UNIVERSITÀ DI PISA

Master's degree in Artificial Intelligence and Data Engineering

Symbolic and Evolutionary Artificial Intelligence project documentation

MULTIOBJECTIVE ANT COLONY OPTIMIZATION ALGORITHM

Denny Meini, Matteo Razzai

A.Y. 2023/2024

GitHub repository: https://github.com/matteorazzai1/MOEA-D-ACO

Index

1. Introduction

1.1 Project description

The aim of this project is to make the Ant Colony Optimization (ACO) algorithm a multiobjective optimization algorithm. The resulted algorithm has been applied to the Traveling Salesman Problem (TSP), which is a problem that plays a very important role in the development of the ACO algorithm, indeed the Ant System algorithm (see chapter 2), which is the first ACO algorithm, was first tested on the TSP.

1.2 Multiobjective problem

A multiobjective optimization problem (MOP) is defined by:

$$\begin{cases} \min f(x) = \big(f_1(x), f_2(x), \dots, f_s(x)\big) \\ x \in X \end{cases}$$

where *s* is the number of objectives to be simultaneously optimized.

This happens in many real-life applications, in which a decision maker has several conflicting objectives to consider and wants to determine an optimal tradeoff among them. A Pareto-optimal solution to a MOP is a candidate to be an optimal tradeoff. In a MOP may exist many Pareto-optimal solutions, the Pareto set (PS)/Pareto front (PF) is the set of all the Pareto-optimal solutions in the decision/objective space.

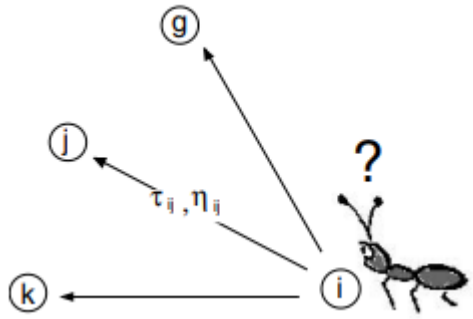
We need the decision maker preference information to decide which Pareto-optimal solution is the best approximation between the PS and/or PF.

1.3 Traveling Salesman Problem

The Traveling Salesman Problem is the problem of a salesman who, starting from his hometown, has to visit a certain group of towns, and he wants to determine the shortest tour that takes him through each of these towns. This problem can be represented by a complete weighted graph $G = (N, A)$ with N the number of nodes representing the cities and A the set of arcs.

There are several reasons for the choice of the TSP as the problem to explain the working of ACO algorithms, it is an important *NP*-hard optimization problem, it is easily understandable, it is a problem to which ACO algorithms are easily applied and eventually is demonstrated that if an ACO algorithm is the most efficient for the TSP problem it will be also among the most efficient ones for a variety of other problems.

In all available ACO algorithms for TSP, the pheromone trails τ_{ij} correspond to the desirability of the arcs *(i,j)* and the heuristic information is chosen as $\eta_{ij} = \frac{1}{d_{ij}}$, which is the heuristic desirability of arc *(i,j)*. In case of $d_{ij} = 0$ the heuristic value η_{ij} is set to a very small number.



In the image on the left we can see how an ant on city *i* decides which city to move to depending on the two measures mentioned earlier.

2. Ant System

The ACO (Ant Colony Optimization) algorithm has been developed in many versions, the version from which we started is the Ant System (AS) algorithm.

Initially, three different versions of AS were proposed: ant-density, ant-quantity and ant-cycle. In the first two quoted versions the ants updated the pheromone directly after a move from a city to an adjacent city, in the ant-cycle version, instead, the pheromone update was only done after all the ants had constructed the tours and the amount of pheromone deposited by each ant was set to be a function of the tour quality. The third version outperformed the other two due to superior performance [1].

The two main phases of the AS algorithm constitute the ants' solution construction and the pheromone update.

2.1 Tour Construction

In AS, *m* ants concurrently build a tour of the TSP (Traveling Salesman Problem). At the beginning, an ant starts its tour on a randomly chosen node. At each construction step, ant *k* applies a probabilistic action choice rule to decide to which city the ant must go at the next step. The probability used by ant k to decide from city i to which city j go is the following:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \quad , \quad if\ j \in N_i^k \ ,$$

where $\eta_{ij} = \frac{1}{d_{ij}}$ is a heuristic value available a priori, α and β are two parameters which respectively influence the pheromone trial from city *i* to city *j* τ_{ij} and the heuristic information η_{ij}, and N_i^k is the neighborhood of ant *k* when being at city *i*, which is the set of city that ant k has not visited yet.

By this probabilistic rule, we can see that the probability of going from node *i* to node *j* increases with the pheromone trail and the heuristic information associated with the arcs *ij*.

The usage of α and β are related to make more important the heuristic information, and so the distance between two cities (setting $\alpha=0$), or the pheromone trail (setting $\beta=0$). In this latter case, if $\alpha>1$ this can lead to the rapid emergence of a stagnation situation, in which all the ants follow the same path and construct the same tour.

Each ant k maintains a memory of the city already visited; in the order they were visited from the ant. In this way the ant can keep trace of its own tour and can determine the feasible neighborhood N_i^k.

2.2 Update of Pheromone

After the construction of the tour, the pheromone will be updated on the trails. This is done in two phases:

- **Evaporation**: The evaporation of the pheromone is implemented in the following way

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \quad , \quad \forall (i,j) \in L$$

  where L is the set of arcs and $0 < \rho \leq 1$ is the pheromone evaporation rate. This value allows to avoid unlimited accumulation of pheromone and to forget about bad decisions taken in previous step. In fact, in this way, if an arc is not crossed by any ant, the value of pheromone on that arc will decrease exponentially with the number of iterations.

- **Depositing**: After the evaporation, all ants deposit pheromone on the arcs they have crossed during their tour

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \quad , \quad \forall (i,j) \in L$$

  where $\Delta\tau_{ij}^k$ is the amount of pheromone ant *k* deposits on the arcs it has visited, and it is defined as follows:

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k & , \text{ if arc } (i,j) \text{belongs to } T^k ; \\ 0 & , \text{ otherwise ;} \end{cases}$$

  where C^k is the length of the tour T^k constructed by the ant k, computed as the sum of the lengths of the arcs belonging to the tour.

From this last equation, we can see that the arcs crossed by more ants and that belongs to short tours, receive more pheromone and so they will be more likely to be chosen by other ants in the future.

3. MOEA/D

The algorithm we chose to make ACO a multiobjective algorithm is MOEA/D. MOEA/D stands for MultiObjective Evolutionary Algorithm based on Decomposition and how told by its name, do its job decomposing the original multiobjective problem into a list of smaller single-objective subproblems.

The algorithm creates N lambda-vectors, where N is the number of individuals of the population. Each lambda-vector indicates a direction into the objective space and is assigned to an agent, that will try to minimize the values moving in that direction. The lambda-vectors have the following structure:

$$\lambda^i = (\lambda_1^i, \lambda_2^i, \dots, \lambda_m^i),\ i = 1, \dots, N,$$

$$\sum_{j=1}^{m} \lambda_j^i = 1,\ \lambda_1^i, \dots, \lambda_m^i > 0,\ i = 1, \dots, N,$$

where *m* is the number of objective functions.

The agent *i* has to solve one of the two subsequent optimization problem:

$$g^{ws}(x, \lambda^i) = \sum_{j=1}^{m} \lambda_j^i f_j(x)$$

if we choose the weighted sum methodology, or:

$$g^{tc}(x|\lambda^i, z) = \sum_{j=1}^{m} \lambda_j^i |f_j(x) - z_j|$$

if we choose the Tchebycheff methodology, where *z* is the ideal point and its components z_j are the best values find for each objective function f_j.

In this way the agent will find a solution which is part of the Pareto front.

Each agent has a neighborhood, which is composed of the T agents with the nearest subproblem to resolve with respect to the agent itself, it included. At each generation every agent obtains the current solutions generated by some of their neighbors, then it creates a new solution applying the

reproduction operators, which are crossover and mutation, successively it updates the optimal point z (if we chose the Tchebycheff approach), and finally it replaces its solution, if the new decomposed cost $g^{tc}(y|\lambda^j, z)$ (or $g^{ws}(y, \lambda^j)$ in the weighted sum approach) is lower than the old one $g^{tc}(x^i|\lambda^j, z)$ ($g^{ws}(x^i, \lambda^j)$). The following step will be to confront the various solutions found by the neighbors and determine the neighborhood's best solution, which will replace the others.

In the image below is presented the pseudocode of a Tchebycheff MOEA/D [2].

Algorithm 2. The MOEA/D general framework

Input:

- MOP
- the number of the sub-problems considered in MOEA/D, N
- a uniform spread of N weight vectors: $\lambda^1, \ldots, \lambda^N$
- the number of the weight vectors in the neighborhood of each weight vector, T
- the maximum number of generations, gen_{max}

Output:

- EP

Step 0 - Setup:

- Set EP = ∅
- $gen = 0$

Step 1 – Initialization

- Uniformly randomly generate an initial internal population, $IP_0 = \{x^1, \ldots, x^N\}$ and set $FV^i = F(x^i)$.
- Initialize $z = (z_1, \ldots, z_n)^T$ by a problem-specific method.
- Compute the Euclidean distances between any two weight vectors and then work out the T closest weight vectors to each weight vector. $\forall i = 1, \ldots, N$, set $B(i) = \{i_1, \ldots, i_T\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.

Step 2 – Update: For $i = 1, \ldots, N$

- Genetic operators: Randomly select two indexes k, l from $B(i)$, and then generate a new solution y from x^k and x^l by using genetic operators.
- Update of z , $\forall j = 1, \ldots, n$, if $z_j < f_j(y)$, then set $z_j = f_j(y)$.
- Update of Neighboring Solutions: For each index $j \in B(i)$, if $g^{te}(y|\lambda^j, z) \leq g^{te}(x^j|\lambda^j, z^*)$, then set $x^j = y$ and $FV^j = F(y^j)$.
- Update of EP: Remove from EP all the vectors dominated by $F(y)$.
  Add $F(y)$ to EP if no vector in EP dominate $F(y)$.

