single underscore character "\_" appears on a new line at the bottom of the visible output.

The output using the genetic algorithm :

A screenshot of a Windows command prompt window titled "C:\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the following output:  

```
80.645 0  
0 1 2 3 0  
0 4 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 3.525s  
Time Elapsed: 3.52786s
```

The output consists of several lines of space-separated integers. The first line is "80.645 0". The second line is "0 1 2 3 0". The third line is "0 4 0". This is followed by 18 more lines, each containing two zeros ("0 0"). At the bottom, there are two lines of floating-point numbers: "Clock Ticks: 3.525s" and "Time Elapsed: 3.52786s".

### Part B:

7- Problem Name : **E-n22-k4**

### Simulated Annealing Algorithm:

```
C:\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
<375.28 0
"0 16 19 21 14 0
"0 9 7 5 2 1 6 0
"0 10 8 3 4 11 13 0
"0 12 15 18 20 17 0
"0 0
"0 0
<0 0
0 0
0 0
r0 0
l0 0
e0 0
e0 0
e0 0
y0 0
e0 0
0 0
p0 0
l0 0
Clock Ticks: 48.66s
Time Elapsed: 48.6601s
```



### Tabu Search algorithm:

C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\Release\CVRP.exe

```

A375.28 0
0 9 7 5 2 1 6 0
t0 13 11 4 3 8 10 0
0 17 20 18 15 12 0
0 14 21 19 16 0
i0 0
i0 0
S0 0
S0 0
a0 0
V0 0
V0 0
V0 0
t0 0
l0 0
0 0
0 0
t0 0
0 0
0 0
i0 0
Clock Ticks: 9.663s
Time Elapsed: 9.66403s

```

The screenshot shows a black terminal window titled "C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\Release\CVRP.exe". The output consists of several rows of numbers:  

```
375.28 0  
0 6 1 2 5 7 9 0  
0 13 11 4 3 8 10 0  
0 17 20 18 15 12 0  
0 14 21 19 16 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0
```

  
At the bottom left, it displays performance metrics:  

```
Clock Ticks: 7.798s  
Time Elapsed: 7.79842s
```





Problem Name : **E-n33-k4**  
Simulated Annealing Algorithm:

C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe

```
856.97 0  
0 3 2 12 32 10 9 8 7 6 5 4 0  
0 1 18 19 21 20 22 23 24 25 17 15 13 0  
0 26 27 16 28 29 0  
0 30 31 14 11 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
  
Clock Ticks: 30.616s  
Time Elapsed: 30.6183s  
-
```

Tabu search algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

849.078 0  
0 26 27 16 28 29 0  
0 4 5 7 6 8 9 32 11 12 2 0  
0 13 17 25 24 23 22 20 21 19 18 10 3 0  
0 30 15 14 1 31 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 10.511s  
Time Elapsed: 10.5117s

ACO algorithm:

[illegible]

## Genetic algorithm:

[illegible]

Problem Name : **E-n51-k5**

### Simulated Annealing Algorithm:







Problem Name : **E-n76-k10**

### Simulated Annealing Algorithm:

[illegible]

Tabu search algorithm:

[illegible]

ACO algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
993.203 0  
0 27 52 34 46 67 75 0  
0 68 6 51 17 33 73 1 62 0  
0 28 74 48 47 21 61 22 0  
0 64 42 41 43 23 56 16 63 0  
0 3 44 32 40 12 58 72 0  
0 39 9 25 55 18 50 24 49 0  
0 2 30 45 29 54 19 8 35 0  
0 7 53 14 59 66 65 0  
0 10 38 11 31 26 0  
0 13 57 15 20 37 5 36 69 71 60 70 0  
0 4 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
  
Clock Ticks: 38.304s  
Time Elapsed: 38.3045s
```

### Genetic algorithm :

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1321.91 0  
0 1 28 30 48 5 6 0  
0 7 8 53 38 11 58 0  
0 52 20 15 14 59 54 19 0  
0 16 21 22 23 24 18 26 0  
0 27 29 17 12 32 40 0  
0 4 13 57 36 37 45 34 0  
0 56 41 42 43 64 2 0  
0 46 73 33 49 50 44 3 0  
0 9 25 55 31 39 72 10 35 0  
0 61 62 63 51 65 66 67 0  
0 74 69 71 70 60 47 68 75 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 71.578s  
Time Elapsed: 71.5793s  
  
-
```

Problem Name : **E-n76-k8**

### Simulated Annealing Algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
806.926 0  
0 68 4 45 29 48 21 61 28 0  
0 26 12 72 31 65 66 59 14 0  
0 6 2 62 73 1 43 41 42 64 22 0  
0 67 58 10 38 11 53 7 0  
0 33 63 23 56 18 50 55 25 9 39 0  
0 40 32 44 3 24 49 16 51 17 0  
0 34 46 8 35 19 54 13 57 15 27 52 0  
0 5 37 20 70 60 71 69 36 47 74 30 75 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
  
Clock Ticks: 50.532s  
Time Elapsed: 50.5326s
```

Tabu search algorithm:



## Genetic algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1269.24 0  
0 1 2 33 28 5 45 36 47 0  
0 38 10 11 53 13 57 15 27 0  
0 17 48 37 20 21 22 23 24 56 0  
0 68 30 4 7 66 65 32 44 3 0  
0 75 58 72 9 39 64 41 42 43 73 0  
0 67 46 8 26 40 50 51 62 0  
0 12 18 55 25 31 14 59 19 34 0  
0 16 63 49 54 35 52 29 60 70 71 69 61 74 6 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 90.115s  
Time Elapsed: 90.1157s
```

Problem Name : **E-n101-k8**

### Simulated Annealing Algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1020.13 0  
0 68 77 79 78 81 9 20 30 32 90 63 64 0  
0 53 13 94 18 83 45 8 82 48 7 31 0  
0 50 51 33 3 29 34 35 71 65 66 10 62 70 1 69 0  
0 89 60 5 84 61 16 17 46 36 49 47 19 11 88 52 0  
0 40 57 15 43 38 86 44 41 22 74 72 73 21 26 0  
0 12 54 55 4 39 56 75 23 67 25 24 80 76 0  
0 58 2 87 42 14 91 100 37 59 92 97 95 6 27 28 0  
0 96 99 93 85 98 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 75.377s  
Time Elapsed: 75.3775s
```

### Tabu search algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1170.5 0  
0 96 92 13 85 91 61 82 8 7 48 31 69 0  
0 28 80 68 3 81 9 35 65 71 51 33 50 0  
0 77 79 78 34 29 24 25 55 54 76 70 32 90 63 30 1 0  
0 94 14 38 86 16 98 37 93 99 84 6 27 0  
0 89 18 5 17 45 46 47 36 19 49 64 11 88 52 0  
0 12 4 41 56 75 22 74 23 39 67 26 0  
0 53 58 40 73 72 2 87 97 42 57 44 43 15 21 0  
0 66 20 10 62 83 60 59 95 100 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
  
Clock Ticks: 31.762s  
Time Elapsed: 31.7626s
```

ACO algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1044.36 0  
0 7 88 31 10 62 11 63 90 32 30 70 69 1 50 29 24 0  
0 12 80 68 76 77 3 79 81 33 51 20 0  
0 66 71 35 65 9 34 78 26 28 27 52 89 6 0  
0 94 96 99 93 98 37 92 59 95 97 87 57 2 0  
0 13 58 53 40 21 73 74 72 75 22 41 15 43 42 100 91 0  
0 85 61 16 86 44 14 38 17 84 5 0  
0 60 83 8 45 46 48 47 36 49 64 19 18 82 0  
0 54 55 25 39 4 56 23 67 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
  
Clock Ticks: 162.079s  
Time Elapsed: 162.08s  
-
```



## Genetic algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1513.97 0  
0 26 2 72 4 39 6 18 8 48 90 63 52 0  
0 13 14 38 16 17 7 19 69 21 22 23 67 25 0  
0 1 27 10 70 30 31 50 33 34 35 71 81 40 0  
0 95 15 41 42 43 44 45 46 47 36 49 32 51 12 0  
0 53 54 55 56 57 58 59 60 88 62 11 64 65 66 20 0  
0 68 24 29 9 3 73 74 75 76 77 78 79 80 28 0  
0 82 83 84 85 86 93 61 100 37 91 92 87 0  
0 94 5 96 97 98 99 89 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
Clock Ticks: 114.874s  
Time Elapsed: 114.876s
```

```
std::vector<vec2> savedCoordinates = myProblem.getCoordinates();
```

