Report

Samer Kharouba → 209050202

Abed Abu Hussein → 208517631

Part 1

Section 1:

Heuristics chosen:

1) Clark and wright heuristic.
2) Nearest neighbor heuristic.

Clark and wright heuristic:

This multi-stage heuristic calculates savings between each pair of cities by the formula:

$$savings(i,j) = cost(i,0) + cost(0,i) - cost(i,j)$$

And arranges them in non ascending order.
Starts with initial tracks while i run over the number of cities and starts running on the savings list and connecting two tracks x, y Assuming that their connection does not exceed the contents of the truck, and there is also the bow $(0,j)$ In lane X and bow $(i,0)$ in lane Y,
Until no possible connections remained.

Section 2:

a) Tabu search:

The algorithm gave a relatively good performance compared to the rest of the algorithms, using a building heuristic CW savings.

We think the algorithm can give very good performance if given enough time to run.

```
class tabu(algortithem):
    def __init__(self, target, tar_size, pop_size, problem_spec, fitnessType, max_iter,mutation, selection=None

    def mutatate(self, s):...

    def getNeighbors(self, s):...

    def fitness(self, obj):...

    def algo(self, i):

        # generate neighbors and take the fittest candidatee
        neighborhood = self.getNeighbors(self.bestCandidate)
        self.bestCandidate = neighborhood[0]
        for candidate in neighborhood:
            if candidate not in self.tabuList and self.fitness(candidate) < self.fitness(self.bestCandidate):
                self.bestCandidate = candidate
        if self.fitness(self.bestCandidate) < self.fitness(self.sbest):
            # tabulist = last best candidate
            self.sbest = self.bestCandidate
            self.tabuList.append(self.bestCandidate)

        if len(self.tabuList) > tabuMaxSize:
            self.tabuList.remove(self.tabuList[0])
        # update solution
        self.solution.object = self.sbest
        self.solution.calculate_fittness(self.target, self.target_size, "fitness")
```

b) ACO:

The algorithm gave relatively good performance with the parameters (we tried multiple parameters and we think those are the best parameters):

$\alpha = 4,\ \beta = 3,\ \rho = 0.1,\ q = 100000.$

```
class ACO_alg(algortithem):
    def __init__(self, target, tar_size, pop_size, problem_spec, fitnesstype,max_iter, selection=None):...

    def fitness(self, obj):...
    def algo(self, i):
        cities=self.cities
        if self.glob_counter == round(self.max_iter / 16):
            self.PheromoneMatrix, self.temp = self.initiate_phermones()
            self.glob_counter = 0
            self.current_sol =[]

        visited = self.visited_cities(cities)
        dist = self.sum_of_distances(visited)

        self.PheromoneMatrix = self.updatePheromone(self.PheromoneMatrix, visited, dist)
        if len(self.current_sol) == 0 or len(self.global_sol_) == 0 or self.fitness(self.current_sol) > self.fitness(visited):
            self.current_sol = visited
            if len(self.global_sol_) == 0:
                self.global_sol = visited

        if self.fitness(self.current_sol) < self.fitness(self.global_sol_):
            self.glob_counter = 0
            self.global_sol_ = self.current_sol
            self.glob_visit.append(self.global_sol_)
        else:
            self.glob_counter += 1


        if self.fitness(self.current_sol)<self.fitness(self.global_sol):
            self.glob_counter = 0
            self.global_sol = self.current_sol
            self.glob_visit.append(self.global_sol)
        else:
            self.glob_counter += 1

        self.solution.object=self.global_sol
        self.solution.calculate_fittness(self.target, self.target_size, "fitness")

    def visited_cities(self, cities):...

    def initiate_phermones(self):...


    def probabilities_of_visited(self,matrix, visited):...

    def sum_of_distances(self,cities_dist):...

    def updatePheromone(self,matrix, visited, pathLen):...

    def stopage(self, i):
        return False
```

c) Simulated annealing:

Similar to Tabu Search its performance with CW savings and inverse swapping / 2Opt enhancement methods Were the best, although the error rate from the optimal solution was greater than TS.

```
class simulated_annealing(algortithem):
    def __init__(self, target, tar_size, pop_size, problem_spec, fitnessType, max_iter,mutation, selection=None):...

    def fitness(self, obj):...
    def mutatate(self,s):...
    def getNeighbors(self, s):...
    def get_best_Neighbour(self,neighborhood):...
    def algo(self, i):

        neighborhood = self.getNeighbors(self.s)
        sNew = self.get_best_Neighbour(neighborhood)
        de = self.fitness(self.s) - self.fitness(sNew)
        temp = random.random()
        if de > 0 or math.exp(float(de) / 1.38 * float(self.t + 0.000001)) > temp:
            self.s = sNew
        self.t = float(self.t * self.alpha)
        if self.fitness(self.s) < self.fitness(self.global_best):
            self.global_best = self.s

        # update solution
        self.solution.object = self.global_best
        self.solution.calculate_fittness(self.target, self.target_size, "fitness")
```

Section 3:

Genetic algorithm: from previous assignment.

We made changes only on the input and didn’t touch the old algorithm.

Section 4:

Cooperative PSO: from previous assignment.

We made changes only on the input and didn't touch the old algorithm.

PART 2

טיפול בקלט/פלט, ניתוח הבעיה ובנית אב-טיפוס לפטרון – CVRP

Sections 1+2

How we perceived and presented the problem:

We captured the file from the user and created cities and each city has an array for the distance of the neighbors from the current city.