Step 3 – Stopping criteria

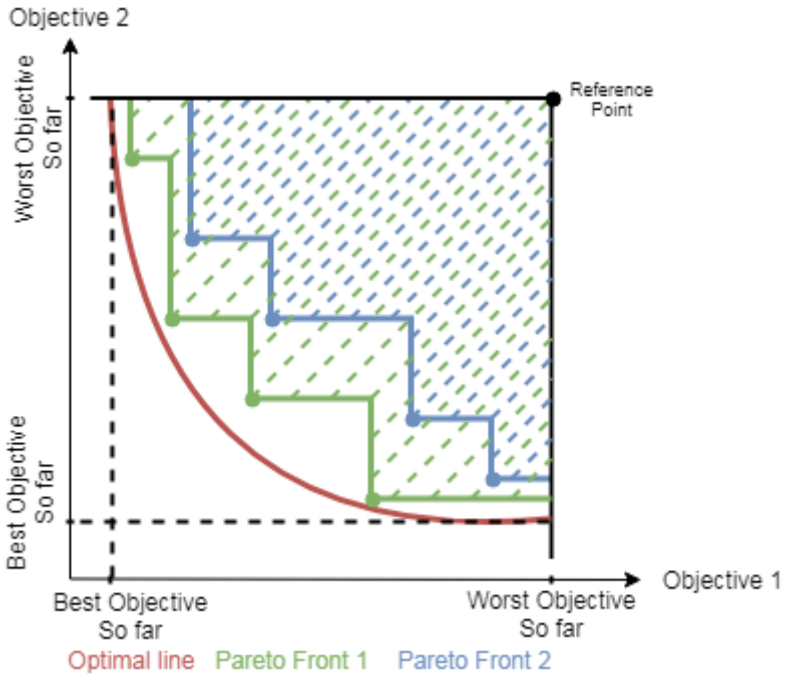
- If $gen = gen_{max}$, then stop and output EP, otherwise $gen = gen + 1$, go to Step 2.

MOEA/D pseudocode

An interesting thing to see is the presence of an external archive EP, which has the task to maintain at each iteration the non-dominated solutions that we have find from the start of the execution. This is a very important structure, because without an external archive would be impossible to keep an optimal solution generated during an early iteration. EP is also the output of the algorithm, so long as it contains all the non-dominated solutions. Is it possible both to keep EP unbounded or make it bounded, in this case we have to design a policy in order to decide which solutions discard when we go out of the bound.

The most popular choice as stopping criterion is to fix a number of iterations gen_{max} and to stop the algorithm when gen_{max} iteration are done. It's even so possible to adopt another stopping criterion, like a certain number of consecutive iterations that does not find a non-dominated solution, but it goes out from the scope of our project.

There are many ways to compare different execution of the MOEA/D, one of them is to measure the **hypervolume** of the pareto front with respect to a reference point, for example the point having as coordinates the highest values in the various objective function of the algorithm. In that case, a greater hypervolume means a better front, since the volume dominated by the front is bigger. For clarity we have an image below[3].



So, it's clear that MOEA/D it's a very efficient algorithm to use when we want to resolve a Multi-objective Optimization Problem and that's why it's an optimal choice to combine it with ACO in order to make the latter a Multi-objective Optimization Problem.

4. MOEA/D-ACO

The result of this project is the fusion between the two algorithms just mentioned, the MOEA/D and the ACO algorithm. Our implementation starts from the code of the ACO algorithm, visible in *https://it.mathworks.com/matlabcentral/fileexchange/69028-ant-colony-optimiztion-aco?s_tid=srchtitle* and we reached the goal mentioned above looking at the implementation in the following paper **"*MOEA/D-ACO: A Multiobjective Evolutionary Algorithm Using Decomposition and Ant Colony*" Liangjun Ke, Qingfu Zhang, Senior Member, IEEE, and Roberto Battiti, Fellow, IEEE (2013).**

4.1 Algorithm

For what concern our implementation, the choice of the parameters is the first thing we have in the code, where we can see four parameters:

- **N**: The number of ants
- **K**: The number of groups in which the ants are divided
- **T**: The dimension of the neighborhood for each ant
- **m:** The number of objective functions

In particular, we chose as objective functions the distance and the cost between two nodes. In this way we tried to represent a more realistic situation like in the TSP problem, assigning at each edge a length and a cost (like in the case of toll for a certain street).

The second step is the initialization of the lambda vectors (weight vectors) and then we set the groups, dividing the lambda vectors into K groups. The lambda vectors are all the weight vectors in which each individual weight component takes a value from $\left\{\frac{0}{H}, \frac{1}{H}, \dots, \frac{H}{H}\right\}$, where $\mathrm{H} = \mathrm{N} - 1$.

To generate these groups, we first generate the representative weight vectors $\xi_1, \dots, \xi_K$ in the same way we generate the lambda vectors, using $\mathrm{H} = \mathrm{K} - 1$. Then we clustered the lambda vectors $\lambda_1, \dots, \lambda_N$ into K groups associating in the group *i* all the lambda vectors for which the ξ_i is the closest representative vector.

The next step is the creation of the graph, with the two values of the objective functions on each edge.

Then we took in exam the initialization of the heuristic information and of the pheromone matrix. For what concern the heuristic information, we initialized it using the following formula:

$$\eta_{k,l}^{i} = \frac{1}{\sum_{j=1}^{m} \lambda_j^i c_{k,l}^j}$$

where k and *l* are the two nodes connected by the edge and *i* is the ant.

The value of the pheromone is instead related to each group and not to each ant, and each $\tau_{k,l}^{i}$ is set to 1, where *k* and *l* are the two nodes connected by the edge and *i* is the number of the group.

Then, we analyzed the **solution construction** phase, that is the construction of the tour for a specific ant. Before it starts, some useful parameters are set, and some data structures are initialized, like the replacingSolution that we will use at the end of each iteration in the step of the update of the solution, and we initialized a first random tour for each ant. Finally, we set the neighborhood for each ant, so that the neighbors of the ant are the ones with the closer lambda vector.

After this initialization phase, the real solution construction starts, where we have a certain number of iterations *maxIter* during which we perform the following steps:

- Update of the phi matrix
- Update of the tour of each ant
- Calculation of the fitness for each ant
- Update of the EP (An external archive for the solutions)
- Update of the pheromone matrix
- Update of the solutions

Assuming that ant *i* is in group *j* and its current solution is $x^i = \left(x_1^i, \dots, x_n^i\right)$, the ant i constructs its new solution $y^i = \left(y_1^i, \dots, y_n^i\right)$, following the path explained in the bulled point list visible above.

The first step is the update of the phi matrix, which represents a combination of the values of the pheromone matrix and of the heuristic information matrix, this combination has been done in the following way, for k,l = 1,…,n nodes we set:

$$\phi_{k,l} = [\tau_{k,l}^{i} + \Delta * \ In(x^i, (k,l))]^{\alpha} (\eta_{k,l}^{i})^{\beta}$$

where Δ is a parameter and $In\left(x^i, (k,l)\right)$ is the indicator function, an integer equal to 0 if the link (k,l) is not in the path of the tour x^i, 1 otherwise.

Then each ant uses the information generated in the phi matrix to build its own tour. In the function *createColony* each ant chooses a random initial node, and then use the phi matrix to choose the next node to visit. In particular, a random value is extracted, if it is lower than the fixed threshold r (passed as parameter to this function) than the ant will choose between the nodes that are not already visited, the one which edge connected with the current node has the greatest phi value, otherwise, it performs the roulette wheel selection according to a certain probability of this type:

$$\frac{\phi_{k,l}}{\sum_{s \in C} \phi_{s,l}}$$

where C is the set of unvisited cities for the ant.

Once visited all the nodes, we add as last node the first node in order to close the circle and terminate the tour.

The next step is the calculation of the fitness for the tour just constructed by each ant. The fitness function is the following:

```
function [ fitness ] = fitnessFunction ( tour , graph)


fitnessDist = 0;
fitnessCost = 0;

for i = 1 : length(tour) -1

    currentNode = tour(i);
    nextNode = tour(i+1);


    fitnessDist = fitnessDist + graph.edges( currentNode ,  nextNode,1 );
    fitnessCost = fitnessCost + graph.edges( currentNode ,  nextNode,2 );

    fitness=[fitnessDist,fitnessCost];

end


end
```

Then, we have to update EP with the aim to keep in it all the non-dominated solutions. For each tour constructed at the current iteration we first check if the fitness related to it is dominated from at least one solution in the EP, if so we go on with the tour of the next ant. Otherwise, we insert this solution into EP, discarding each EP solution dominated by this new one. We decided to keep the archive unbounded, like in the reference paper.

The next step is the update of the pheromone matrix, this update is divided into two parts, like mentioned for Ant System algorithm:

- The evaporation of the pheromone, which is computed in the following way:

  $$\tau_{k,l}^{j} = \rho * \tau_{k,l}^{j}$$

  where j is the group and (k,l) is the link.

  We can note the difference between this formula and the one in the Ant System algorithm, because the concept of ρ is different, here it is the persistence (of the pheromone) rate, instead in AS it was the evaporation rate.

- The release of the pheromone, computed in this way:

$$\tau_{k,l}^{j} = \sum_{x \in \Pi} \frac{1}{g(x|\lambda^{j})}$$

where Π is the set of all the new solutions that:

a) were constructed by the ants in group j in the current iteration,

b) were just added to EP,

c) contain the link between the cities k and l.

- Then, we calculate ($tau \min e\ taumax$) according to the following formulas:

$$\tau_{max} = \frac{(B+1)}{(1-\rho)g_{min}}$$

$$\tau_{min} = \varepsilon \tau_{max}$$

where ε is a parameter, B is the number of non-dominated solutions found in this last iteration and g_{min} is the lowest value obtained for the objectives functions in all the N subproblems.

- Then, we check if there is some value in the pheromone matrix that does not respect the interval $[\tau_{min}, \tau_{max}]$, in those case we substitute the values smaller than $\tau_{min}$with $\tau_{min}$and the values greater than $\tau_{max}$with τ_{max}.

The last step is the update of the solution, in which we replace the solution of each ant with the best solution in its own neighborhood that was not used to replace any other old solution. To choose the best one we check the value $g^{ws}(x|\lambda) = \sum_{i=1}^{m} \lambda_i f_i(x)$ and we choose the minimum one, and for understand if the solution has already been used, we check the data structure *replacingSolution*, which contains the solution already used to replace other old solution.

At the end of all the iterations the EP will contain all the non-dominated solutions that has been found during the execution and we conclude printing them on a graph, showing the Pareto Front and computing the hypervolume for this set of solutions.

4.2 Functions

In the following chapter we will analyze more in the detail the functions used in our implementation, that we just described.

The list of the functions called in the main part of the implementation, visible in *moead_aco.m* is the following:

- *createColony.m*
- *createGraph.m*
- *initializedTour.m*

- *findBestNeighborhood.m*
- *findGMin.m*
- *fitnessFunction.m*
- *indicatorFunction.m*
- *PlotCosts.m*
- *rouletteWheel.m*
- *updateEP.m*
- *updatePhromone.m*
- *hypervolume.m*

Function: [colony] = createColony(nodeNo, colony , antNo, r, phi)

Input:

- *nodeNo*: number of nodes in the graph.
- *colony*: all the colony of ants to which the tour has to be constructed. From this structure we can obtain the tour, and the fitness for each ant.
- *antNo*: number of ants
- *r*: control parameter for the probability of choosing the next node.
- *phi*: the phi matrix, necessary to gives weight to the edge and understand, according to some probability related to it, to which city goes next.