Problem Name : **E-n101-k14**

### Simulated Annealing Algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
```

```
1279.85 0  
0 52 88 82 83 93 59 96 0  
0 50 78 34 29 67 23 56 21 0  
0 99 91 44 16 61 84 5 60 0  
0 27 33 81 79 3 12 0  
0 69 51 9 35 71 65 66 20 0  
0 26 55 25 39 57 2 40 0  
0 48 7 47 36 19 10 70 0  
0 74 75 22 41 15 43 42 87 0  
0 62 11 49 64 63 90 32 0  
0 76 77 80 24 54 4 72 73 0  
0 31 30 1 68 28 0  
0 18 8 46 45 17 86 38 14 0  
0 53 58 13 94 6 89 0  
0 85 100 98 37 97 95 92 0  
0 0  
0 0  
0 0  
0 0  
0 0  
0 0  
  
Clock Ticks: 90.126s  
Time Elapsed: 90.1264s
```

Tabu search algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
1478.71 0
0 36 47 49 45 82 19 0
0 69 51 9 78 79 34 81 3 0
0 59 37 85 91 97 0
0 13 95 92 14 43 98 100 0
0 48 46 64 63 11 32 10 0
0 99 16 61 84 17 83 8 7 31 0
0 86 38 44 42 57 53 0
0 58 72 2 87 15 93 0
0 28 94 6 60 5 89 18 52 0
0 12 76 54 80 68 77 0
0 27 33 71 35 65 66 20 70 0
0 23 56 67 39 25 55 24 29 0
0 40 73 75 22 41 4 74 21 0
0 26 50 88 62 90 30 1 96 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0

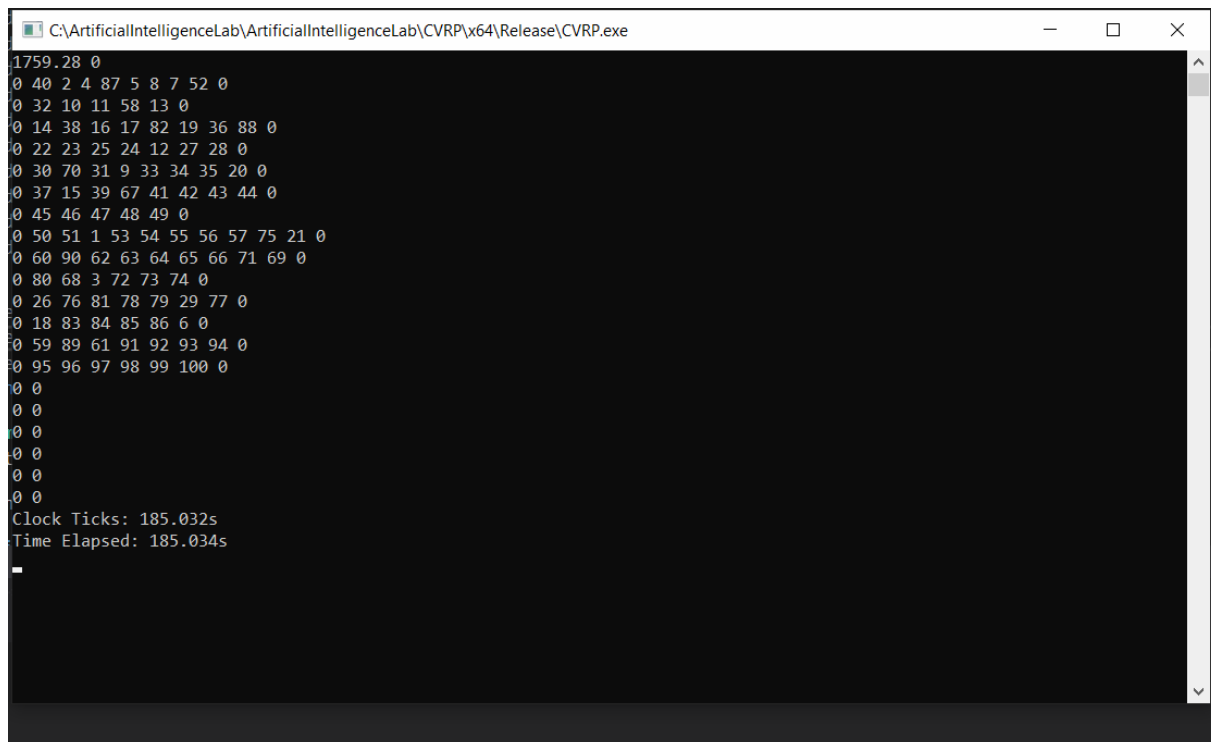
Clock Ticks: 78.233s
Time Elapsed: 78.233s
```

ACO algorithm:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
1322.59 0
0 69 27 28 26 40 58 53 0
0 95 97 92 98 37 100 91 85 0
0 61 59 99 96 94 6 89 0
0 88 31 10 62 11 19 0
0 47 36 49 64 63 90 32 0
0 70 30 20 66 71 35 9 51 0
0 81 33 79 3 77 76 0
0 68 80 12 54 55 25 4 0
0 72 74 22 41 75 56 23 0
0 39 67 24 29 78 34 65 0
0 1 50 52 18 83 60 5 84 17 45 0
0 8 82 7 48 46 86 0
0 16 44 14 38 43 15 57 2 73 0
0 21 13 87 42 93 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0

Clock Ticks: 314.961s
Time Elapsed: 314.961s
```

## Genetic algorithm :