We have represented the problem by permutation so that each city appears once and is represented in the permutation by its id which receive it as input.

Section 3

1) Clark and wright
2) Nearest neighbor

Section 4

1) Tabu search
2) Simulated annealing simplified
3) ACO

Section 5

The first heuristic we used was a type of constructive heuristic nearest neighbor: This heuristic produces an initial route, so that at first you choose a random city and another: nearest neighbor.
this is how you choose the city closes to it until the contents of the truck run out and so on.

Section 6:

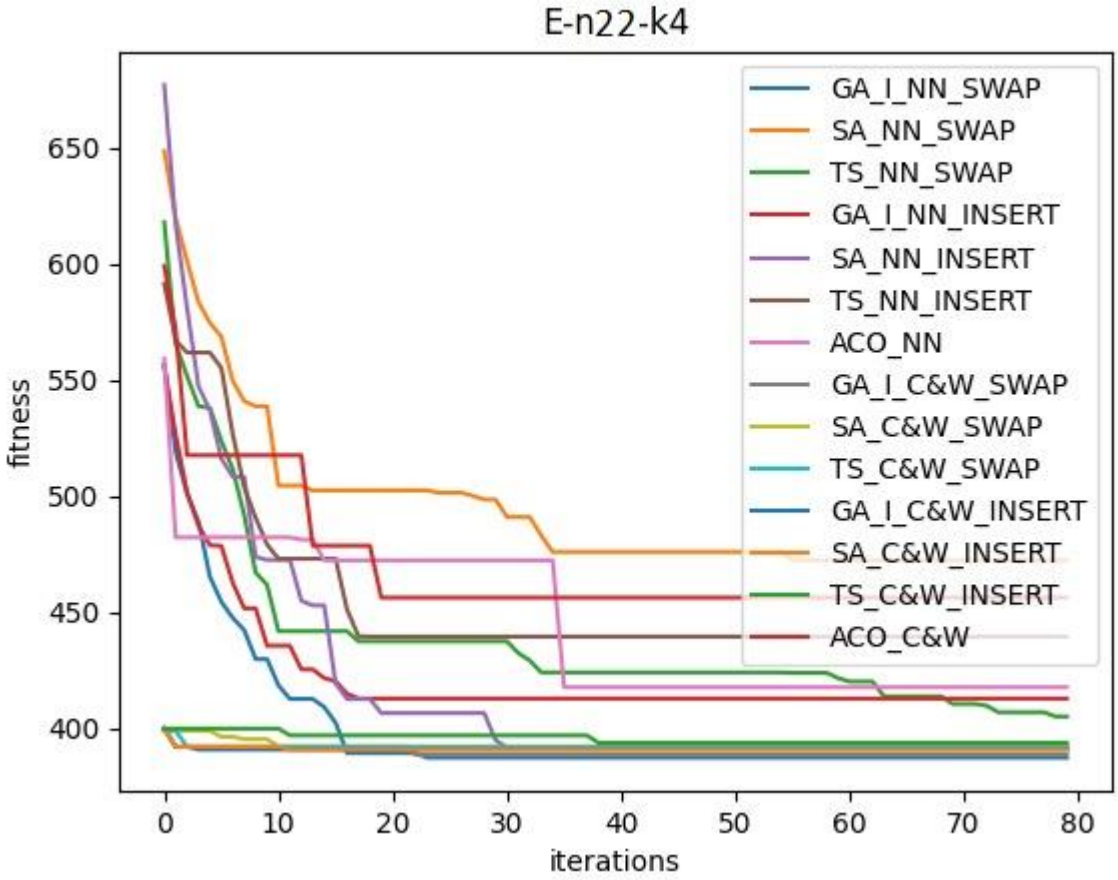
```
GA_I_NN_SWAP
 Best:10,8,6,3,2,7,11,9,4,5,12,14,13,16,18,17,15,19,21,22,20,Fitness: 387.5429560469644
Car 1: 0 9 7 5 2 1 6 0
Car 2: 0 10 8 3 4 11 13 0
Car 3: 0 12 15 17 16 14 0
Car 4: 0 19 21 20 18 0
 ,fittness: 387.5429560469644  Time :  7.8489662  ticks: 7.8488805294036865
SA_NN_SWAP
 Best:17,15,13,6,14,20,22,18,21,19,16,11,9,12,10,8,7,5,4,2,3,Fitness: 472.5327818969959
Car 1: 0 16 14 12 5 0
Car 2: 0 13 19 21 17 0
Car 3: 0 20 18 15 10 8 11 0
Car 4: 0 9 7 6 4 3 1 2 0
 ,fittness: 472.5327818969959  Time :  0.17642069999999954  ticks: 0.17604732513427734
TS_NN_SWAP
 Best:20,17,14,16,19,21,22,18,15,13,10,6,8,11,12,9,7,3,2,4,5,Fitness: 405.2751409933177
Car 1: 0 19 16 13 0
Car 2: 0 15 18 20 21 17 14 0
Car 3: 0 12 9 5 7 10 0
Car 4: 0 11 8 6 2 1 3 4 0
 ,fittness: 405.2751409933177  Time :  0.19153859999999945  ticks: 0.19154715538024902
GA_I_NN_INSERT
 Best:11,12,5,4,7,8,10,6,3,2,9,14,13,16,19,21,18,15,17,20,22,Fitness: 412.9774354269643
Car 1: 0 11 4 3 6 7 9 10 0
Car 2: 0 5 2 1 8 13 0
Car 3: 0 12 15 18 20 17 0
Car 4: 0 14 16 19 21 0
 ,fittness: 412.9774354269643  Time :  7.926954500000001  ticks: 7.926443576812744
```