Output:

- *colony*: the modified colony with the new tour for each ant.

This function constructs the tour for each ant in the colony.

Function: [graph] = createGraph()

Output:

- the graph randomly generated, with the two costs (distance and price) on each edge

---

Function: [colony] = initializedTour(nodeNo, colony , antNo)

Input:

- *nodeNo*: number of nodes in the graph.
- *colony*: all the colony of ants to which the tour has to be constructed. From this structure we can obtain the tour, and its fitness, of each ant.
- *antNo*: number of ants

Output:

- *colony*: the modified colony with the new tour for each ant.

This function does the same operation than the createColony, but with a random tour initialized for each ant.

Function: [colony, replacingSolution] = findBestNeighborhoodTour(colony,graph,antIndex, neighborhood, replacingSolution,T,lambda_weights)

Input:

- *colony*: the colony of the ants
- *graph*: the total graph of nodes and edges
- *antIndex*: index of the ant to which we are computing the best neighborhood tour
- *neighborhood*: matrix (numberOfAnt x T(dimension of neighborhood)) in which, for each ant, we have a vector with the indexes of the other ants in its own neighborhood.
- *replacingSolution*: structure that contains all the solution that in the past has already replace some other solution. To avoid that a solution that has already replace some other old solution can again relace a new solution.
- *T:* the dimension of the neighborhood
- *lambda_weights*: all the lambda_weights generated at the beginning.

Output:

- *colony*: the colony with the replaced tour.
- *replacingSolution*: the structure mentioned before after modification.

In the function above, we substitute the tour of each ant with the best solution in its neighborhood, that was not used to replace any other old solution.

Function: gmin = findGMin(colony,lambda_weights,startAnt,endAnt)

Input:

- *colony*: the colony of the ants.
- *lambda_weights*: all the lambda_weights generated at the beginning.
- *startAnt*: index of the ant to which start the interval taken in exam by this function.
- *endAnt*: index of the ant to which end the interval taken in exam by this function.

Output:

- gmin: The smallest $g^{ws}(x|\lambda)$ between the ants in the specified interval.

The findGMin function finds the smallest value of $g^{ws}(x|\lambda)$ between a certain specified interval of ants. This function is also used during the implementation to compute the $g^{ws}(x|\lambda)$ of a single ant, specifying ad startAnt and endAnt the same value, obviously belonging to the ant to which we want to compute the value.

Function: [fitness] = fitnessFunction (tour , graph)

Input:

- *tour:* the tour of a single ant is passed as parameter.
- *graph:* the complete graph.

Output:

- *fitness:* the fitness of the specified tour, as a couple of value for the two objective functions.

This function simply computes the fitness for a specified tour.

Function: result= indicatorFunction(tour,node1,node2)

Input:

- *tour:* the tour of a single ant is passed as parameter.
- *node1:* the first node of the edge taken in exam.
- *node2:* the second node of the edge taken in exam.

Output:

- result: an integer equal to 0 if the link (*node1, node2*) is not in the path *tour*, 1 otherwise.

Function: PlotCosts(EP)

Input:

- EP: All the non-dominated solutions saved so far.

It takes all the non-dominated solutions saved so far and print it on a graph

---

Function: [nextNode] = rouletteWheel(P)

Input:

- P: set of normalized probabilities.

Output:

nextNode: the node chosen by the rouletteWheel.

This function takes a set of probabilities, compute a cumulative sum, extract a random value and finds the first index where the random number extracted is less than or equal to the cumulative probability and return it as the next node of the tour.

Function: [new_sol_EP,ant_best_tour,EP] = updateEP(colony,EP)

Input:

- colony: the colony of the ants.
- EP: all the non-dominated solutions saved so far.

Output:

- new_sol_EP: the solution that entered in EP in this iteration.
- ant_best_tour: the index of the ants whose tours are in new_sol_EP.
- EP: the updated EP.

This function checks the last tour of each ant, inserting the non-dominated new solutions in EP and removing from it the solutions dominated by the inserted ones.

Function: [taumax,taumin,tau] = updatePhromone(tau, colony,new_sol_EP,ant_best_tour,rho,groupAnt,lambda_weights,eps)

Input:

- tau: the pheromone matrix.
- colony: the colony of the ants.
- new_sol_EP: the solution that entered in EP in this iteration.
- ant_best_tour: the index of the ants whose tours are in new_sol_EP.
- rho: the persistence rate.
- groupAnt: structure that relates each ant to its group.
- lambda_weights: all the lambda_weights generated at the beginning.
- eps: a control parameter set at the beginning.

Output:

- taumax: the maximum possible value for tau.
- taumin: the minimum possible value for tau.
- tau: the updated pheromone matrix.

This function updates the values on the pheromone matrix and the interval [taumin, taumax].

Function: v=hypervolume(P,r,N)

Input:

- P: Pareto Front.
- r: reference point.
- N (optional): number of uniformly distributed points for the evaluation, 1000 by default.

Output:

- v: the computed hypervolume.

5. Test

For what regards the test of the application, we decided to report for each execution the **number of solutions**, the **execution time** and the **hypervolume**. We analyzed two scenarios:

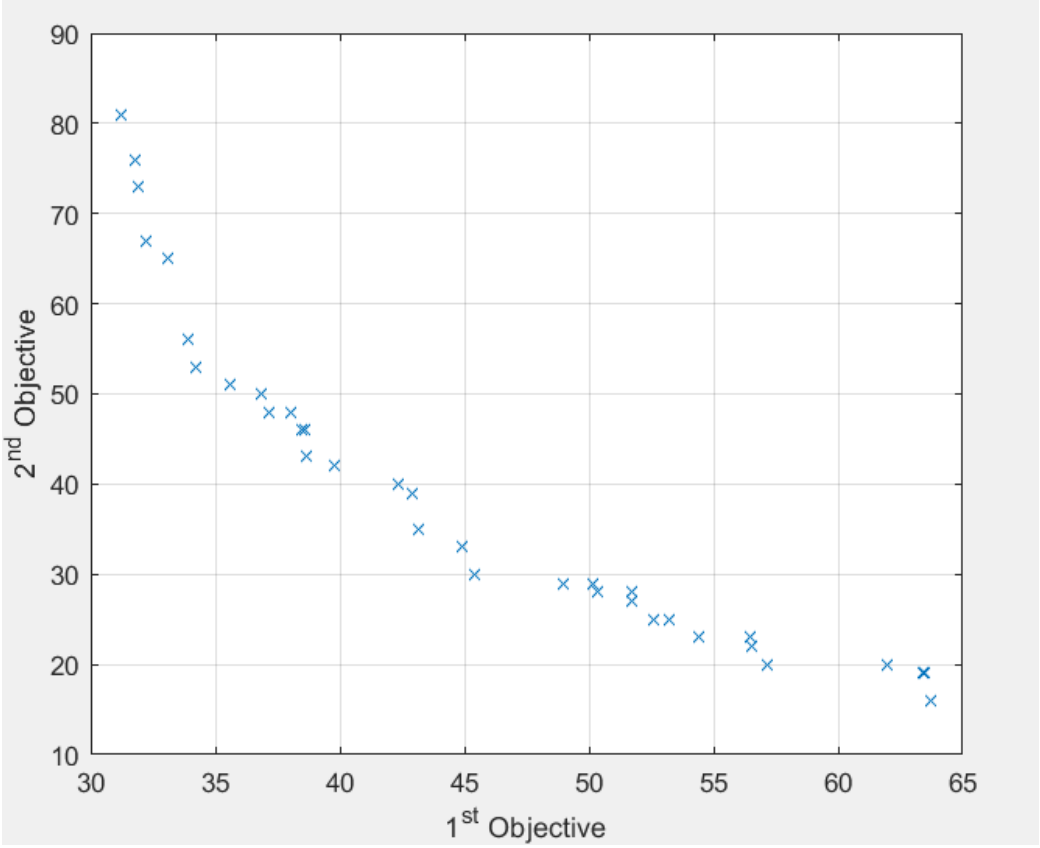
- Scenario 1: 14 ants, 2 groups, 5 ants for neighborhood
- Scenario 2: 50 ants, 4 groups, 11 ants for neighborhood

In every scenario we considered 3 values of β (1, 2, 5) and 3 values of ρ (0.5, 0.7, 0.9), since in literature we saw they are the most variable values. We will report also the pictures of the various Pareto front, where the 1st objective is the total distance covered by the tour and the 2nd objective is the cumulative cost of the edges that compose the tour.

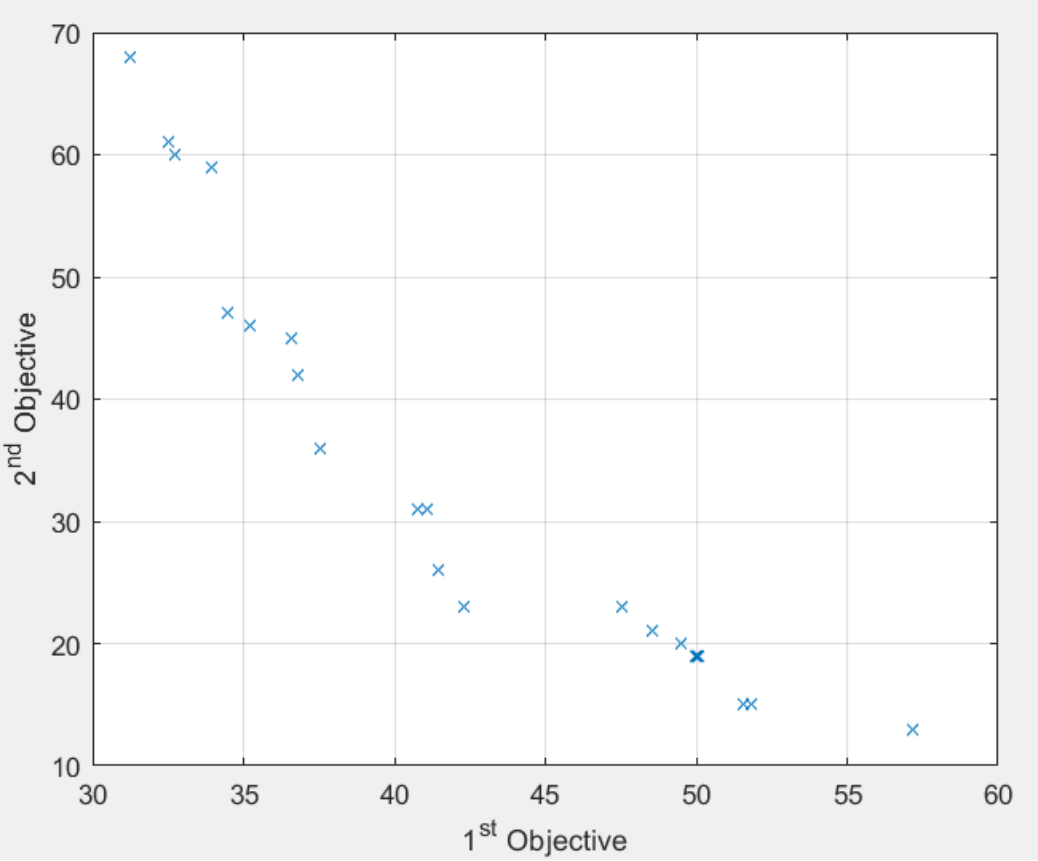
5.1 Scenario 1

β \ ρ	0.5	0.7	0,9
1	48 solutions 48370 ms 0.299 hypervolume	31 solutions 50060 ms 0.258 hypervolume	20 solutions 46860 ms 0.267 hypervolume
2	57 solutions 43280 ms 0.173 hypervolume	39 solutions 43920 ms 0.249 hypervolume	71 solutions 43770 ms 0.193 hypervolume
5	94 solutions 42990 ms 0.333 hypervolume	83 solutions 42380 ms 0.269 hypervolume	62 solutions 43550 ms 0.300 hypervolume

