2022_SEAI_C5
NSGA-II with partially provided k-medoids

Students: Armando Macrì and Giuseppe Cancello Tortora
a.macri1@studenti.unipi.it, g.cancellotortora@studenti.unipi.it
Supervisor: Marco Cococcioni

June 30, 2022

Abstract

The goal of this project is to propose a variant of the well-known NSGA-II algorithm able to use a modified version of the K-Medoids clustering algorithm (i.e., Partially-Provided K-Medoids) in place of the crowding distance.

1 Introduction

Clustering is an unsupervised machine learning task. Using a clustering algorithm means that, giving the algorithm a lot of input data with no labels, it will find any groupings in the data it can, called 'clusters'. For what concerns centroid-based clustering algorithms, 'K-means' and 'K-medoids' algorithms must be cited.

K-means tries to minimize the variance of data points within a cluster. It is best used on smaller datasets because it iterates over all of the data points, meaning that it will take more time to classify data points. Indeed, this clustering algorithm is known to be sensitive to the outliers, because a mean is easily influenced by extreme values, although it is quite efficient in terms of the computational time.

K-medoids clustering algorithm is a variant of K-means that is more robust to noises and outliers. Instead of using the mean point as the center of a cluster, K-medoids uses an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points. This increases the explainability of the approach, as the representative point in the data can always be retrieved. K-medoids algorithm commonly uses the Manhattan metric. In this approach actual objects are used to represent the clusters, selecting one representative object per cluster.

Each remaining object is assigned to the cluster of which the representative object is most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object p and its corresponding representative object, in other words it tries to minimize the cost function defined in this way:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \|p + o_i\|_2^2$$

where p is the data point in the cluster C_i, o_i is the medoid of C_i.

The most common realization of K-Medoids clustering is the Partitioning Around Medoids (PAM) algorithm and is as follows:

Initialize: k data points out of n randomly selected as medoids.

Assignment step: each data point is associated to the closest medoid.

Update step: For each medoid m and each data point o associated to m swap m and o and compute the total cost of the configuration (i.e., the average dissimilarity between o and all the data points associated to m). The medoid with the lowest cost of the configuration is selected.

The K-Medoids algorithm achieved great success in various fields such as cluster color space for color image segmentation methods [1], grouping similar pattern in big data [2] or a fuzzy version of the algorithm (FCM) for geospatial data [3].

In this work, we want to extend the K-Medoids algorithm providing a new version able to find clusters given one or more fixed centroids. As already demonstrated with the ppcFCM [4], this new version of the algorithm may be exploited in real-life problems.

2 Partially-provided K-Medoids algorithm

A variant of the classic K-Medoids has been implemented from scratch. This version, called *Partially-provided K-Medoids algorithm*, differs from the original one for the fact that a set of centroids is fixed and passed as parameter to the algorithm. In this way, the algorithm will clusterize the data points by computing only a subset of free medoids.

The algorithm is composed of the following steps:

Algorithm 1 Partially-provided K-medoids algorithm

```
Input:
  $K$, number of clusters; $D$, a data set of N points, $F$ fixed centroids.
Output:
  A set of $K$ clusters.

  Arbitrary chose $C = K - F$ points from $D$ as candidate medoids
  Pre-compute distances between all the points
  repeat
      Update clusters
      for each cluster $c$ do
          Select the point that minimizes the sum of
          distances as new medoid of $c$
      end for
  until no clusters change.
```

The algorithm takes as input:

- K, the number of clusters
- D, a dataset of N points
- F, the number of fixed centroids

and gives as output a set of K clusters.
The algorithm chooses K - F points from the dataset D arbitrarily, as candidate centroids. Then distances between all the points in D are pre-computed. The set of $F + C$ centroids are used to update the clusters. Furthermore, for each cluster, the point that minimizes the sum of the distances between all the points is selected as new medoid. This is repeated until there is no cluster change.
The main problem of this approach is that it is quite computationally expensive since it requires to pre-compute the distances between all the points in the dataset and so it requires $O(n^2)$ space to fill the squared matrix of distances.