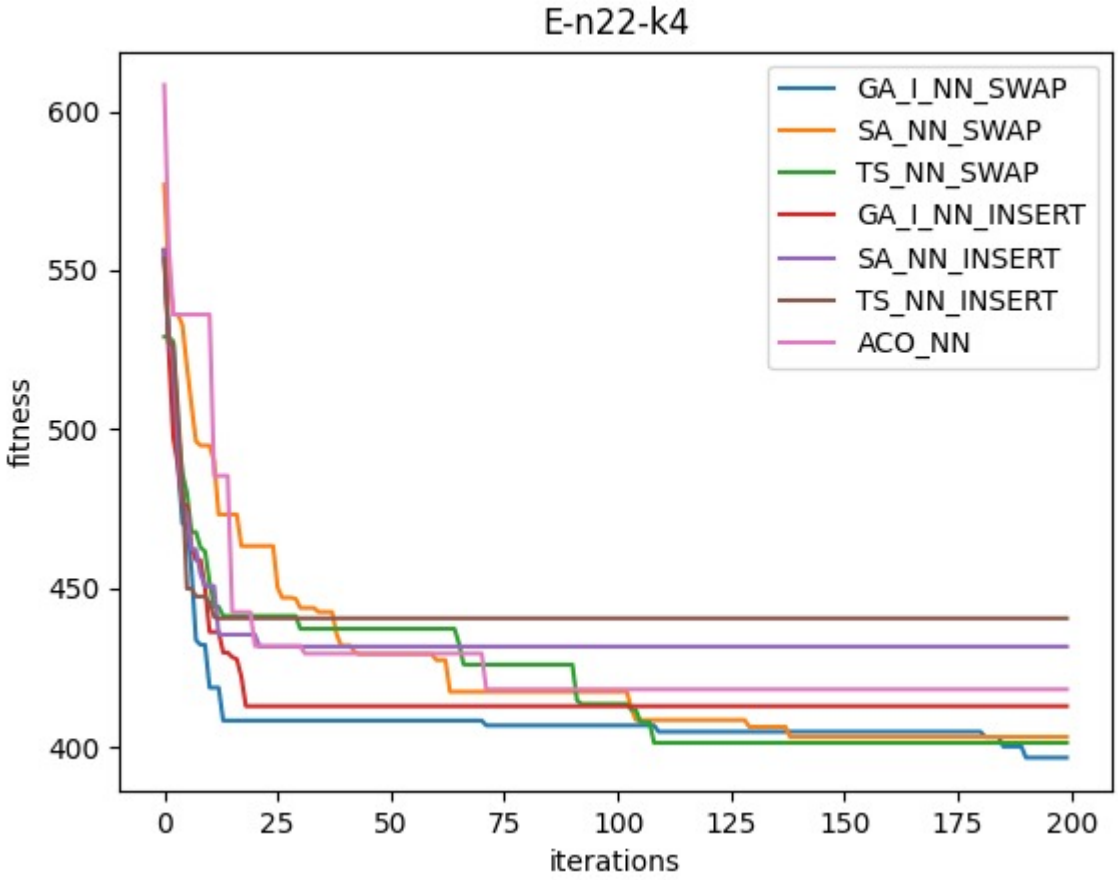
```
SA_NN_INSERT
 Best:10,8,6,3,2,11,12,5,4,7,9,14,18,21,19,16,13,15,17,20,22,Fitness: 391.8896793091878
Car 1: 0 9 7 5 2 1 10 0
Car 2: 0 11 4 3 6 8 13 0
Car 3: 0 17 20 18 15 12 0
Car 4: 0 14 16 19 21 0
 ,fittness: 391.8896793091878  Time :  0.16935729999999793  ticks: 0.17003083229064941
TS_NN_INSERT
 Best:15,14,20,22,18,21,19,16,17,13,10,6,8,11,5,4,12,9,7,3,2,Fitness: 439.65540793523405
Car 1: 0 14 13 19 21 17 0
Car 2: 0 20 18 15 16 0
Car 3: 0 12 9 5 7 10 0
Car 4: 0 4 3 11 8 6 2 1 0
 ,fittness: 439.65540793523405  Time :  0.22137400000000085  ticks: 0.22105145454406738
ACO_NN
 Best:18,15,17,13,16,19,21,22,20,14,12,4,5,7,9,11,10,8,6,3,2,Fitness: 417.8573669848362
Car 1: 0 17 14 16 12 15 0
Car 2: 0 18 20 21 19 0
Car 3: 0 13 11 3 4 6 8 10 0
Car 4: 0 9 7 5 2 1 0
 ,fittness: 417.8573669848362  Time :  0.3109737999999993  ticks: 0.31036901473999023
GA_I_C&W_SWAP
 Best:19,21,22,20,10,8,6,3,2,7,9,14,12,4,5,11,17,15,18,13,16,Fitness: 388.7722858391293
Car 1: 0 18 20 21 19 0
Car 2: 0 9 7 5 2 1 6 8 0
Car 3: 0 13 11 4 3 10 0
Car 4: 0 12 15 17 16 14 0
 ,fittness: 388.7722858391293  Time :  13.000229299999997  ticks: 13.0000479221344
```