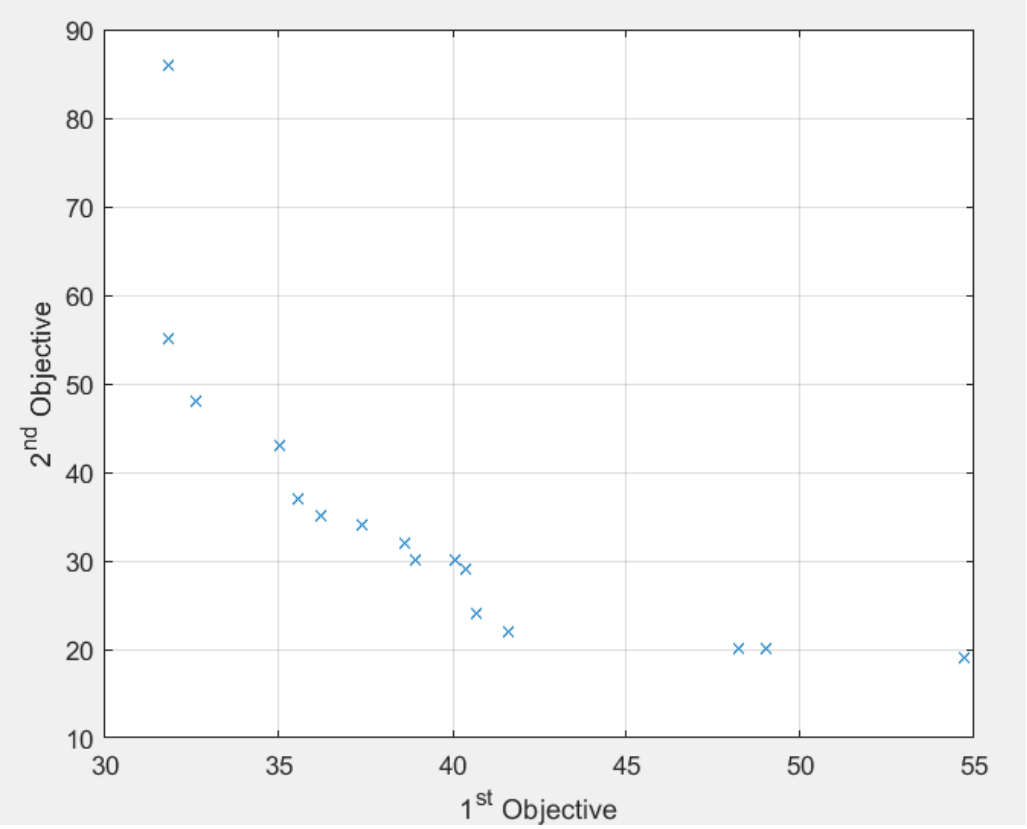
- β=1, ρ=0.5


90
80
70
60
50
40
30
20
10
30
35
40
45
50
55
60
65
2nd Objective
1st Objective

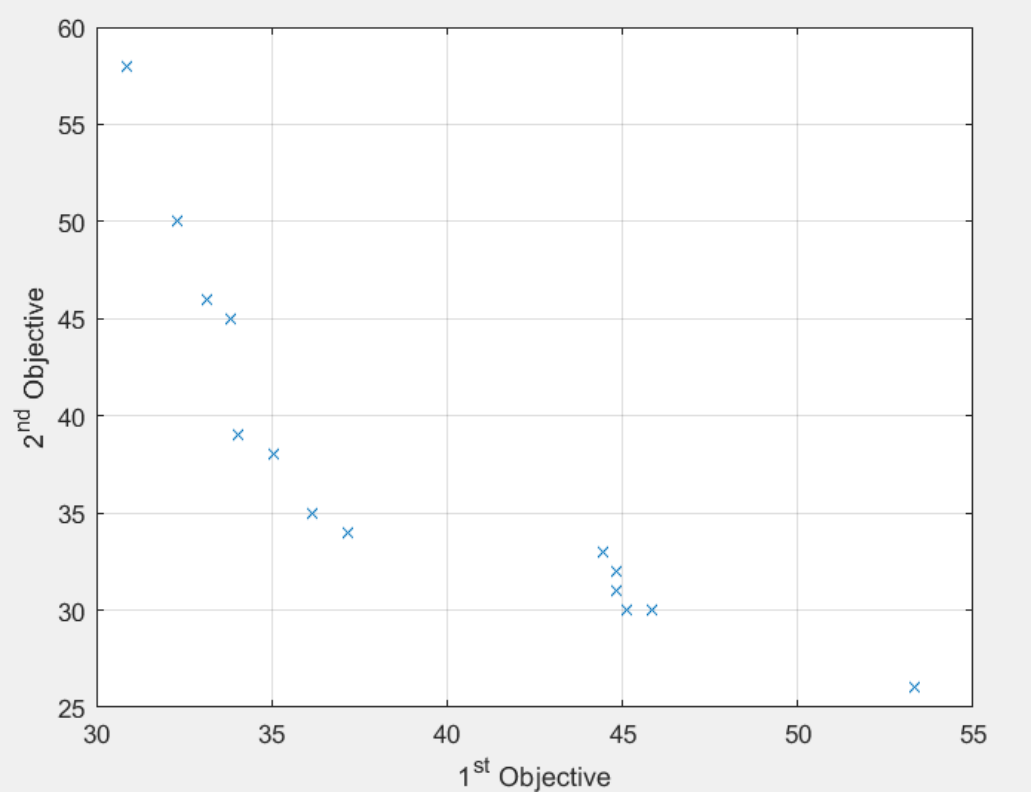
- β=1, ρ=0.7


70
60
50
40
30
20
10
30
35
40
45
50
55
60
2nd Objective
1st Objective

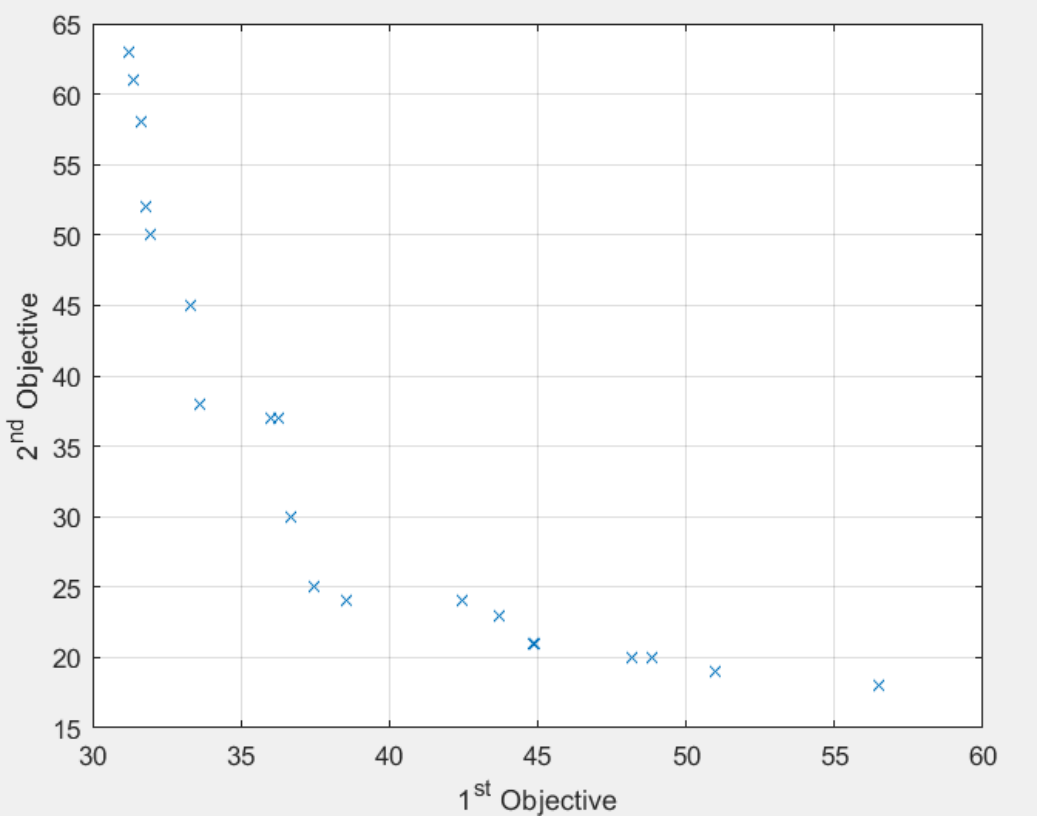
- β=1, ρ=0.9


90
80
70
60
50
40
30
20
10
2nd Objective
30
35
40
45
50
55
1st Objective

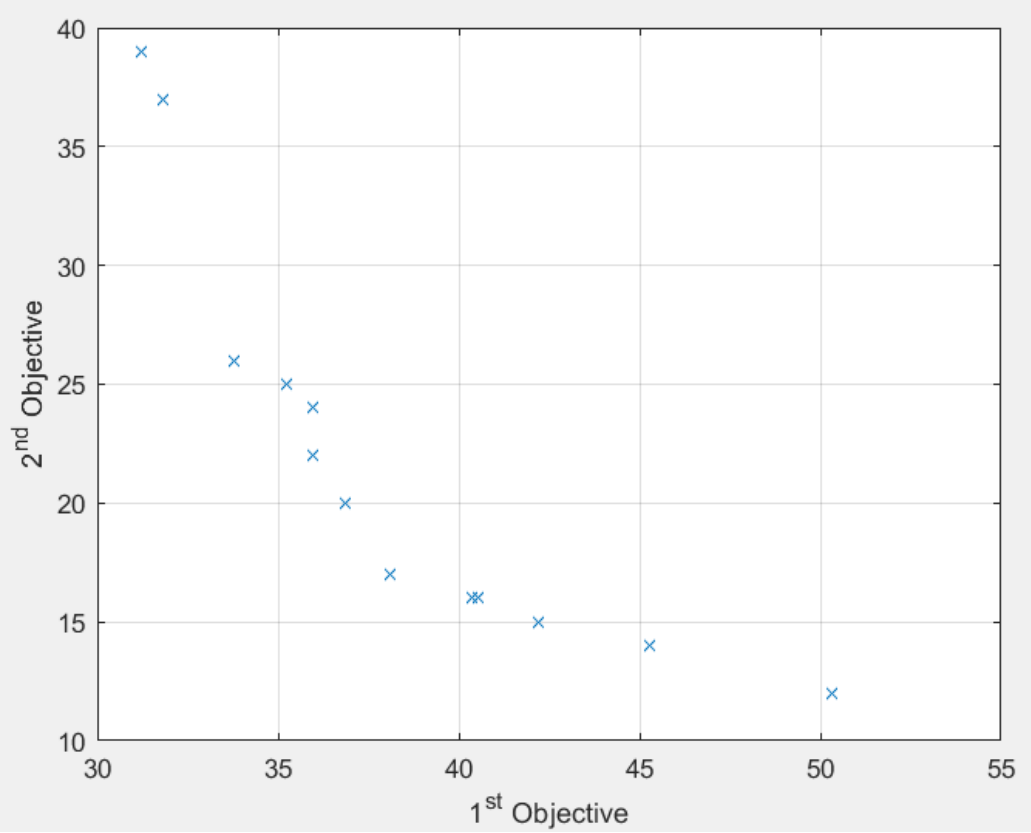
- β=2, ρ=0.5


60
55
50
45
40
35
30
25
2nd Objective
30
35
40
45
50
55
1st Objective

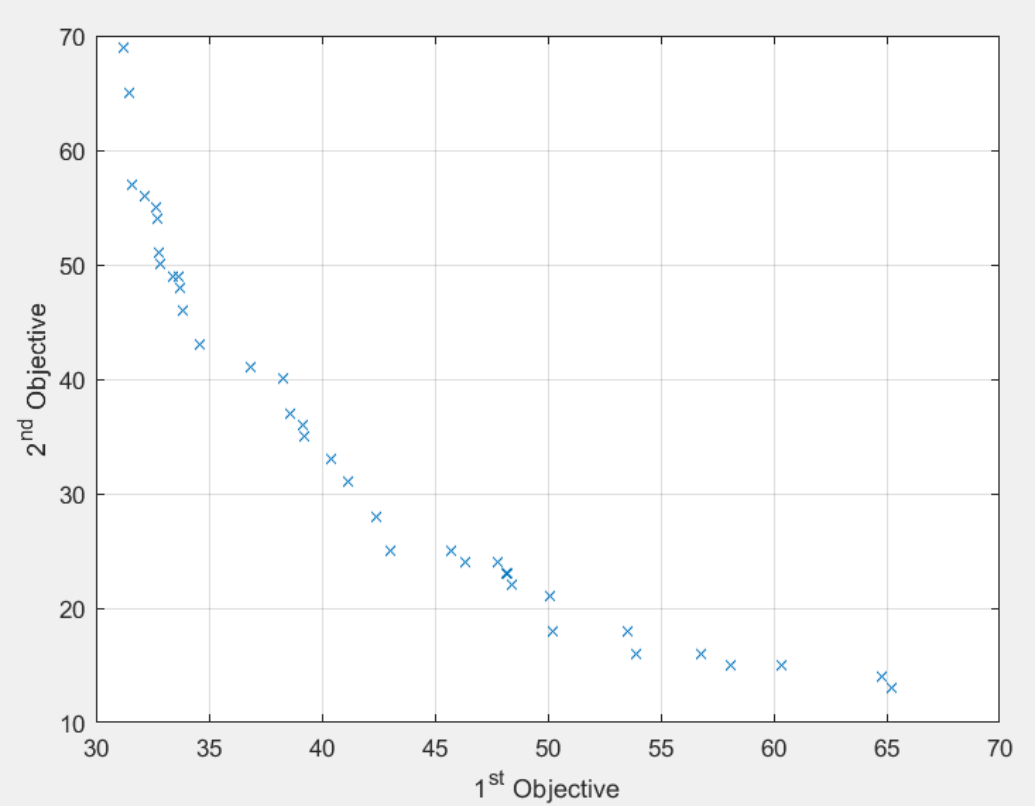
- β=2, ρ=0.7


65
60
55
50
45
40
35
30
25
20
15
30
35
40
45
50
55
60
2nd Objective
1st Objective