```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\CVRP\x64\Release\CVRP.exe
1759.28 0
0 40 2 4 87 5 8 7 52 0
0 32 10 11 58 13 0
0 14 38 16 17 82 19 36 88 0
0 22 23 25 24 12 27 28 0
0 30 70 31 9 33 34 35 20 0
0 37 15 39 67 41 42 43 44 0
0 45 46 47 48 49 0
0 50 51 1 53 54 55 56 57 75 21 0
0 60 90 62 63 64 65 66 71 69 0
0 80 68 3 72 73 74 0
0 26 76 81 78 79 29 77 0
0 18 83 84 85 86 6 0
0 59 89 61 91 92 93 94 0
0 95 96 97 98 99 100 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
Clock Ticks: 185.032s
Time Elapsed: 185.034s
-
```

**הערה:** עבור כל אחד מהאלגוריתם ועבור כל אחת מהבעיות , אנחנו נתנו להם לרוץ מספיק זמן עד שיימצאו את הפתרון האופטימלי ולא רצינו לתת להם מעט זמן ממה שהם צריכים כדי שהבעיה תהפוך ליותר מעניינת ונצליח להגיע כמה שקרוב לפתרון האופטימלי

Problem Name	Optimal Value	Algorithm Name	Did we find solution in time (Yes/No)	Solution Cost	How far the solution from the optimal	CPU Time (seconds)	Elapsed Time (seconds)
E-n5-k2	80.46	Simulated Annealing	Yes	80.46	0	6.47	6.4709
E-n5-k2	80.46	Tabu Search	Yes	80.46	0	4.142	4.14268
E-n5-k2	80.46	ACO	Yes	80.46	0	0.016	0.0158516
E-n5-k2	80.46	Genetic Algorithm	Yes	80.46	0	3.525	3.52786
E-n22-k4	375	Simulated Annealing	Yes	375.28	0.28	15.862	15.8625
E-n22-k4	375	Tabu Search	Yes	375.28	0.28	9.663	9.66403