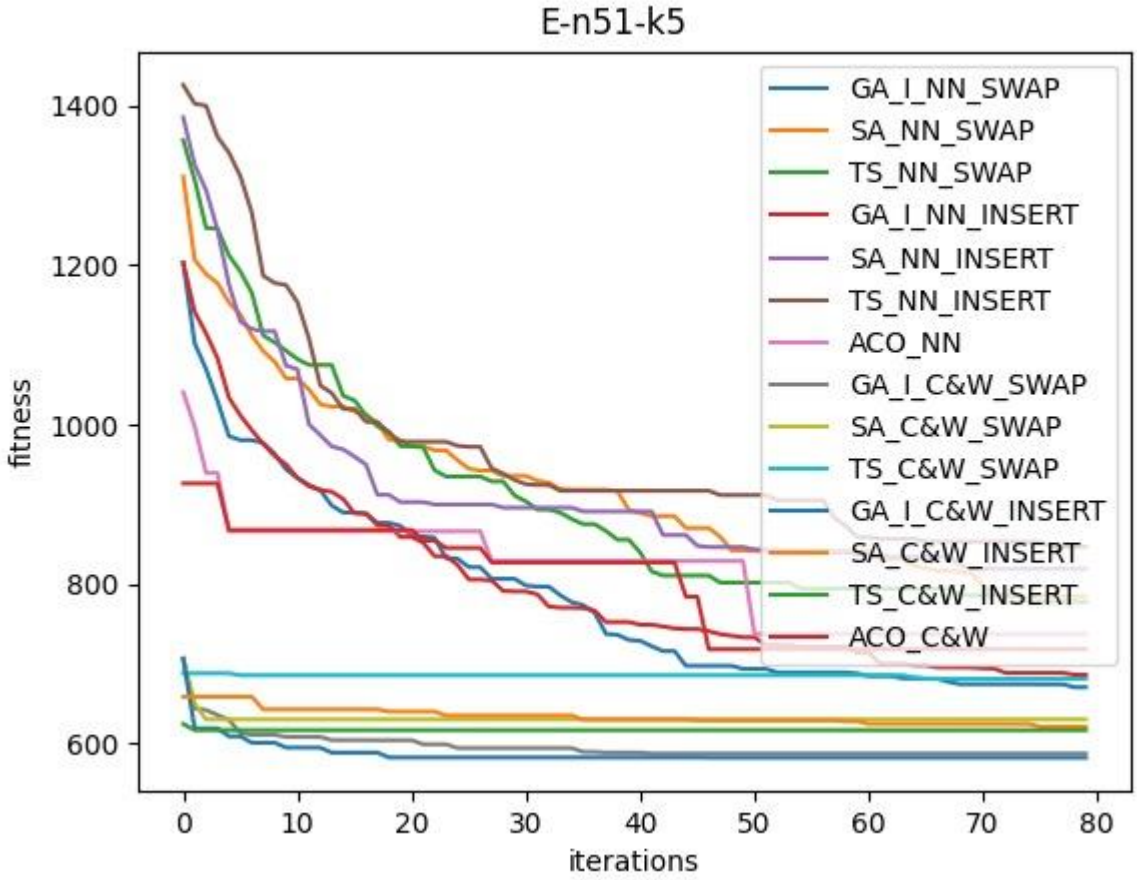
```
SA_C&W_SWAP
 Best:19,21,22,20,10,8,6,3,2,7,9,14,12,5,4,11,17,15,18,16,13,Fitness: 392.2655511441565
Car 1: 0 18 20 21 19 0
Car 2: 0 9 7 5 2 1 6 8 0
Car 3: 0 13 11 4 3 10 0
Car 4: 0 16 14 17 15 12 0
 ,fittness: 392.2655511441565  Time :  5.468596400000003  ticks: 5.469094276428223
TS_C&W_SWAP
 Best:19,21,22,20,10,8,6,3,2,7,9,14,12,5,4,11,17,15,18,16,13,Fitness: 392.2655511441565
Car 1: 0 18 20 21 19 0
Car 2: 0 9 7 5 2 1 6 8 0
Car 3: 0 13 11 4 3 10 0
Car 4: 0 16 14 17 15 12 0
 ,fittness: 392.2655511441565  Time :  5.401179599999999  ticks: 5.401923418045044
GA_I_C&W_INSERT
 Best:19,21,22,20,10,8,6,3,2,7,14,12,5,4,9,11,17,15,18,16,13,Fitness: 391.0362213519916
Car 1: 0 18 20 21 19 0
Car 2: 0 9 7 5 2 1 6 0
Car 3: 0 13 11 4 3 8 10 0
Car 4: 0 16 14 17 15 12 0
 ,fittness: 391.0362213519916  Time :  13.077825600000004  ticks: 13.077657699584961
SA_C&W_INSERT
 Best:19,21,22,20,10,8,6,3,2,7,14,12,5,4,9,11,17,15,18,16,13,Fitness: 391.0362213519916
Car 1: 0 18 20 21 19 0
Car 2: 0 9 7 5 2 1 6 0
Car 3: 0 13 11 4 3 8 10 0
Car 4: 0 16 14 17 15 12 0
 ,fittness: 391.0362213519916  Time :  5.392942100000006  ticks: 5.393404960632324
```