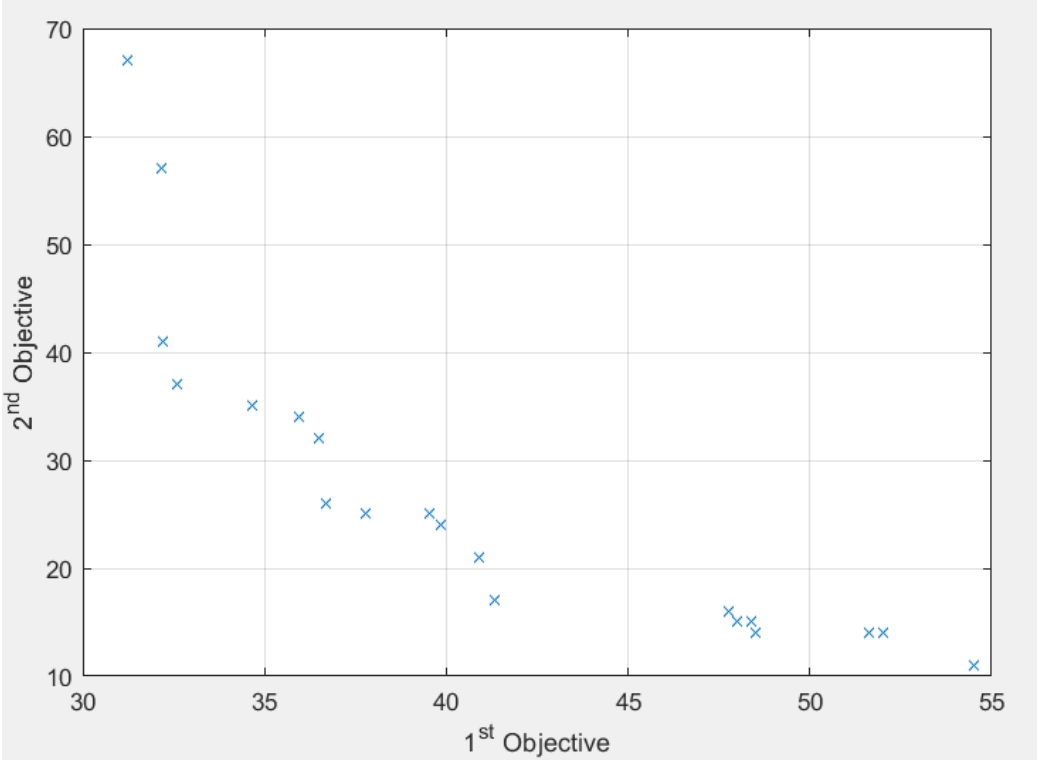
- β=2, ρ=0.9


40
35
30
25
20
15
10
30
35
40
45
50
55
2nd Objective
1st Objective

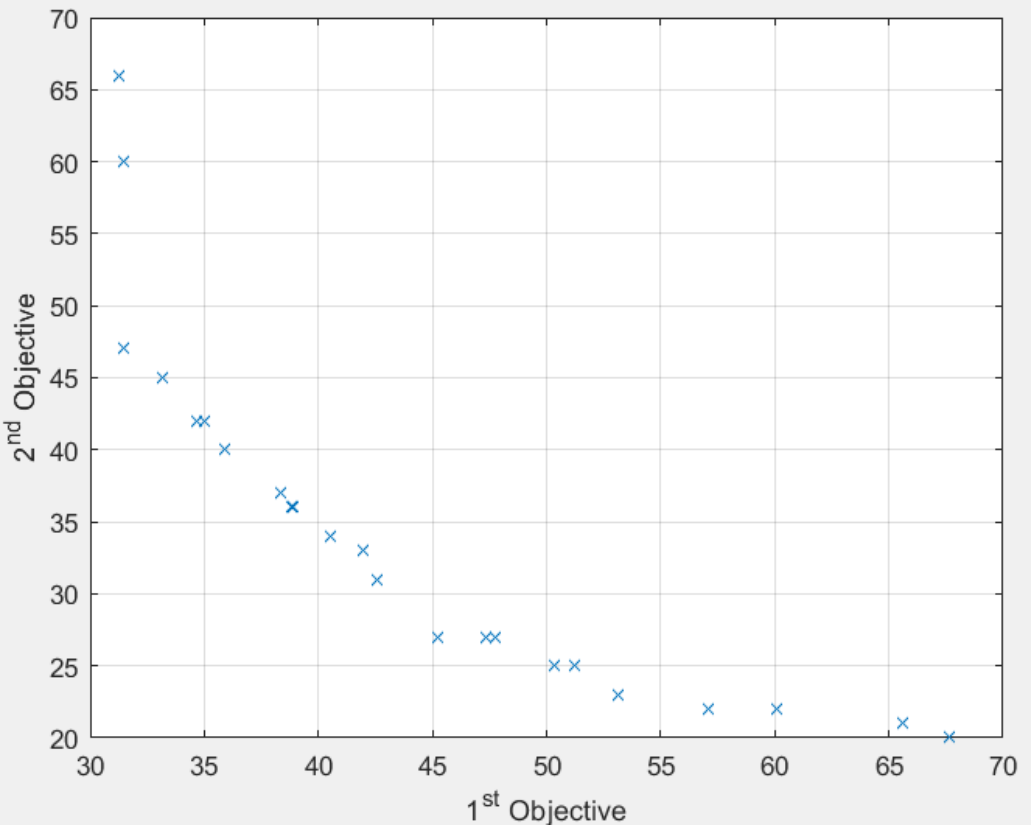
- β=5, ρ=0.5


70
60
50
40
30
20
10
30
35
40
45
50
55
60
65
70
2nd Objective
1st Objective

- β=5, ρ=0.7


70
60
50
40
30
20
10
30
35
40
45
50
55
2nd Objective
1st Objective

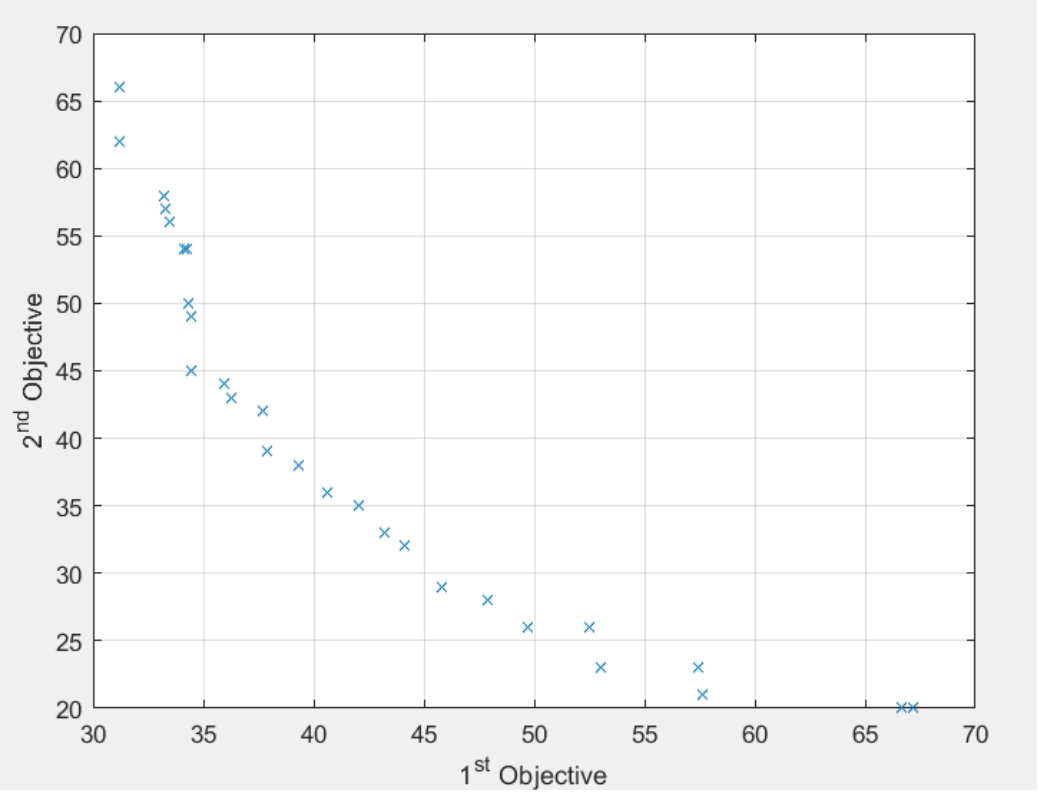
- β=5, ρ=0.9



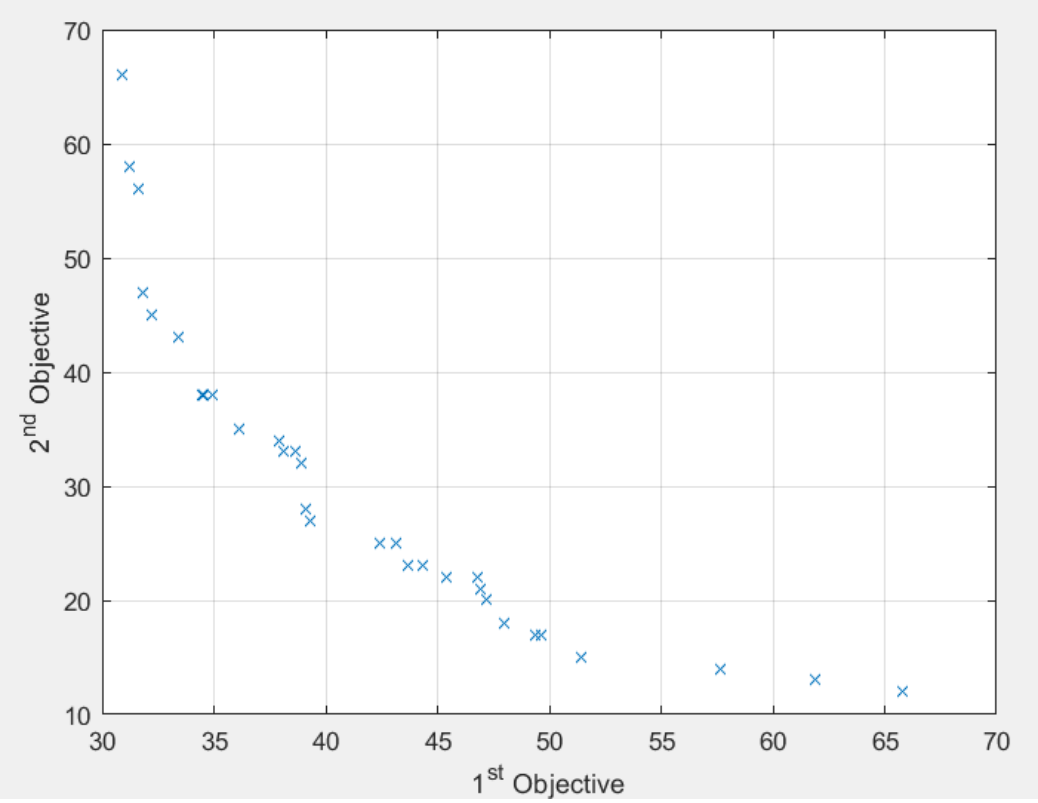
5.2 Scenario 2

β \ ρ	0.5	0.7	0,9
1	63 solutions 431530 ms 0.267 hypervolume	88 solutions 438410 ms 0.344 hypervolume	48 solutions 396850 ms 0.235 hypervolume
2	125 solutions 434700 ms 0.281 hypervolume	91 solutions 526310 ms 0.310 hypervolume	72 solutions 444150 ms 0.317 hypervolume
5	116 solutions 386680 ms 0.218 hypervolume	145 solutions 389020 ms 0.309 hypervolume	149 solutions 378730 ms 0.273 hypervolume

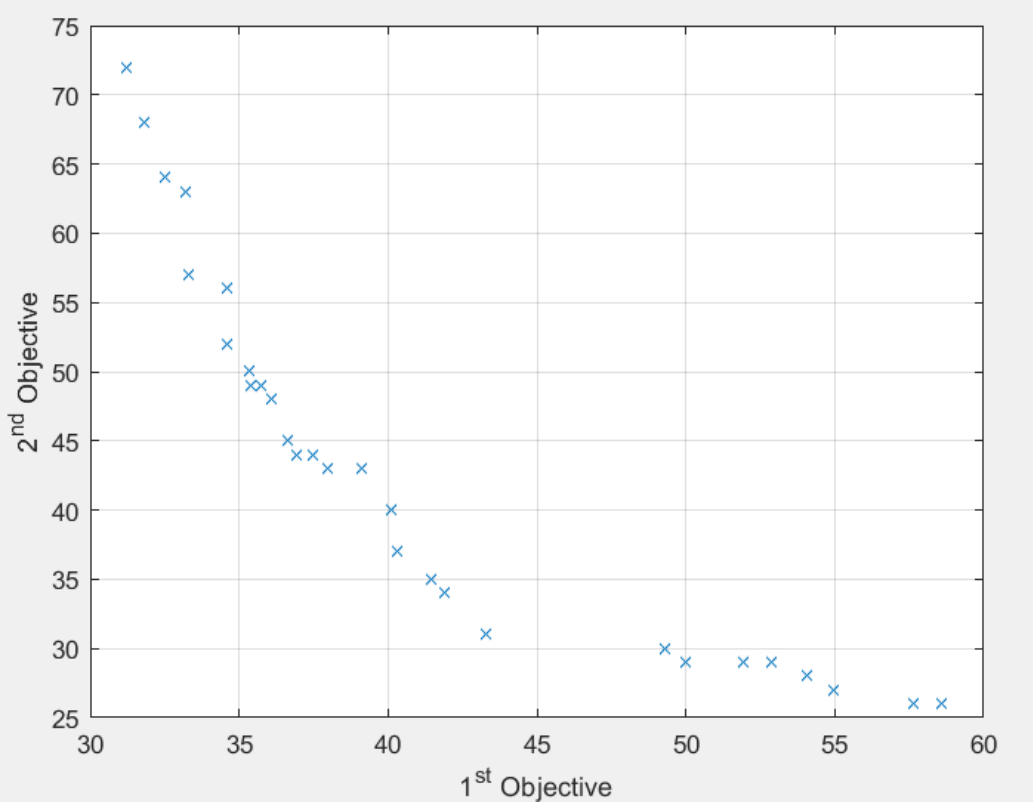
- β=1, ρ=0.5


70
65
60
55
50
45
40
35
30
25
20
2nd Objective
30
35
40
45
50
55
60
65
70
1st Objective