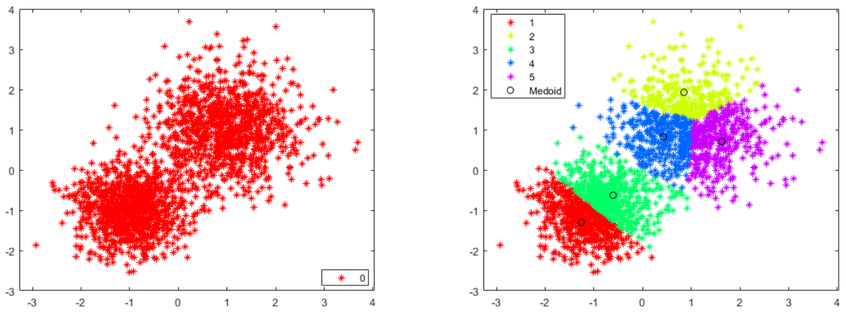

Figure 1: Example of k-M on a set of 1000 points, 5 clusters with no fixed medoids.

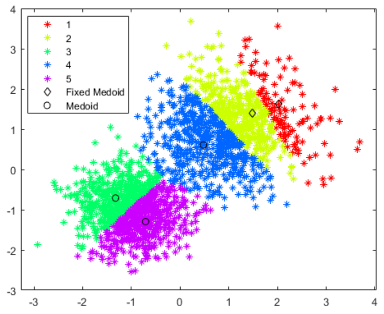

Figure 2: Example of PPk-M on a set of 1000 points, 5 clusters and 2 fixed medoids.

In the figure 1 there is an example of the classical K-Medoids clustering algorithm. The algorithm takes as input a set of 1000 points and returns as output the set of 5 clusters.
In the figure 2, instead, is reported an example of our version of the *Partially-Provided K-Medoids algorithm.* As illustrated in the legend, the two rightmost clusters (i.e., clusters identified by the diamond symbol) are the clusters whose medoids are the fixed medoids passed as input to the algorithm, while clusters identified with the circle are the one whose medoids' positions are computed by using our custom version of the *Partially-Provided K-Medoids algorithm.*

2.1 Comparison with classic K-Medoids

The classic K-Medoids algorithm computes the position of the K centroids by minimizing the following cost function:

$$J_m^{C(F)} = \sum_{d=1}^{D} \sum_{k=1}^{K} \|\boldsymbol{\delta_d} - \boldsymbol{\gamma_k}\|^2, \tag{1}$$

where $\{\boldsymbol{\delta_d}\}_{d=1}^{D}$ are the D points in an M-dimensional space ($\boldsymbol{\delta_d} \in \mathbb{R}^{1xM}$) to clusterize; $\{\boldsymbol{\gamma_c}\}_{c=1}^{C}$ are the C unknown centroids to compute (again, $\boldsymbol{\gamma_d} \in \mathbb{R}^{1xM}$).

In our modified version, we can define a cost function involving all the F+C=K centroids, where we have F centroids that must remain constant to the value passed as parameter and so only C centroids can be optimized in order to minimize the cost function $J_m^{C(F)}$:

$$\begin{cases} J_m^{C(F)} = \sum_{d=1}^{D} \sum_{k=1}^{K} \|\boldsymbol{\delta_d} - \boldsymbol{\gamma_k}\|^2 \\ \textit{subject to:} \\ \boldsymbol{\gamma_k} = \boldsymbol{\pi k}, \forall k = 1...F \textit{ (with } F < K\textit{)} \end{cases}$$

This means that part of the centroids is known in advance, while the other part has to be found. The F provided centroids can be organized within matrix

$$\mathbf{\Pi} \in \mathbb{R}^{MxF} = \begin{bmatrix} \boldsymbol{\pi_1} \\ ... \\ \boldsymbol{\pi_F} \end{bmatrix}.$$

Therefore, the whole centroid matrix $\mathbf{\Gamma}$ can be expressed as:

$$\mathbf{\Gamma} = \begin{bmatrix} \mathbf{\Pi} \\ \mathbf{\Sigma} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\gamma_1} \\ ... \\ \boldsymbol{\gamma_F} \\ \boldsymbol{\gamma_{F+1}} \\ ... \\ \boldsymbol{\gamma_K} \end{bmatrix},$$

where $[\gamma_1, ..., \gamma_P]$ are the F fixed centroids, while $[\gamma_{P+1}, ..., \gamma_K]$ are the K-F=C centroids to be computed.