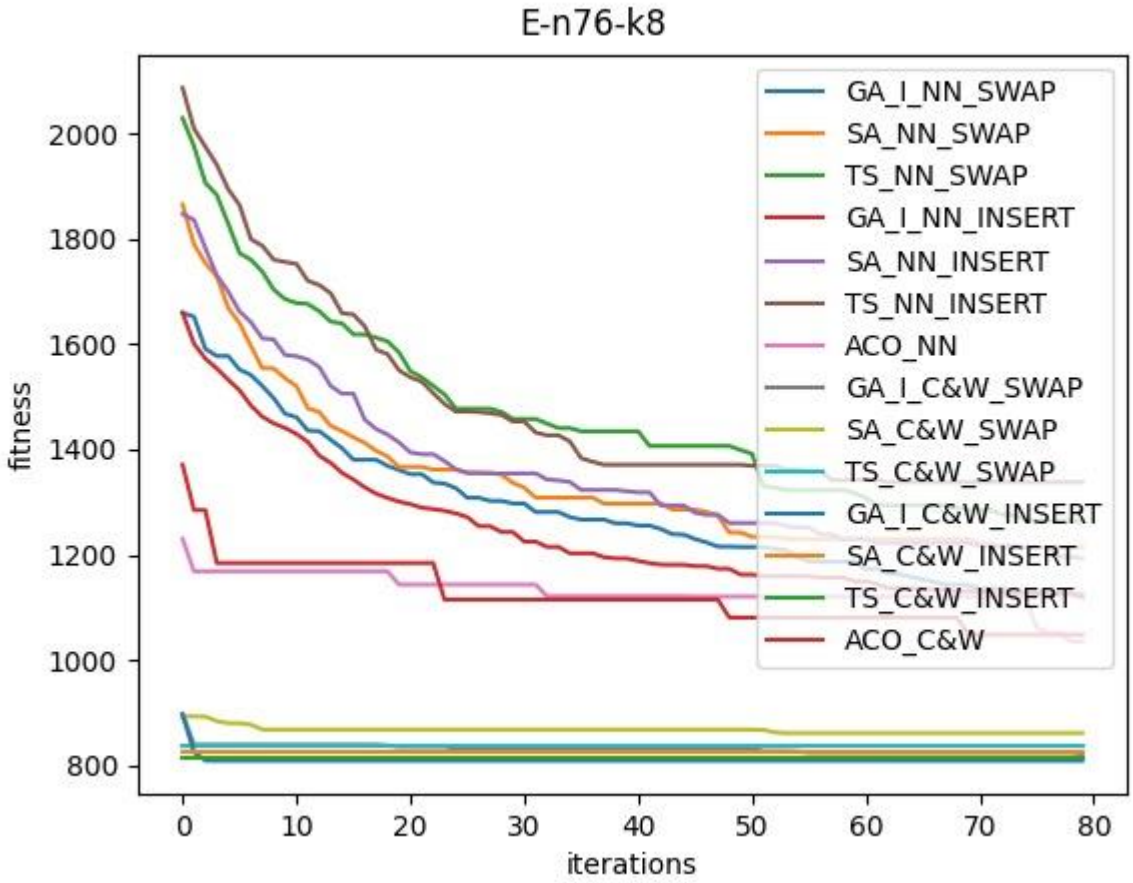
```
TS_C&W_INSERT
 Best:19,21,22,20,10,8,6,3,2,14,12,5,4,7,9,11,15,17,18,16,13,Fitness: 393.95269856094166
Car 1: 0 18 20 21 19 0
Car 2: 0 9 7 5 2 1 0
Car 3: 0 13 11 4 3 6 8 10 0
Car 4: 0 14 16 17 15 12 0
 ,fittness: 393.95269856094166  Time :  5.3612124000000065  ticks: 5.361428737640381
ACO_C&W
 Best:22,21,19,16,13,18,15,17,20,14,12,11,9,7,6,8,10,3,2,4,5,Fitness: 456.41940447831837
Car 1: 0 21 20 18 15 12 0
Car 2: 0 17 14 16 19 0
Car 3: 0 13 11 10 8 6 5 0
Car 4: 0 7 9 2 1 3 4 0
 ,fittness: 456.41940447831837  Time :  5.574840899999998  ticks: 5.575167655944824
```