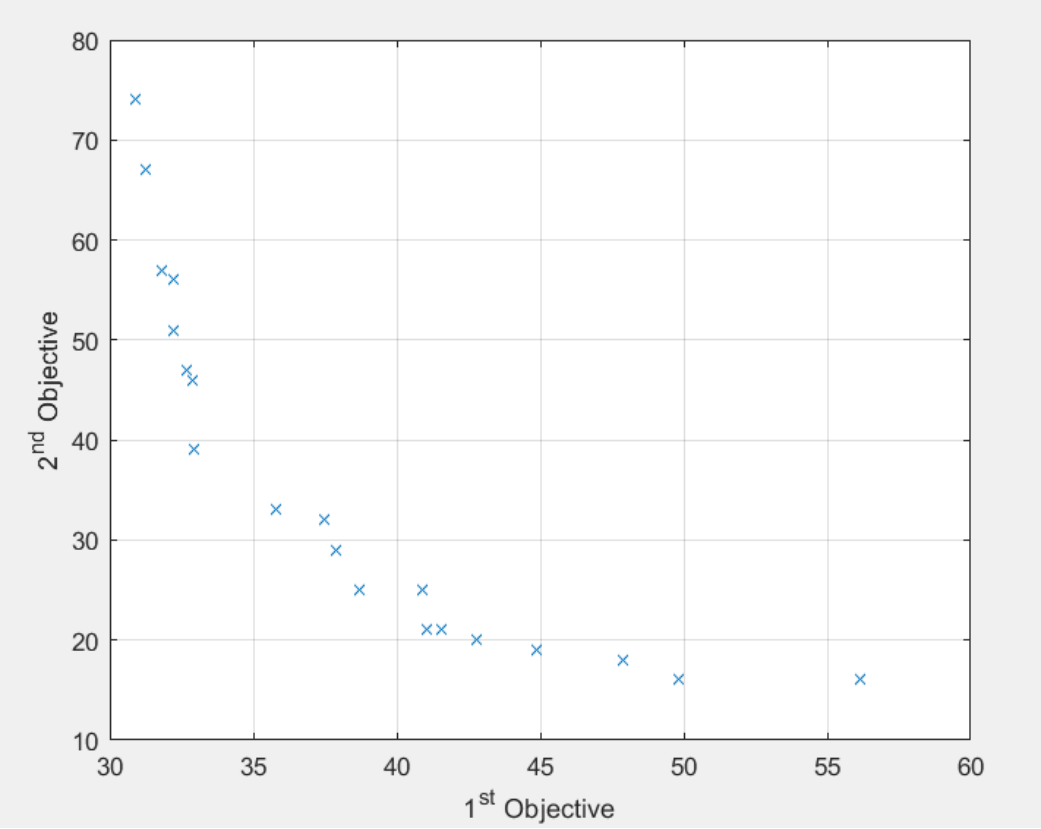
- β=1, ρ=0.7


70
60
50
40
30
20
10
2nd Objective
30
35
40
45
50
55
60
65
70
1st Objective

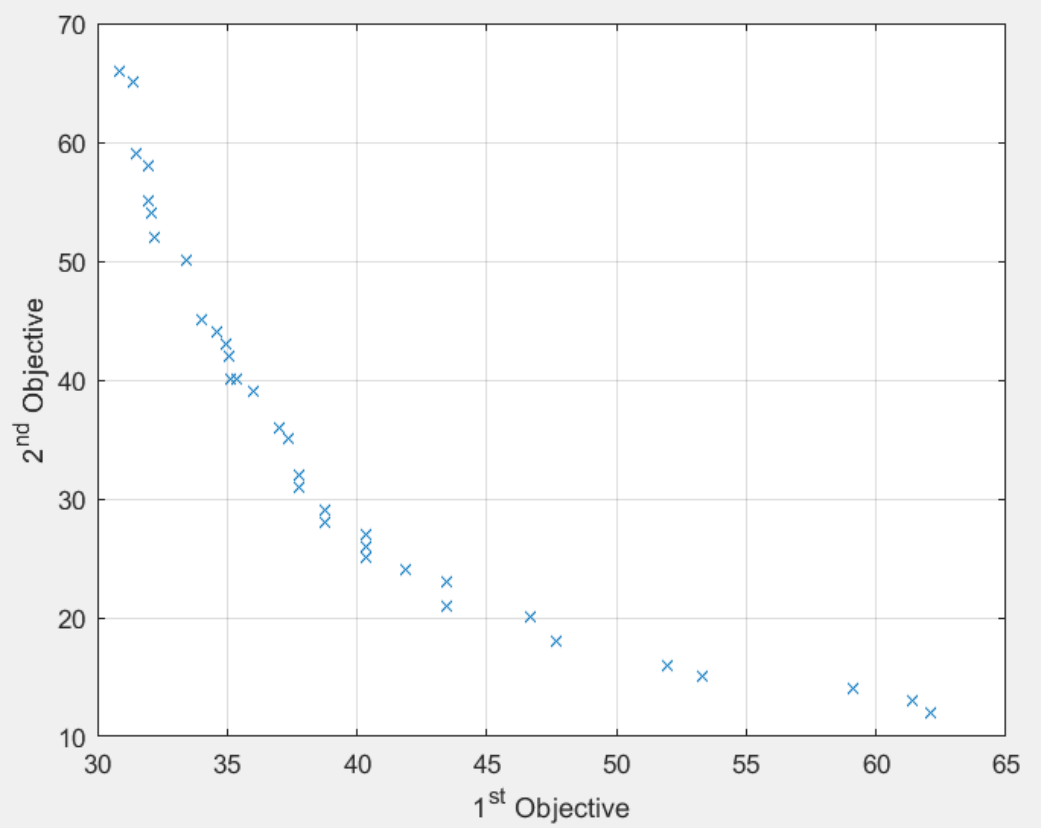
- β=1, ρ=0.9


75
70
65
60
55
50
45
40
35
30
25
30
35
40
45
50
55
60
2nd Objective
1st Objective