E-n22-k4	375	ACO	Yes	378.923	3.643	0.31	0.310838
E-n22-k4	375	Genetic Algorithm	Yes	383.517	8.237	26.478	26.4793
E-n33-k4	835	Simulated Annealing	Yes	856.97	21.97	30.616	30.6183
E-n33-k4	835	Tabu Search	Yes	849.078	14.078	10.511	10.5117
E-n33-k4	835	ACO	Yes	876.962	41.962	0.738	0.738702
E-n33-k4	835	Genetic Algorithm	Yes	889.318	54.318	31.387	31.3877
E-n51-k5	521	Simulated Annealing	Yes	599.851	78.851	64.356	64.3572
E-n51-k5	521	Tabu Search	Yes	606.768	85.768	26.506	26.506
E-n51-k5	521	ACO	Yes	637.239	116.239	2.308	2.30835
E-n51-k5	521	Genetic Algorithm	Yes	727.824	206.824	77.401	77.4029
E-n76-k10	832	Simulated Annealing	Yes	951.999	119.999	128.434	128.435
E-n76-k10	832	Tabu Search	Yes	1056.72	224.72	55.886	55.8871
E-n76-k10	832	ACO	Yes	993.203	161.203	38.304	38.3045

E-n76-k10	832	Genetic Algorithm	Yes	1321.91	489.91	71.578	71.5793
E-n76-k8	735	Simulated Annealing	Yes	806.926	71.926	50.532	50.5326
E-n76-k8	735	Tabu Search	Yes	936.51	201.51	20.311	20.3111
E-n76-k8	735	ACO	Yes	890.821	155.821	40.075	40.0749
E-n76-k8	735	Genetic Algorithm	Yes	1269.24	534.24	90.115	90.1157
E-n101-k8	817	Simulated Annealing	Yes	1020.13	203.13	75.377	75.3775
E-n101-k8	817	Tabu Search	Yes	1170.5	353.5	31.762	31.7626
E-n101-k8	817	ACO	Yes	1044.36	227.36	162.079	162.08