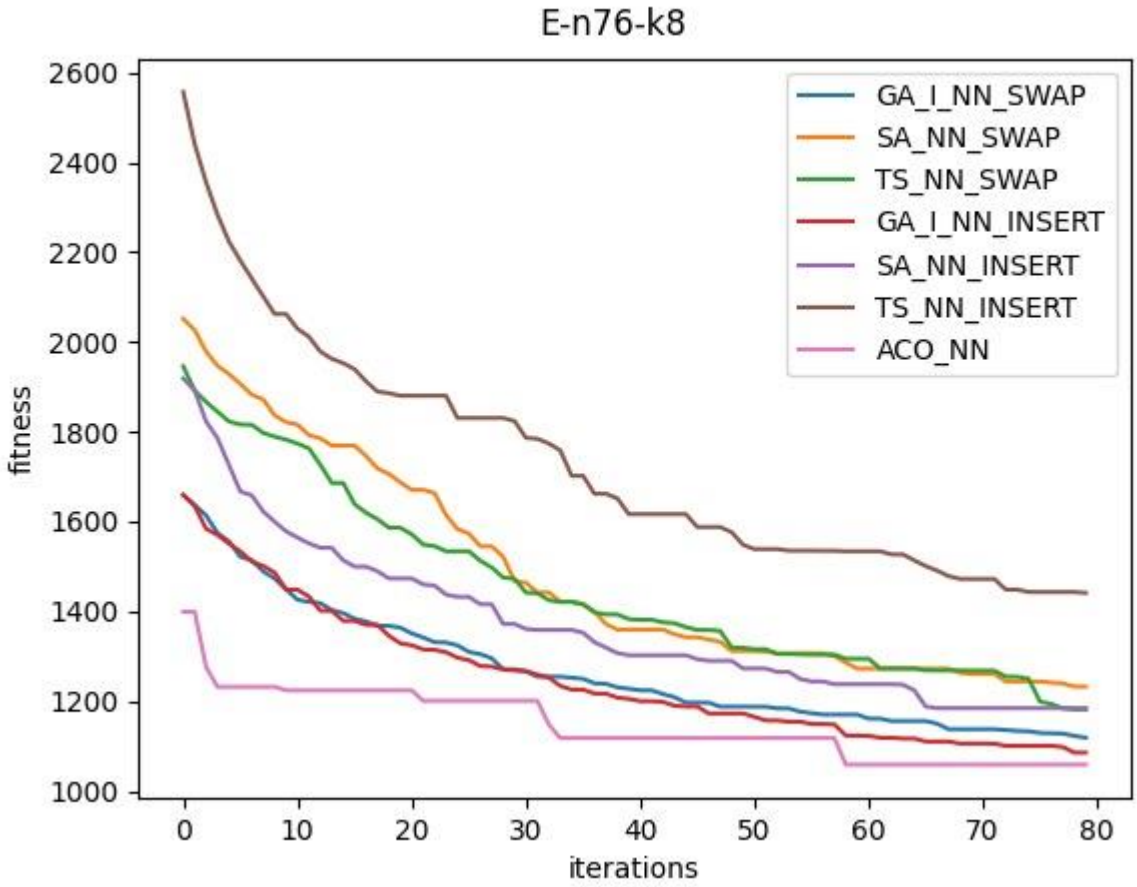
section 7:

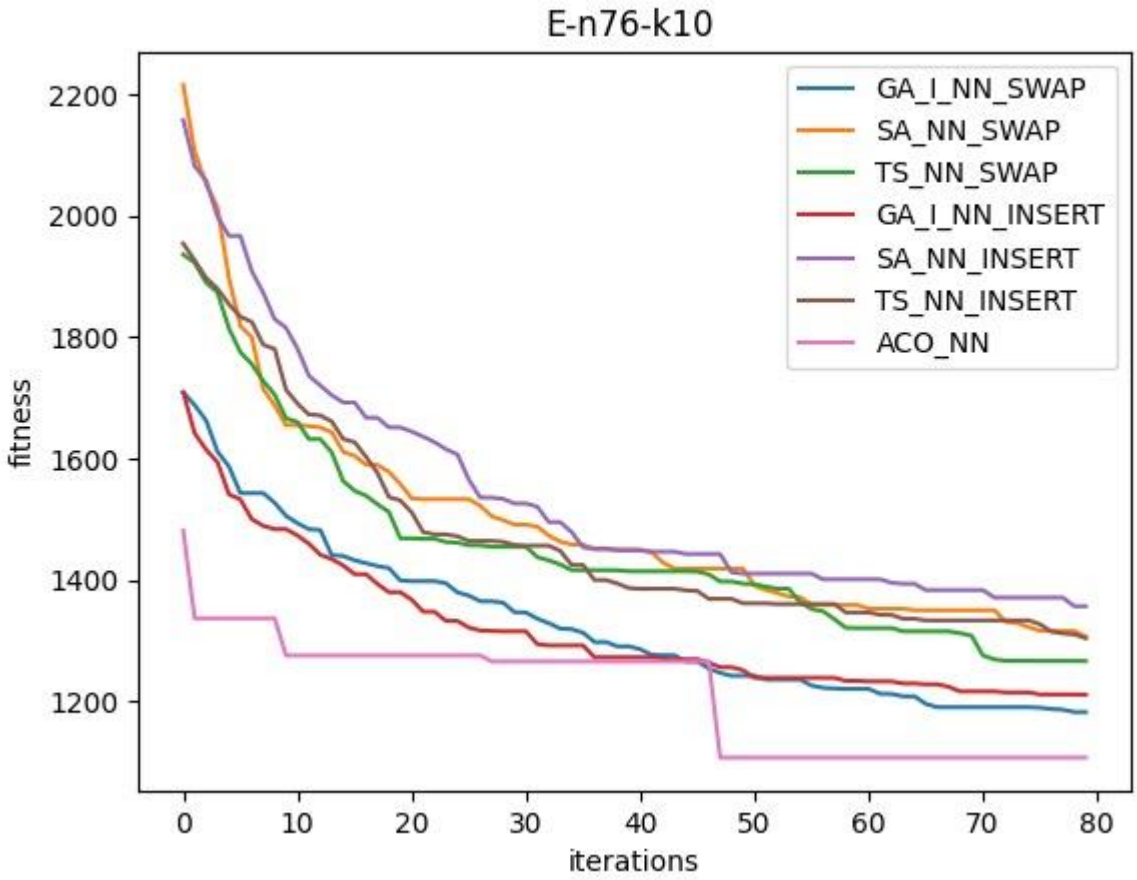

E-n22-k4
fitness
iterations
650
600
550
500
450
400
0
10
20
30
40
50
60
70
80
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
GA_I_C&W_SWAP
SA_C&W_SWAP
TS_C&W_SWAP
GA_I_C&W_INSERT
SA_C&W_INSERT
TS_C&W_INSERT
ACO_C&W