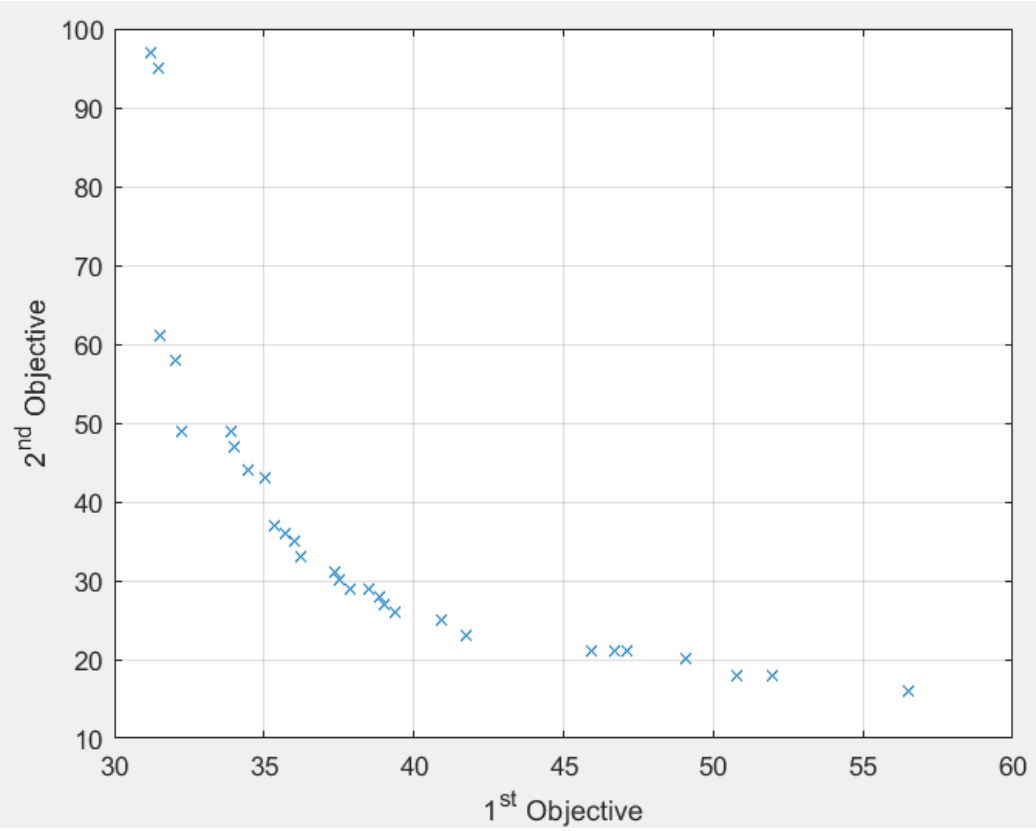
- β=2, ρ=0.5


80
70
60
50
40
30
20
10
30
35
40
45
50
55
60
2nd Objective
1st Objective

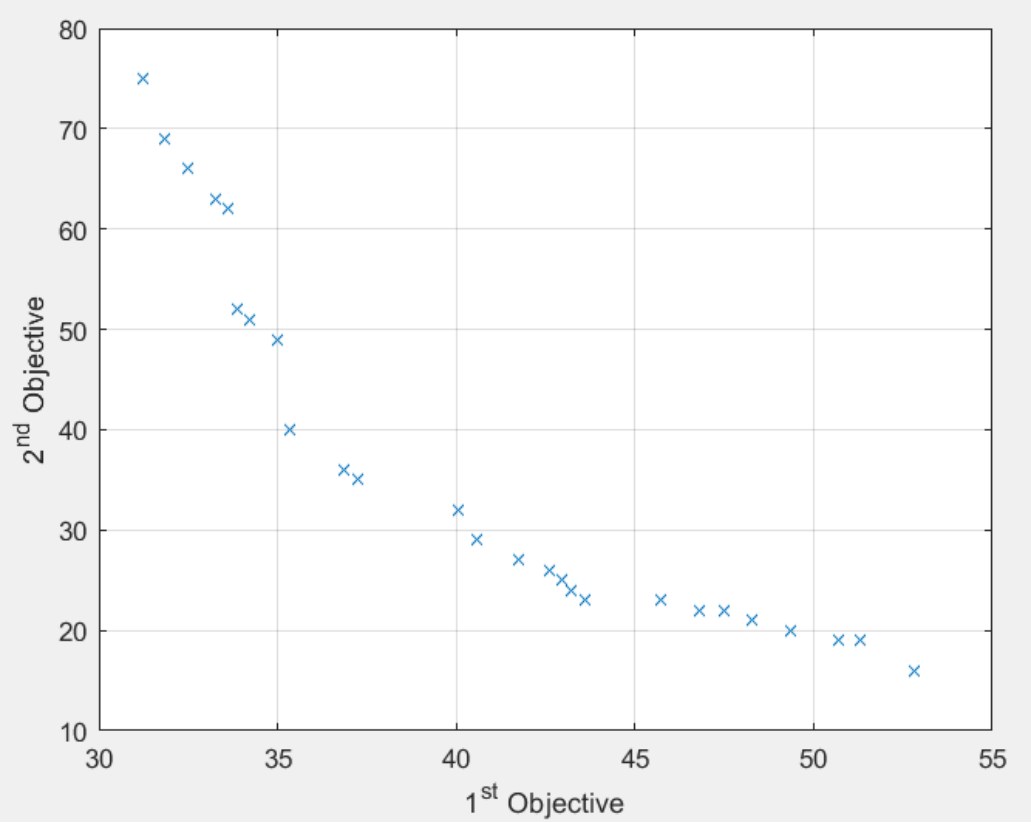
- β=2, ρ=0.7


70
60
50
40
30
20
10
30
35
40
45
50
55
60
65
2nd Objective
1st Objective

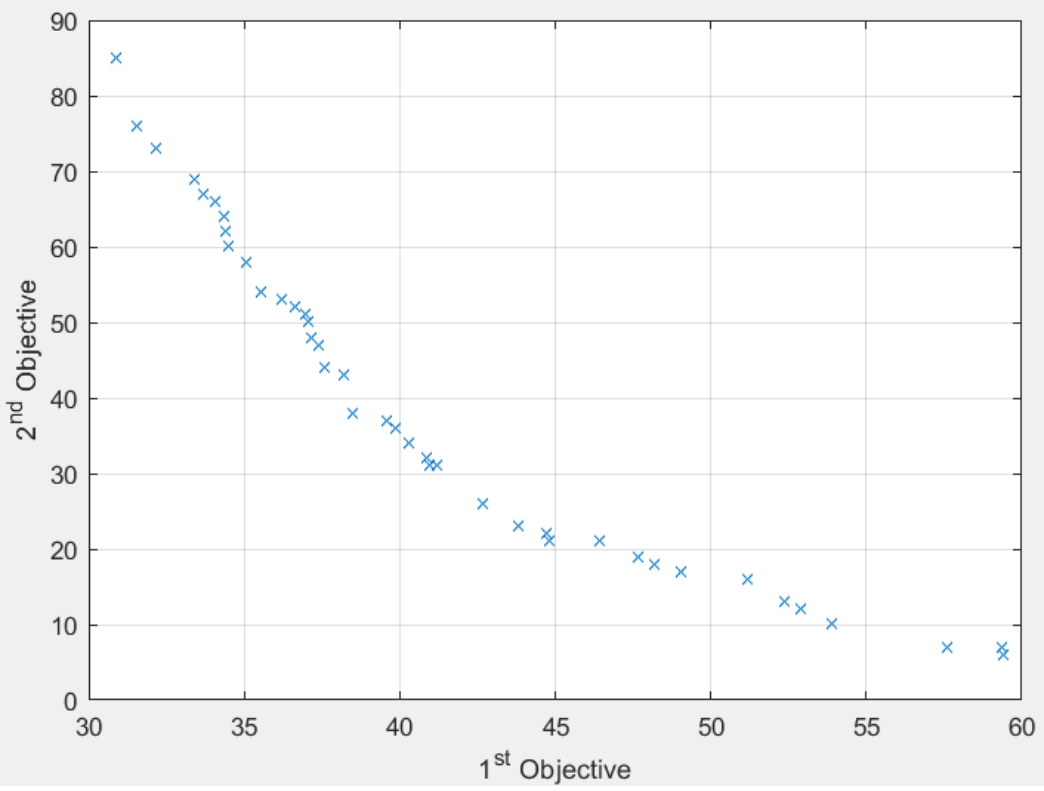
- β=2, ρ=0.9


100
90
80
70
60
50
40
30
20
10
30
35
40
45
50
55
60
2nd Objective
1st Objective

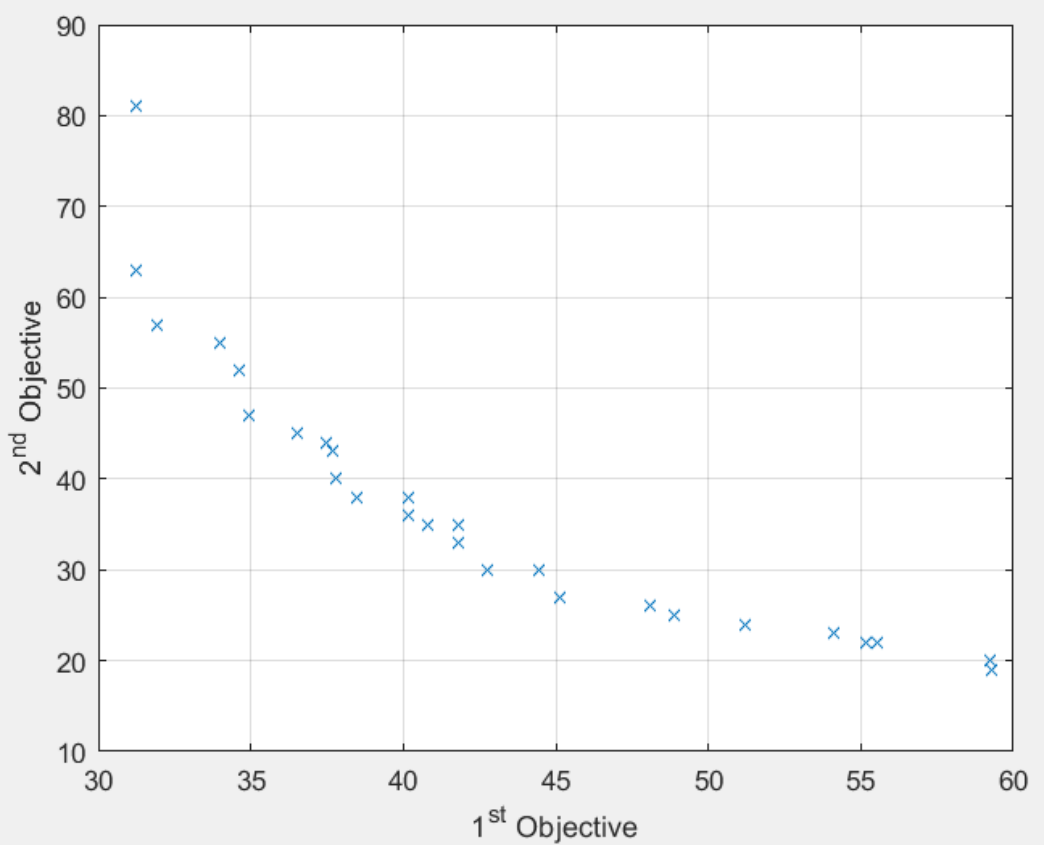
- β=5, ρ=0.5


80
70
60
50
40
30
20
10
30
35
40
45
50
55
2nd Objective
1st Objective

- β=5, ρ=0.7


90
80
70
60
50
40
30
20
10
0
30
35
40
45
50
55
60
2nd Objective
1st Objective

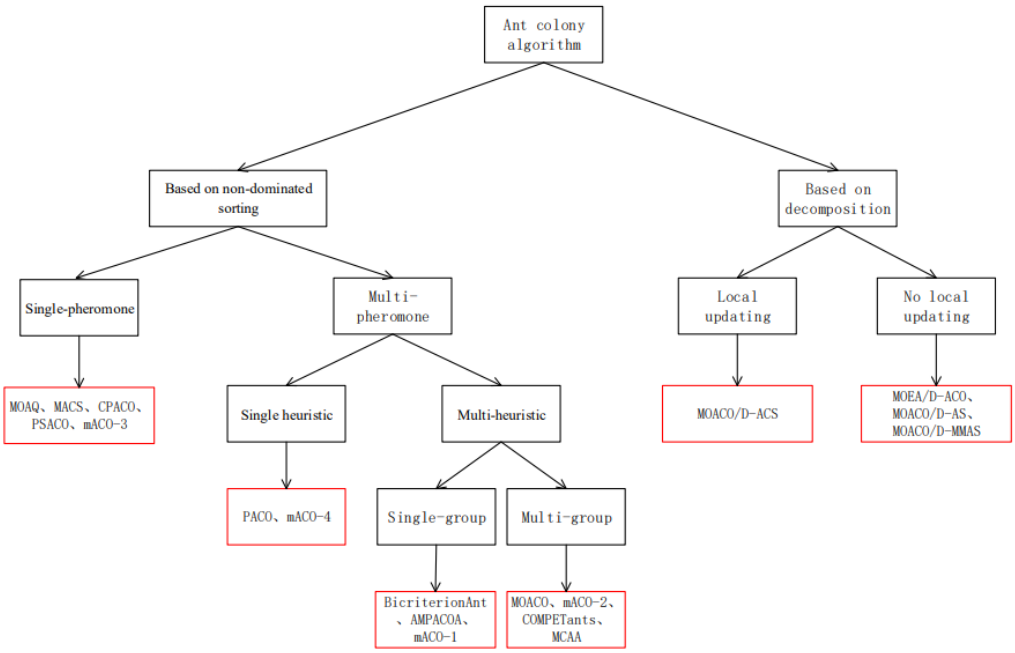
- β=5, ρ=0.9


90
80
70
60
50
40
30
20
10
2nd Objective
30
35
40
45
50
55
60
1st Objective

6. Other MOEA/D-ACO version

In recent years, many excellent multiobjective ant algorithms have been proposed like explained in the paper ***"Comparative Study of Ant Colony Algorithms for MO-Optimization"*** **Jiaxu Ning, Changsheng Zhang, Peng Sun and Yunfei Feng (2018)**.

In the above mentioned paper, a classification of the MO- Ant Colony Algorithms has been done, and we can see in the following tree the families and branches of the various algorithms.



As we can see, the first division is made between algorithms based on non-dominated sorting and algorithms based on decomposition. The former determines the solution set according to the domination relations of each solution constructed by each ant, the latter determines the solution decomposing a MO optimization problem into a number of single-objective optimization problems, and gives a weight vector to each sub-problem.

For all the leaves of the tree, at least one algorithm has been analyzed in the paper mentioned before.

From this first division, we identify our implementation as belonging to the second family of algorithm.

The second division, on the right part of the tree, is between algorithm which use local updating and algorithm which does not use it. In the local update way, the ants update the pheromone on the edge immediately after having crossed that edge, like in the **MOACO/D-ACS**, instead in the No local updating, the pheromone matrix is updated after all the ants have constructed their tours. So, we can

conclude that our version belongs to this category, how we can see from the position of **MOEA/D-ACO** in the tree above.

The algorithm based on **non-dominated sorting** are split into two subfamilies, the **single pheromone** and **multi-pheromone**. For the former, we analyze the **MOAQ**, this algorithm has some differences with our implementation, the first one is the usage of the pheromone matrix and of the heuristic information, this algorithm uses one pheromone matrix and two heuristic information matrices, the first one is the same for all the ants and each heuristic information matrix is related to one objective. This algorithm divides all the ants into two groups, one with weight vector {0,1} and the other with weight vector {1,0}. This means that each group uses only the information of a single heuristic matrix without aggregating the two heuristic information matrices of the two objectives.