E-n101-k8	817	Genetic Algorithm	Yes	1513.97	696.97	114.874	114.876
E-n101-k14	1077	Simulated Annealing	Yes	1279.85	202.85	90.126	90.1264
E-n101-k14	1077	Tabu Search	Yes	1478.71	401.71	78.233	87.2339
E-n101-k14	1077	ACO	Yes	1322.59	245.59	314.961	314.961
E-n101-k14	1077	Genetic Algorithm	Yes	1759.28	682.28	185.032	185.034

9-

Algorithm	Space Complexity
Simulated Annealing	$O(\text{DIMENSION})$
Tabu Search	$O(\text{DIMENSION} * \text{TABU SIZE})$
ACO	$O(\text{DIMENSION}^2 * (\#ants))$
Genetic Algorithm	$O(\text{POPSIZE} * \text{DIMENSION}) + O(\#iterations)$

כאשר ה- DIMENSION הוא מספר הערים שקיבלנו כקלט.

הסבר:

#### SIMULATED ANNEALING:

השתמשנו בווקטור של מערכים כאשר גודל הווקטור הוא לכל היותר 20 שהוא מספר המשאיות לכן  $O(1)$  וגודל המערך עבור כל אחד מהמערכים בווקטור זה (ששמרנו בו את הערים שביקרנו עבור כל משאית) הכי גדול שיכול להיות הוא כגודל ה DIMENSION.

#### TABU SEARCH:

השתמשנו בווקטור של מערכים מספר קבוע של פעמים, שמרנו גם ווקטור של ווקטור של מערכים שבו איחסנו את השכנים של הפתרון, במקרה שלנו עשינו 30 שכן לכן עדיין הסיבכיות היא  $O(DIMENSION)$  עד כה, אחרי זה יצרנו מערך (TABU) בגודל של TABU SIZE והכנסנו לו כל פעם מצב שהוא בגודל  $O(DIMENSION)$  לכן בס"ה יהיה לנו סיבכיות מקום של  $O(DIMENSION * TABU SIZE)$