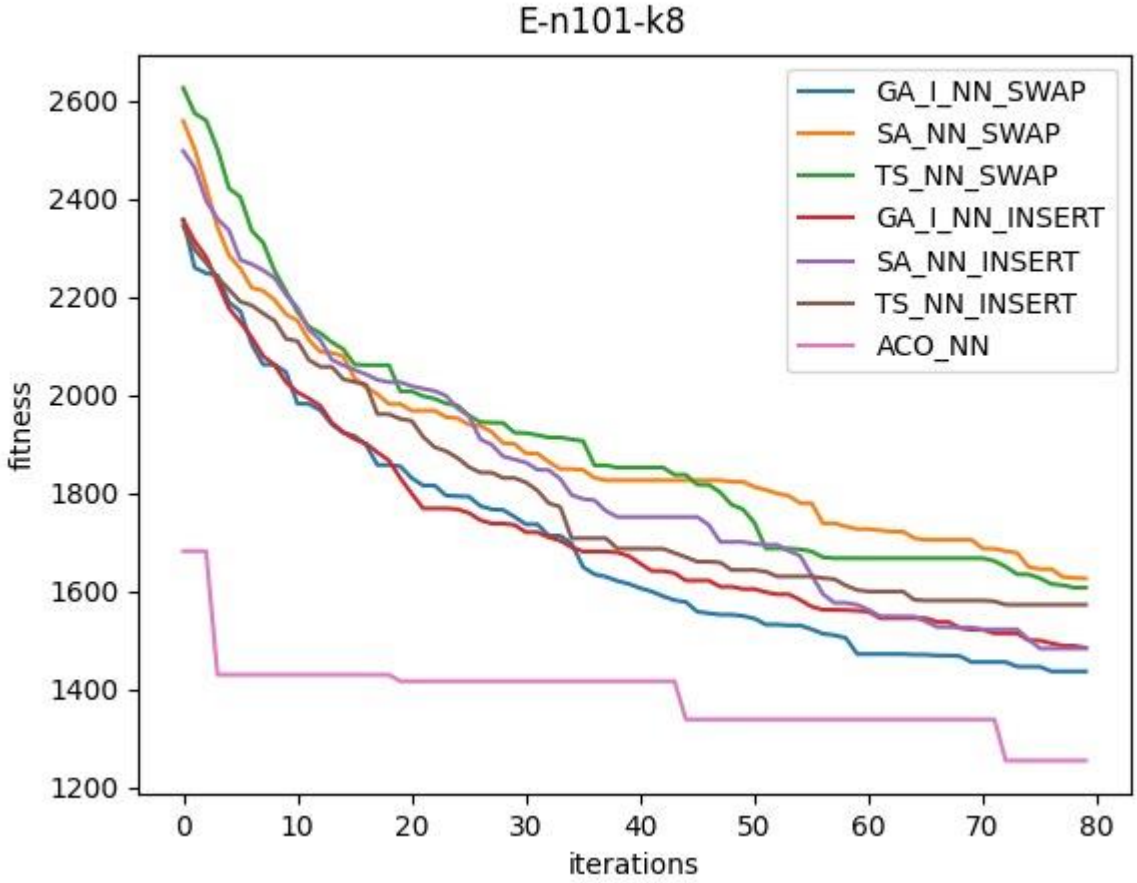

E-n22-k4
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
fitness
600
550
500
450
400
0
25
50
75
100
125
150
175
200
iterations