The second type of non-dominated sorting algorithm are the **multi-Pheromone Matrix** algorithm, in which many pheromone matrices are used, and then, there is a successive division based on the number of heuristic matrices used. For the algorithm with a **single heuristic matrix**, the **PACO** algorithm is an example. At the opposite of the previous family of algorithm, this algorithm uses many pheromone matrices and one heuristic matrix, so all the ants share the latter and each pheromone matrix is responsible for a single objective. This algorithm uses the best solution and second-best solution of each objective to update the pheromone information. For the families of algorithm with multi-pheromone matrix and **multi-heuristic matrix** we have a division according to the number of groups in which the ants are divided. The first part of this family of algorithm uses a **single group** of ants, like in the **BicriterionAnt algorithm**, which, except for the point just made, closely resembles our implementation, both in terms of whether the number of weights equals the number of ants and how the next city in the tour is chosen. The second part of this family of algorithm use instead, more than one group for the ants, an example is the **MOACO algorithm**, which differs from our implementation because the number of weights in a group can be less than the number of ants, and so two or more ants can have the same weights, with which the ants aggregate the pheromone information and the heuristic information by the weighted product method.

7. Conclusion

In the described project we successfully reached the objective of combining the single objective ACO with MOEA/D in order to implement a multiobjective version of the ACO problem.

In the tests we performed several numbers of executions with different parameters, and we always be able to have as output a set of non-dominated solutions, as shown in the Pareto Fronts visible in the images on chapter 5.

ACO is a very good choice to solve real problems, like Traveling Salesman Problem and it is very suitable to a multiobjective solutions instead of the most famous single objective one and MOEA/D confirm itself as a very solid option when we have to solve a MO optimization problem.

As a possible future implementation, we can think about insert a third objective, for example time, that will make the TSP problem a lot more realistic, considering all the three aspects that are related to any possible travel.

8. References

[1] Ant Colony Optimization Marco Dorigo and Thomas Stützle (2004), chapter 3

[2] Moshref, Mahmoud & Al-Sayyed, Rizik & al-sharaeh, Saleh. (2020). Multi-objective optimization algorithms for wireless sensor networks: A comprehensive survey. Journal of Theoretical and Applied Information Technology. 98.

[3] demir, Ibrahim & Corut Ergin, Fatma & Kiraz, Berna. (2019). A New Model for the Multi-Objective Multiple Allocation Hub Network Design and Routing Problem. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2927418.