#### ACO:

השתמשנו במערך בגודל מספר הקשתות שבגרף שלנו כדי לעדכן את ה PHERMONE עבור הקשתות, גודל כל איבר הוא  $O(1)$  לכן  $O(DIMENSION^2)$ , השתמשנו גם במערך מסוג ANT מגודל מספר ה ANTS שרוצים, כל ANT יש לה מערך TABU בגודל  $O(DIMENSION)$  כדי לדעת איפה היא ביקרה, יש לה PHERMONE שהוא FLOAT מגודל  $O(1)$ , יש לא מערך של EDGES כדי לדעת באיזה קשתות היא ביקרה לכן  $O(DIMENSION^2)$  וגם יש לה OBJECT SOLUTION שהוא מגודל  $O(DIMENSION)$  כמו שהוסבר ב SIMULATED ANNEALING לכן גודל כל ANT הוא  $O(DIMENSION) + O(1) + O(DIMENSION^2) + O(DIMENSION)$  =  $O(DIMENSION^2)$  לכן בס"ה יהיה לנו סיבכיות מקום :

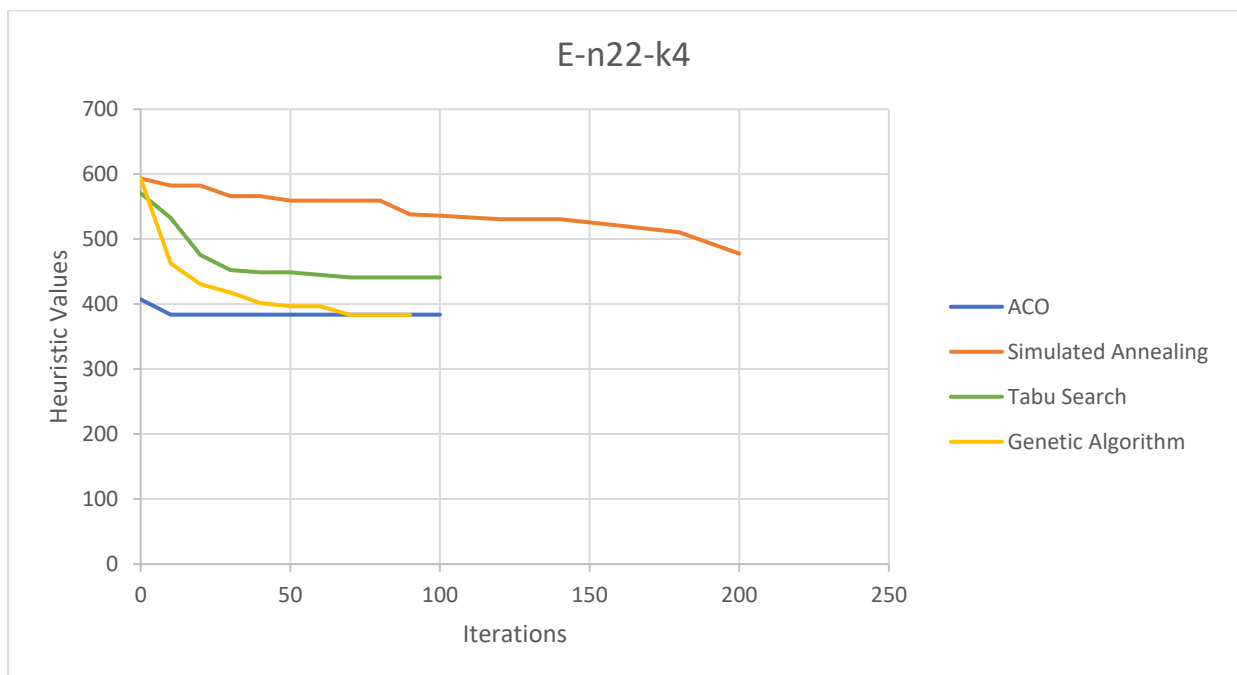
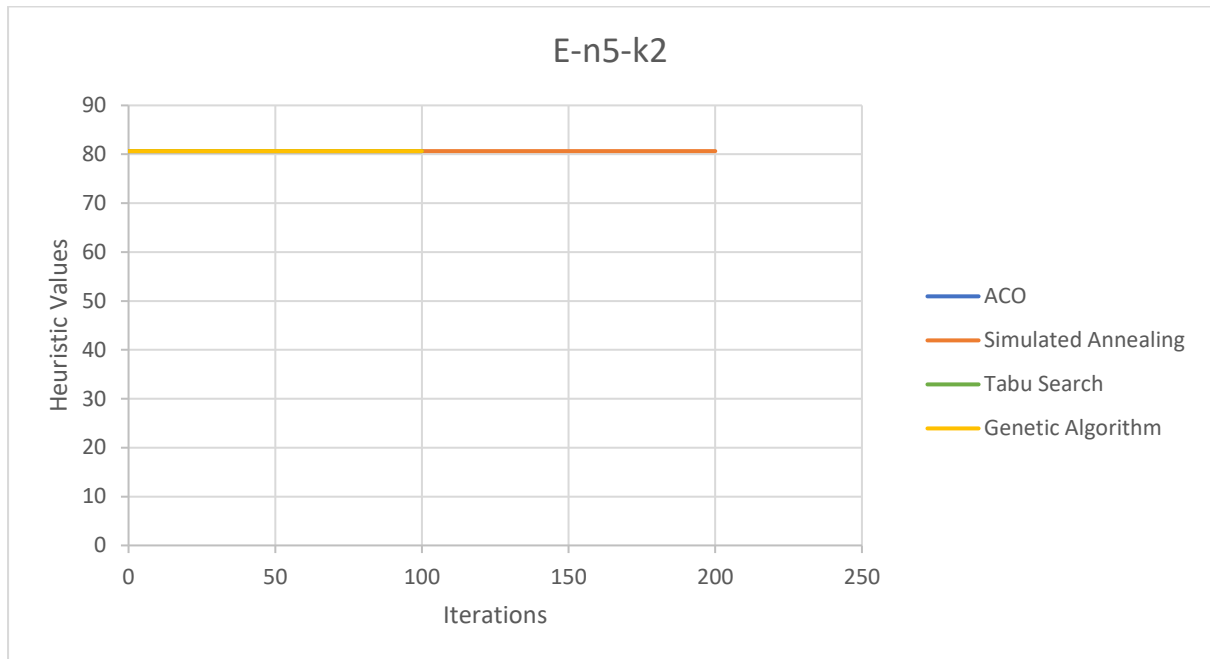
$$O(DIMENSION^2) * \#ANTS + O(DIMENSION^2) = O(DIMENSION^2) * \#ANTS$$

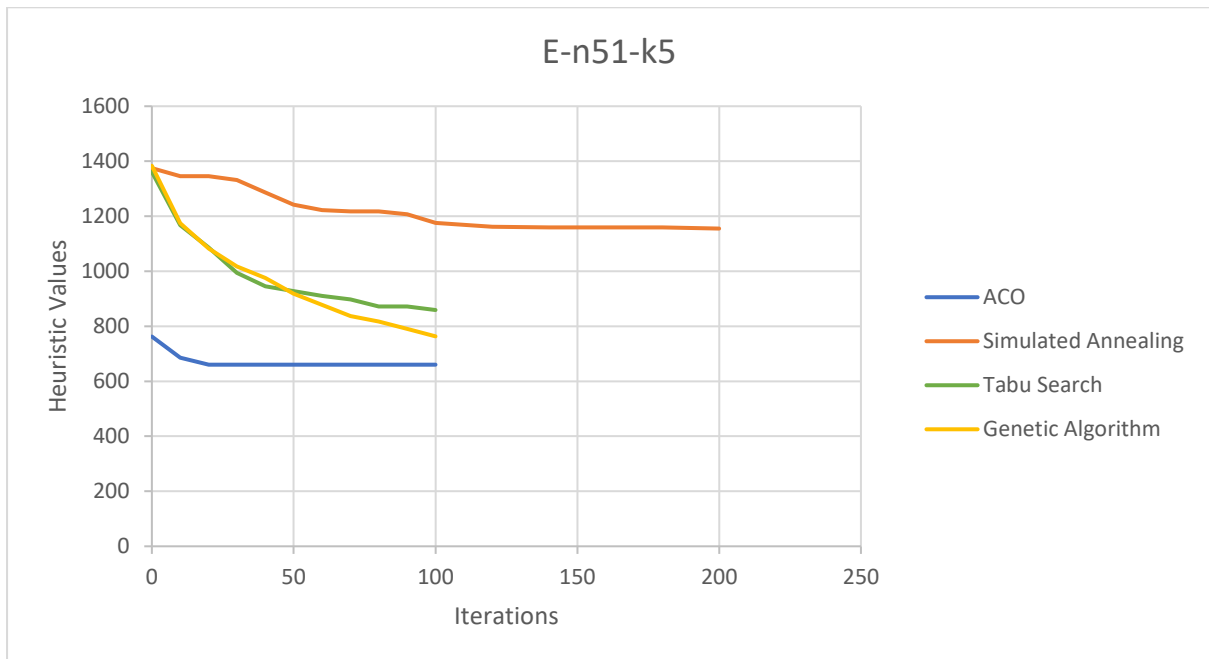
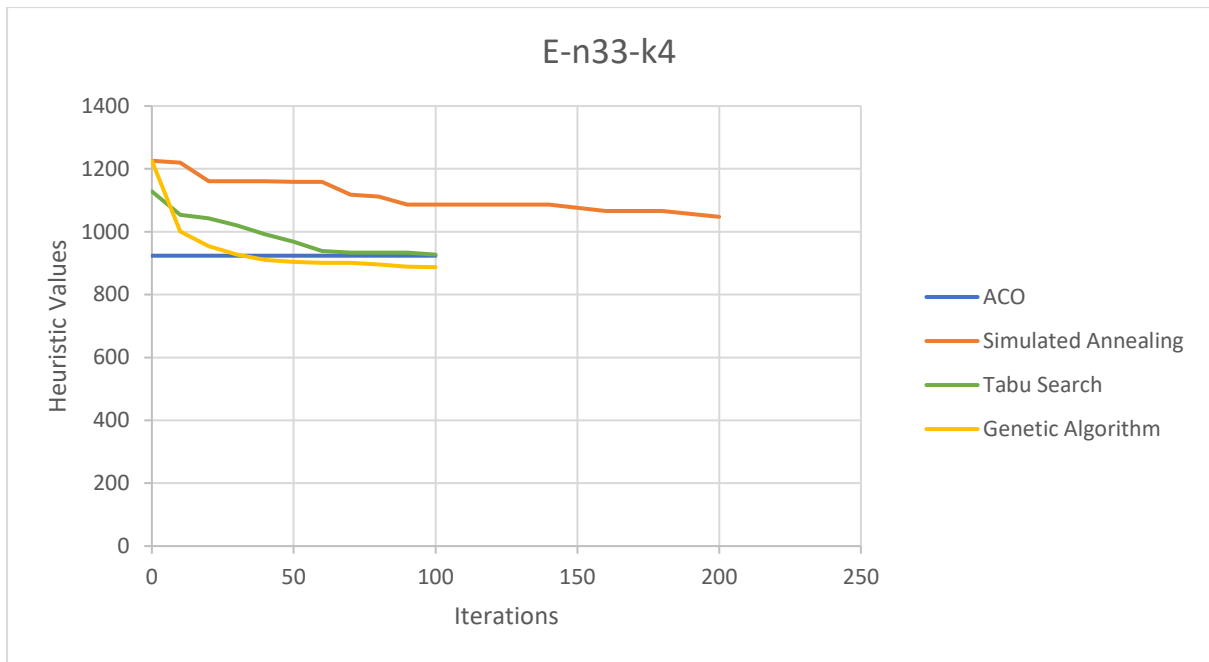
#### Genetic Algorithm:

כל CITEZEN יש לו ווקטור של מערכים לכן  $O(DIMENSION)$  וגם מערך עזר (בשבילי) של CITIES לכן  $O(DIMENSION)$  וגם משתנה AGE ו משתנה FITNESS בגודל  $O(1)$ , גם יש לי שני וקטורים מסוג FLOAT ששומר בו את ה AVERAGES וה DEVIATION בכל GENERATION כלומר כגודל האיטרציות לכן בס"ה יהיה לנו סיבכיות מקום :  $O(POPSIZE * DIMENSION) + O(\#iterations)$

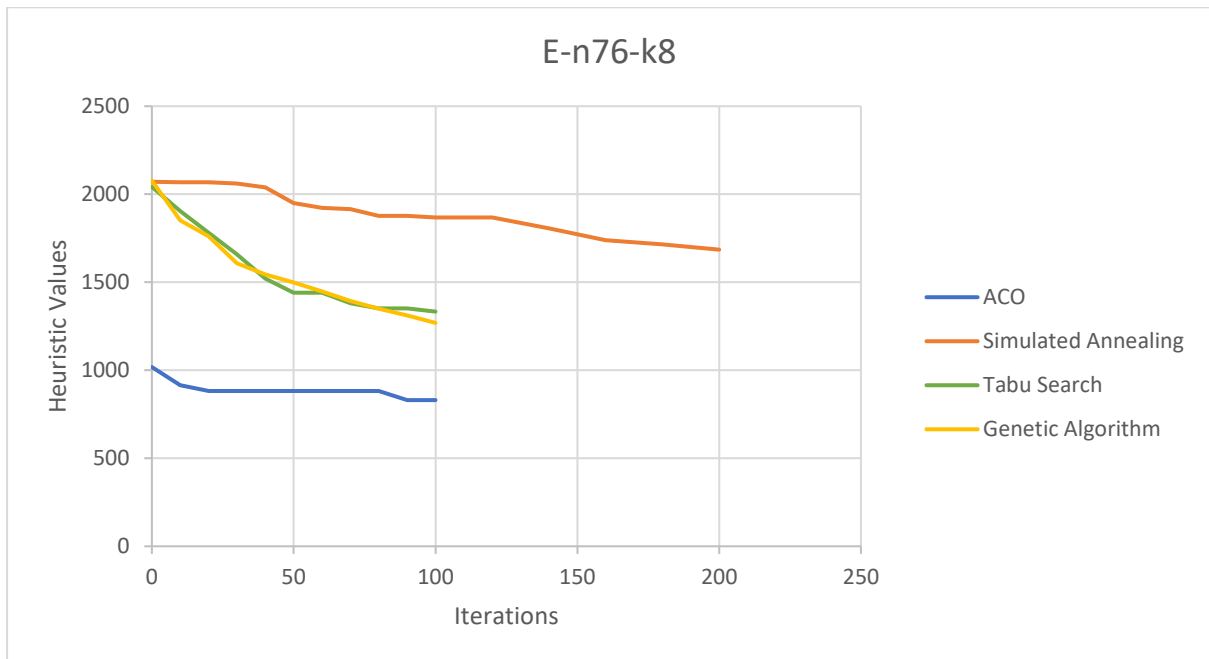
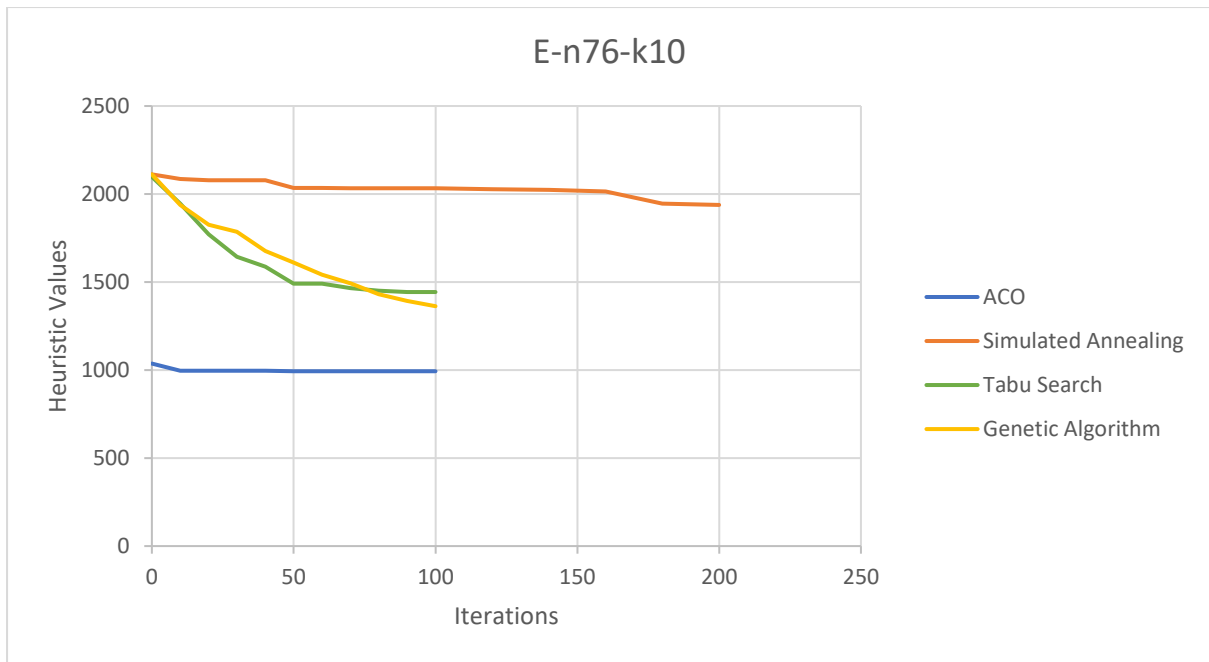
10-

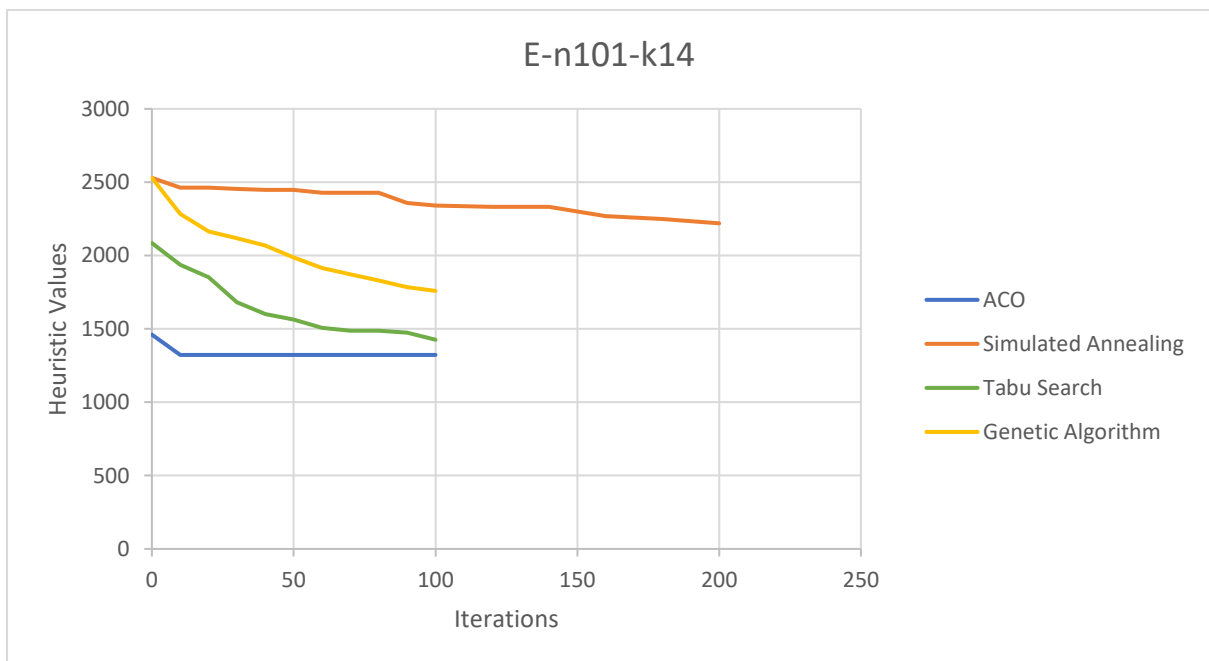
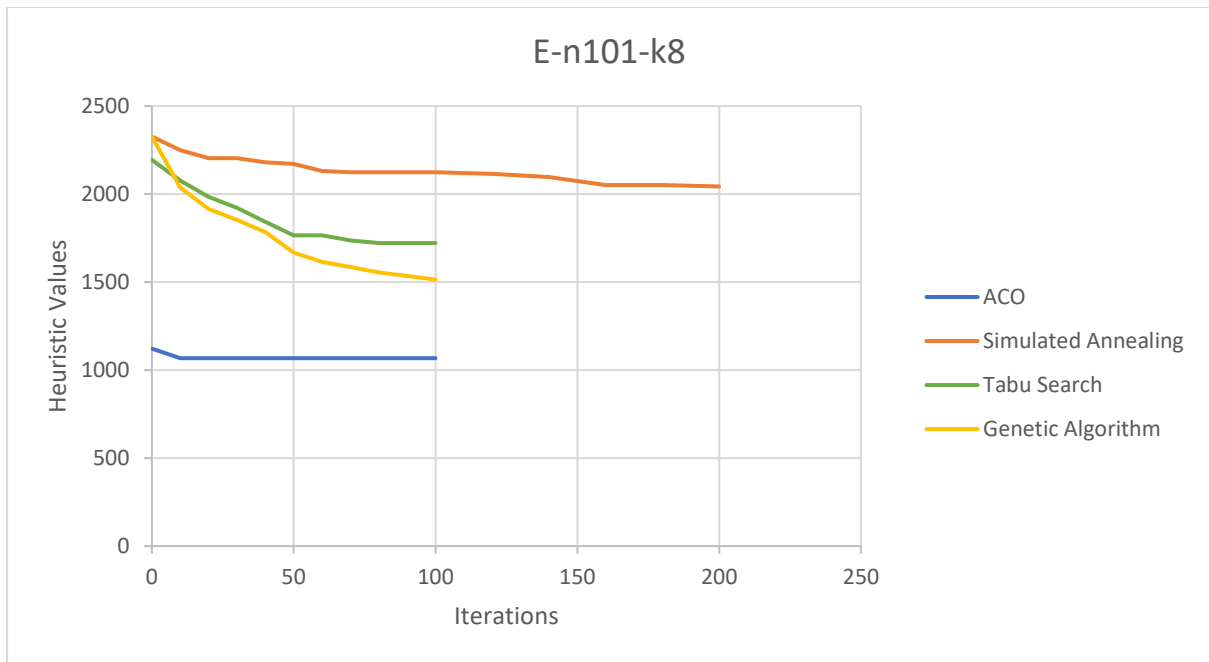
10.1 :