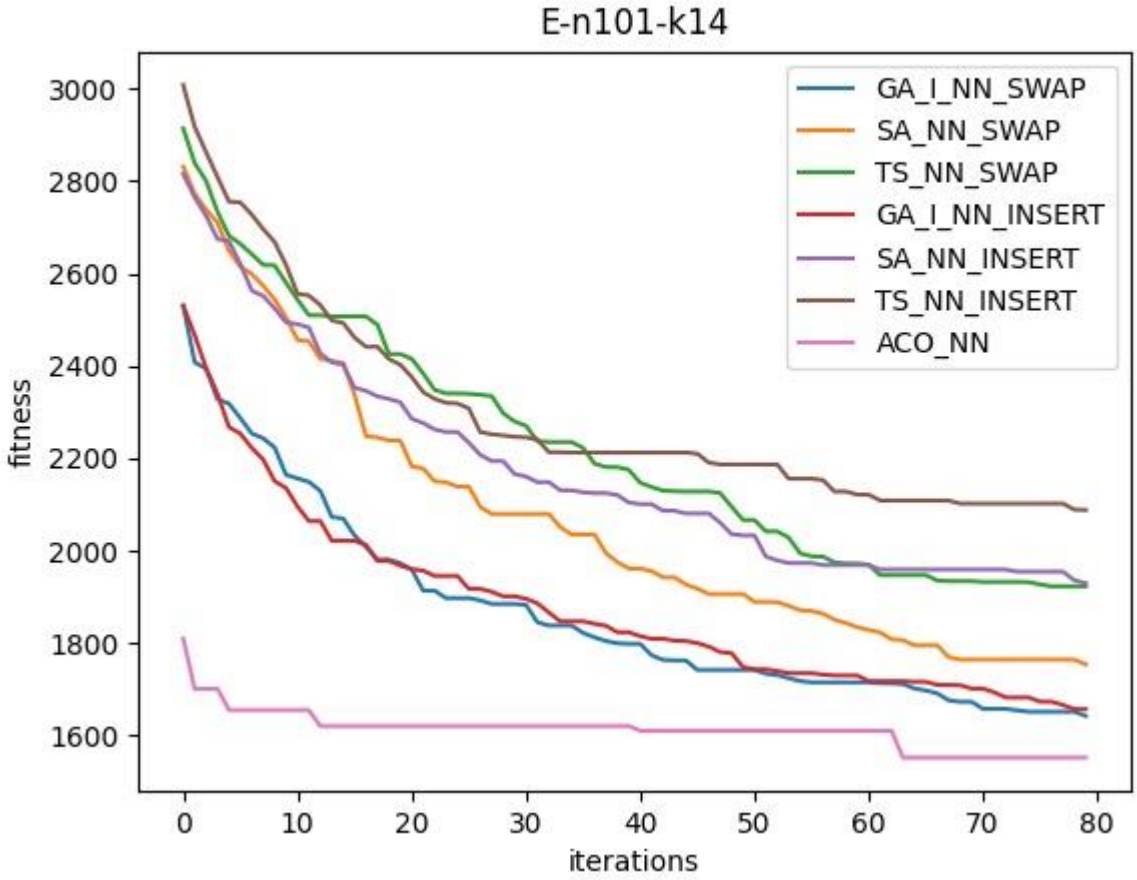

E-n51-k5
fitness
iterations
1400
1200
1000
800
600
0
10
20
30
40
50
60
70
80
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
GA_I_C&W_SWAP
SA_C&W_SWAP
TS_C&W_SWAP
GA_I_C&W_INSERT
SA_C&W_INSERT
TS_C&W_INSERT
ACO_C&W


E-n76-k8
fitness
iterations
2000
1800
1600
1400
1200
1000
800
0
10
20
30
40
50
60
70
80
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
GA_I_C&W_SWAP
SA_C&W_SWAP
TS_C&W_SWAP
GA_I_C&W_INSERT
SA_C&W_INSERT
TS_C&W_INSERT
ACO_C&W


E-n76-k8
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
fitness
iterations
2600
2400
2200
2000
1800
1600
1400
1200
1000
0
10
20
30
40
50
60
70
80


E-n76-k10
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
fitness
iterations
2200
2000
1800
1600
1400
1200
0
10
20
30
40
50
60
70
80


E-n101-k8
GA_I_NN_SWAP
SA_NN_SWAP
TS_NN_SWAP
GA_I_NN_INSERT
SA_NN_INSERT
TS_NN_INSERT
ACO_NN
fitness
iterations
2600
2400
2200
2000
1800
1600
1400
1200
0
10
20
30
40
50
60
70
80



Results chart:

|E-n22-k4|

Algorithem	fitness	Time	ticks
GA_I_NN_SW	387.5429560469644	7.8489662	7.84888052940368
SA_NN_SW	472.5327818969959	0.17642069999999954	0.176047325134277
TS_NN_SW	405.2751409933177	0.19153859999999945	0.191547155380249
GA_I_NN_INSE	412.9774354269643	7.926954500000001	7.9264435768127
SA_NN_INSE	391.8896793091878	0.16935729999999793	0.170030832290649
TS_NN_INSE	439.65540793523405	0.22137400000000085	0.221051454544067
ACO_	417.8573669848362	0.3109737999999993	0.310369014739990
GA_I_C&W_SW	388.7722858391293	13.000229299999997	13.00004792213
SA_C&W_SW	392.2655511441565	5.468596400000003	5.4690942764282
TS_C&W_SW	392.2655511441565	5.401179599999999	5.4019234180450
GA_I_C&W_INSE	391.0362213519916	13.077825600000004	13.0776576995849
SA_C&W_INSE	391.0362213519916	5.392942100000006	5.3934049606323
TS_C&W_INSE	393.95269856094166	5.3612124000000065	5.3614287376403
ACO_C	456.41940447831837	5.574840899999998	5.5751676559448

|E-n33-k4|

Algorithem	fitness	Time	ticks
GA_I_NN_SW	883.8369019477129	18.516504700000013	18.517015457153
SA_NN_SW	951.7603358519534	0.3428380999999945	0.34210300445556
TS_NN_SW	949.8956746791949	0.25980290000001105	0.259272336959838
GA_I_NN_INSE	896.5445369160233	15.494049000000018	15.4946413040161
SA_NN_INSE	1024.4479620133382	0.34646940000004633	0.346991062164306
TS_NN_INSE	1030.2969102665788	0.4127925000000232	0.412266969680786
ACO_	947.97121006277	0.7852495000000204	0.7854847908020
GA_I_C&W_SW	846.5057013190805	65.09921539999999	65.099011898040
SA_C&W_SW	851.7653631367309	41.08235450000001	41.082729101181
TS_C&W_SW	854.702366469085	40.1822631	40.1826956272125
GA_I_C&W_INSE	848.124998958294	61.6686383	61.669096469879
SA_C&W_INSE	851.0784970217779	45.922766999999965	45.922227859497
TS_C&W_INSE	850.4277235016666	48.63766379999993	48.6377313137054
ACO_C	951.8121102382629	40.578713600000015	40.578891515731

|E-n51-k5|

Algorithem	fitness	Time	ticks
GA_I_NN_SW	670.6263955154676	18.4567495	18.4573485851287
SA_NN_SW	784.7868642054705	0.26965659999996205	0.27027583122253
TS_NN_SW	777.7613211268157	0.2641996999999492	0.263923883438110
GA_I_NN_INSE	686.3727151670813	16.078544299999976	16.0785877704620
SA_NN_INSE	819.009012586619	0.2567916000000423	0.25712895393371

- In outputs file we can find each data set’s charts with a run on all algorithms and their variations and a result summing up all those results.