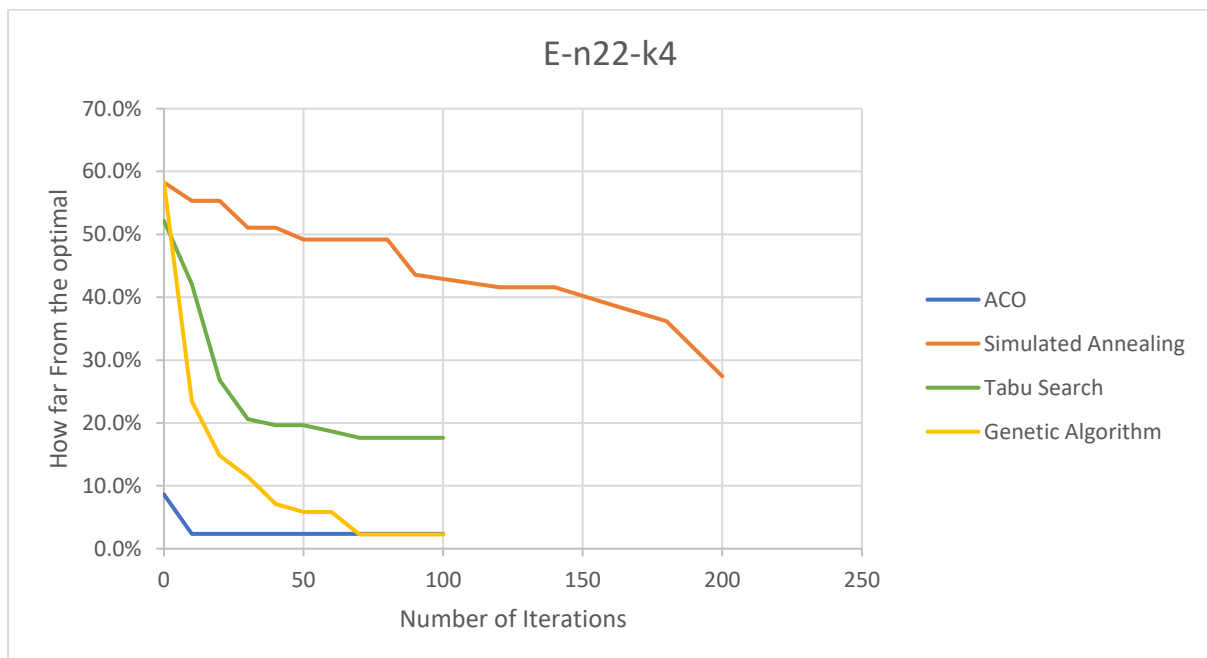
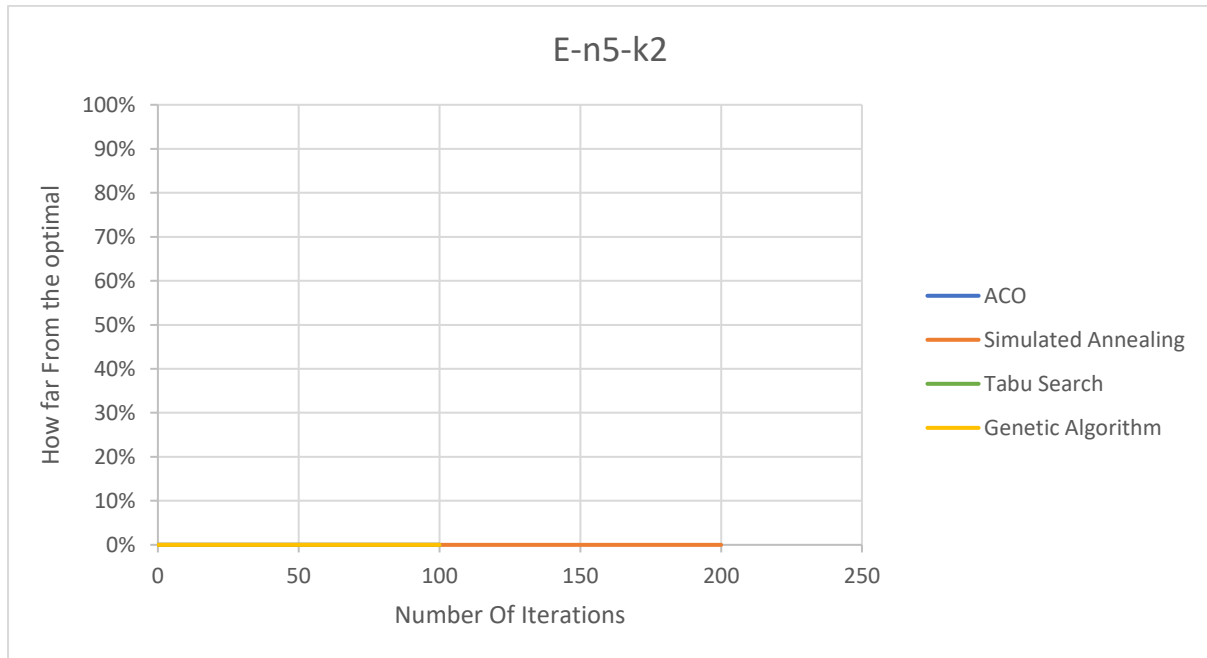


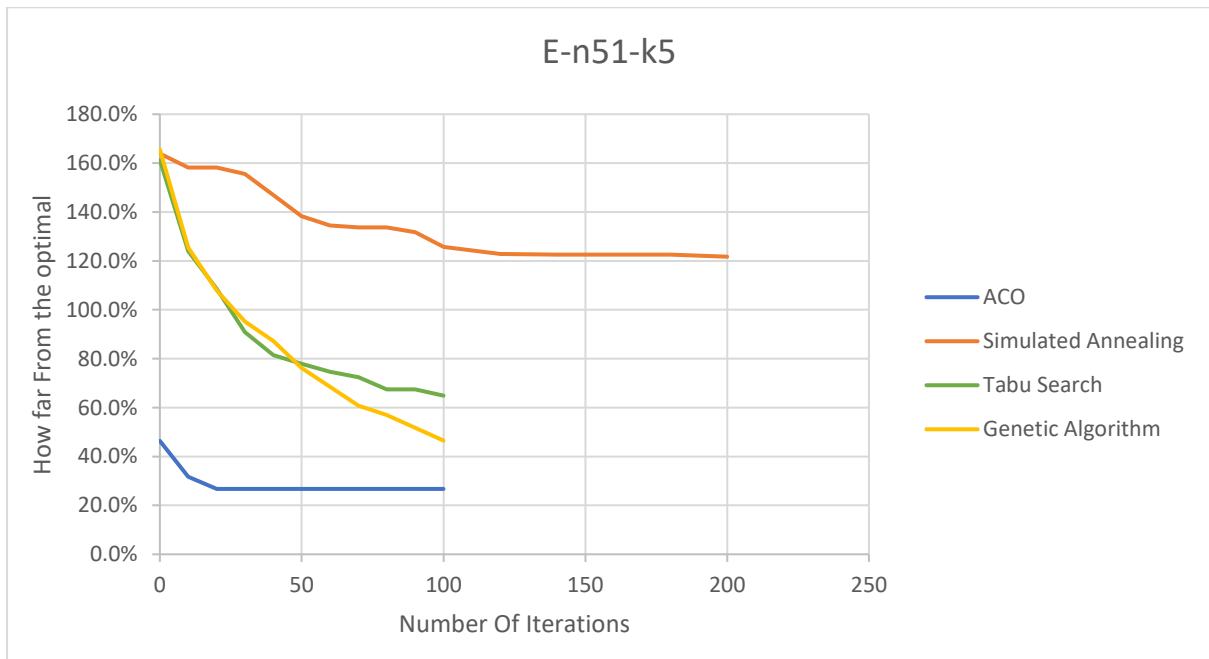
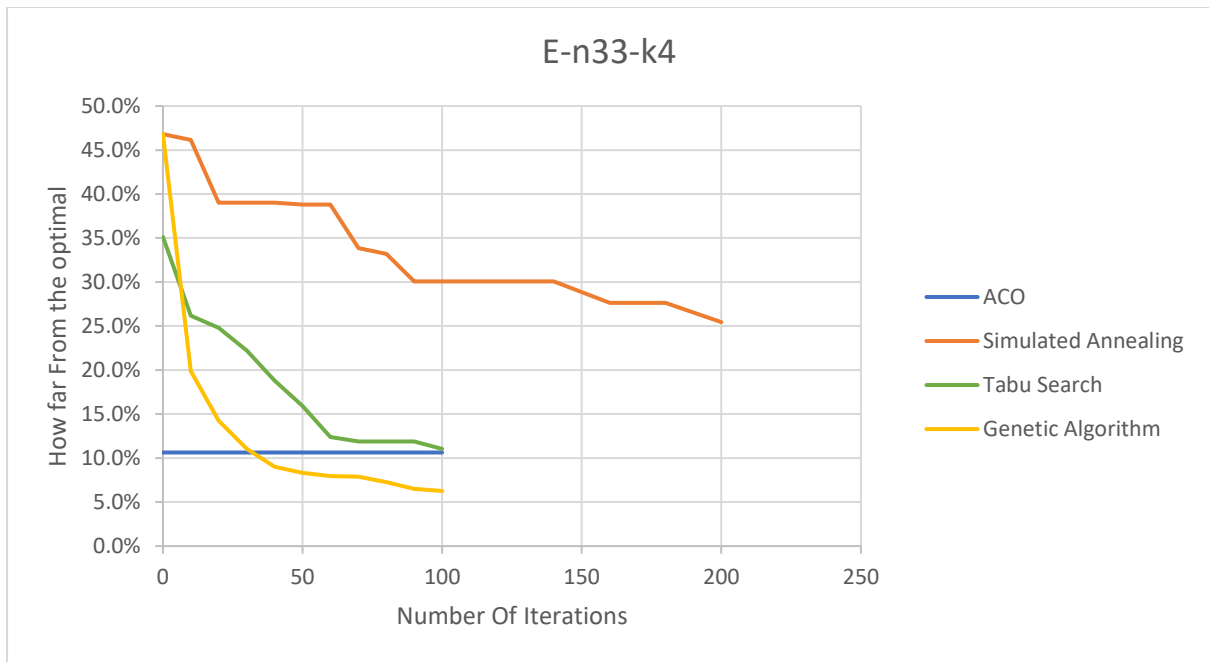


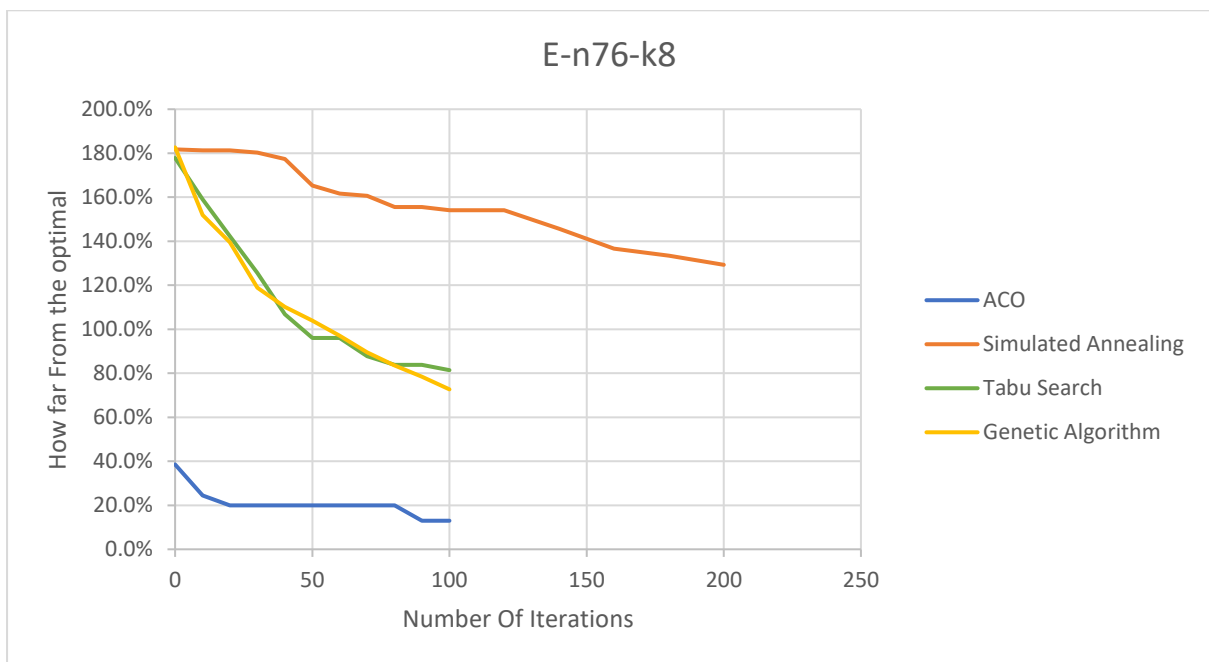
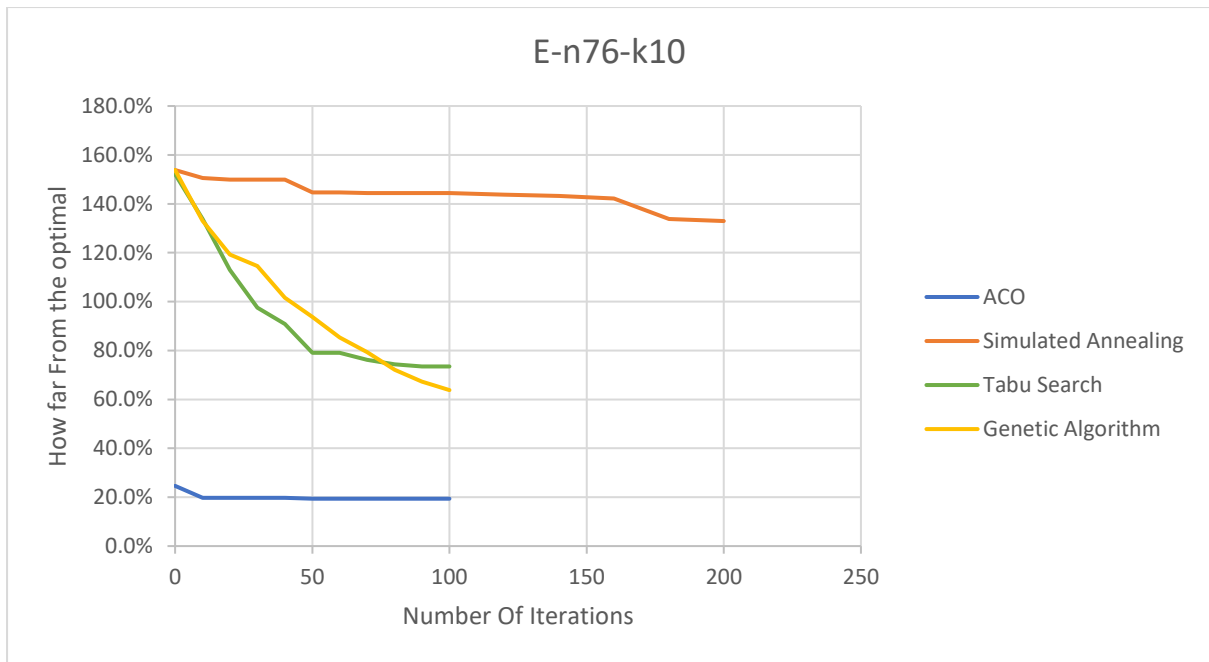


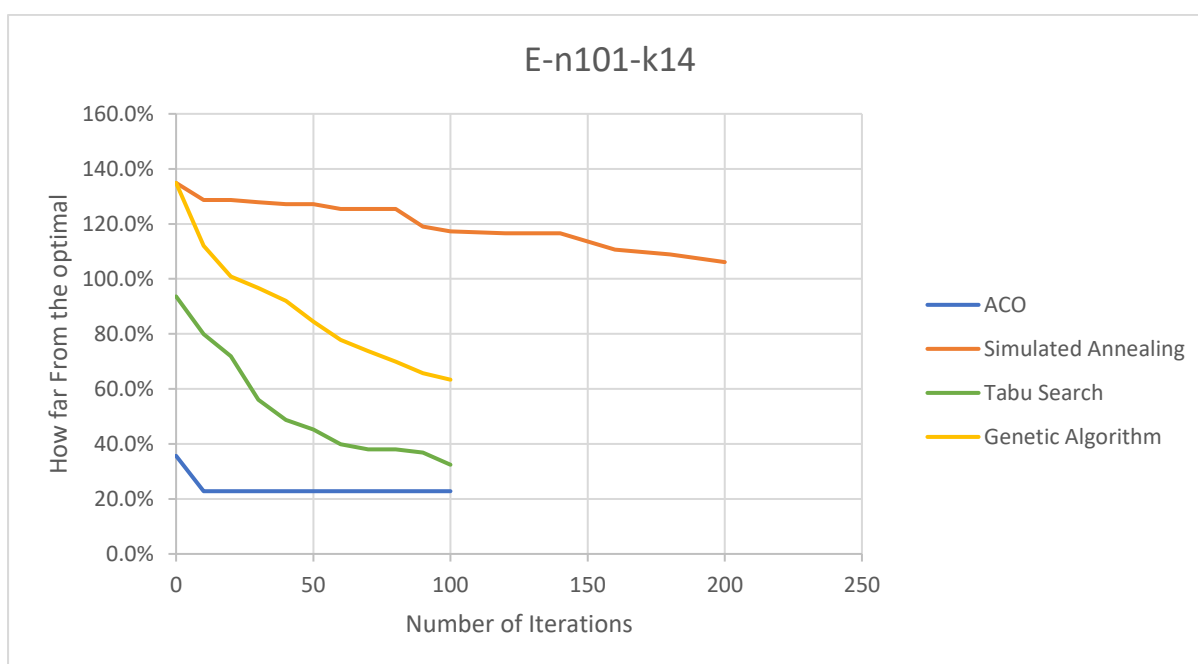
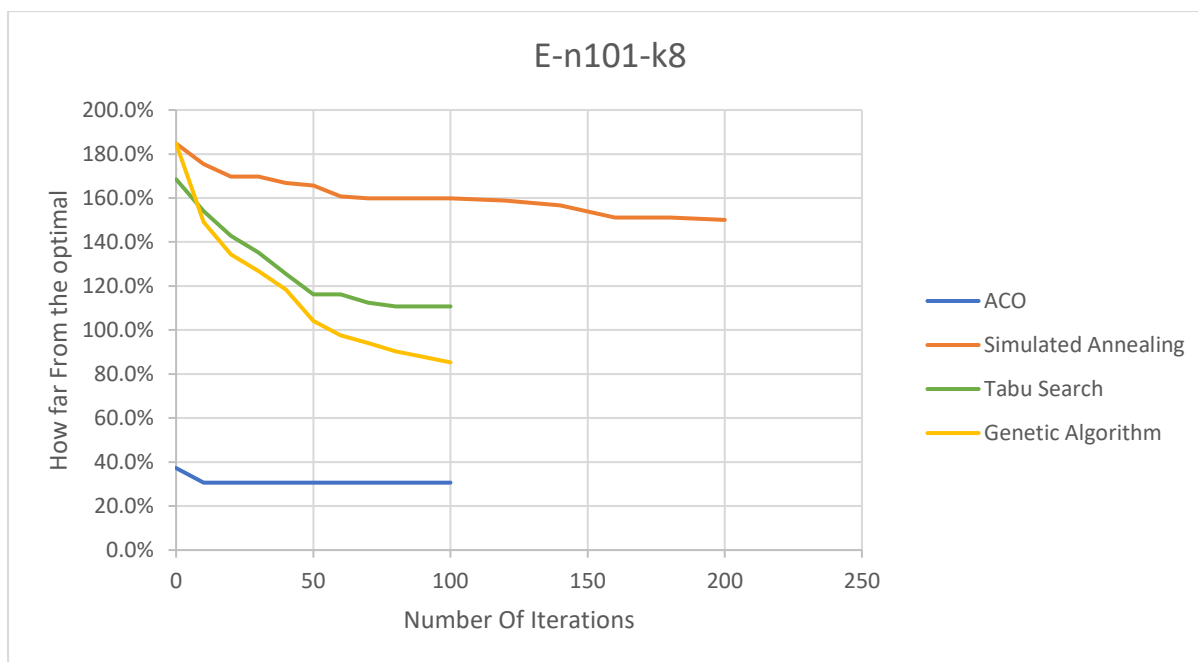


## 10.2 :









11- קודם כל, אנחנו יכולים להגיד עבור כל האלגוריתמים שכלל שהבעיה הולכת וגדלה, אז אנחנו מתרחקים יותר ויותר מהפתרון האופטימלי, למשל עבור אלגוריתם TABU SEARCH, בהתחלה על הבעיה שהיא "צעצוע", הוא נגע בדיוק בפתרון האופטימלי ואילו על בעיות יותר גדולות היה רחוק יחסית מהאופטימלי.

אלגוריתם SIMULATED ANNEALING:

אלגוריתם זה נותן לנו את התוצאה הטובה ביותר על בעיות גדולות אבל על בעיות קטנות הוא לא אכן כך, אנחנו מניחים שתוצאה זו לא כי אלגוריתם זה יותר טוב משאר האלגוריתמים אלה כי נתנו לו לרוץ על יותר איטרציות ואנחנו עשינו את זה כי הוא יותר מהיר משאר האלגוריתמים.

אלגוריתם TABU SEARCH:

אלגוריתם זה הוא הכי טוב על בעיות קטנות אבל על בעיות גדולות הוא יחסית פחות טוב, אני חושב שזה בגלל העובדה שלפעמים הוא לוקח מצב שהוא לא משפר ועבור בעיות גדולות זה ישפיע על הכיוון של מציאת פתרון אופטמלי כלומר שהדרך לפתרון האופטמלי היא לא דווקא מהפתרון "הגרוע" שהוא לקח.

אלגוריתם ACO:

אנחנו באופן אישי היינו מעדיפים אלגוריתם זה כי גם על בעיות גדולות וגם על בעיות קטנות הוא יחסית קרוב לפתרון האופטמלי. אלגוריתם זה למרות שנותנים לו לרוץ מעט איטרציות (100 לעומת 1000000 ב SIMULATED) הוא עדיין קרוב לפתרון האופטמלי.

אלגוריתם GENETIC:

אלגוריתם זה הוא הכי פחות יעיל משאר האלגוריתמים, גם לוקח יותר זמן וגם מגיע יותר רחוק מהפתרון האופטמלי משאר האלגוריתמים, אז אני מניח שלא הייתי הולך להשתמש בו)

12- לפי הגרפים אנו רואים שככל שהבעיה הולכת וגדלה, אז כל האלגוריתמים לוקחים יותר זמן.

אלגוריתם SIMULATED ANNEALING:

אפשר להגיד שעל בעיות קטנות הוא רץ זמן "סביר" והוא כן מחזיר פתרון קרוב לאופטמלי אבל על בעיות יותר רציניות הוא נותן לנו פתרונות טובות והוא רץ בזמן יותר טוב מ ה ACO ו GENETIC לכן בגלל שהוא רץ בזמן מהר אנחנו ננצל את זה ונותנים לו לרוץ יותר איטרציות (1000000 איטרציה).

אלגוריתם TABU SEARCH:

אלגוריתם זה הוא הכי מהיר גם על בעיות גדולות וגם על בעיות יותר קטנות, אבל על בעיות גדולות הוא לא מחזיר לנו את הפתרון הכי טוב לעומת בעיות קטנות שהוא כן מחזיר.

אלגוריתם ACO:

שמנו לב שאלגוריתם זה מאוד תלוי בגודל הבעיה, למשל לפי הגרפים, על בעיות קטנות הוא לוקח פחות זמן משאר האלגוריתמים ואלו עבור בעיות גדולות הוא ייקח יותר זמן מה TABU SEARCH ו SIMULATED ANNEALING.

אלגוריתם GENETIC:

אלגוריתם זה לוקח זמן יותר מכל אלגוריתם אחר גם על בעיות גדולות וגם על בעיות קטנות אבל על בעיות גדולות אם לא מקצים לו זמן ארוך יכול להיות שהוא לא יוכל לסיים ולמצוא פתרון.