Algorithm 2 Partially-provided K-medoids algorithm

```
Input:
  Δ, C, l, Π
Output:
  Γ
    1. Set the initial centroids $\boldsymbol{\gamma_k}$ equal to $\boldsymbol{\pi_k}$, $\forall k = 1..F$ and the remaining
       (K-F=C) centroids at random.
    2. $\gamma_k = \sum_{d=1}^{D} \delta_d, \forall k = (F+1)...K$
```

The algorithm differs from the original K-Medoids in the initialization and in the step that updates the centroids (step 2), since in PPk-M the updating formula is used only for free centroids (because the others will remain unchanged throughout the whole optimization problem).

2.2 Evaluation

First of all, some experiments have been carried out on the new version of k-medoids comparing it with the classic one. Some interesting observations can be made. Suppose that in our application domain we already know which regions of interest may be, so we need to have a placeholder in that area, a Medoid for us. In this section the performances of the proposed algorithm are evaluated in terms of both number of iterations and execution time. In order to obtain these results, the algorithm was run by increasing each time the number of fixed medoids by 10%, starting from 0, on the same data set composed the first time by 1000 points and the second time with 10000 points. More reliable results are generated repeating the experiment 10 times and finally computing the average.

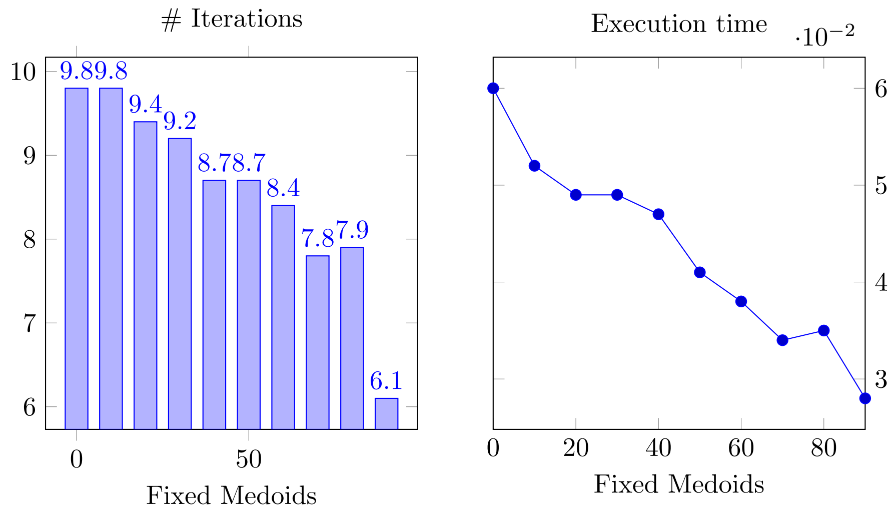

Figure 3: Experiment on 1.000 points in 2D space

Considering 1000 points in a 2D space, the results reported in Figure 3 show an improvement in the algorithm performance if the number of fixed centroids increases. The number of iterations for the algorithm to terminate decreases as the number of fixed medoids increases. The same thing can be said about the running time.

Unlike what we might have expected, the results shown in figure 4 highlight that the number of iterations does not decrease as the number of fixed medoids increases. Although the trend is different from the previous experiment, it is still possible to note that the execution time never exceeds the case in which no medoids are fixed, even if the number of interactions is higher. In any case, the execution time has a decreasing trend, and this time as well.

Obviously the results depend a lot on the initial data set and on the fixed medoids passed to the algorithm.

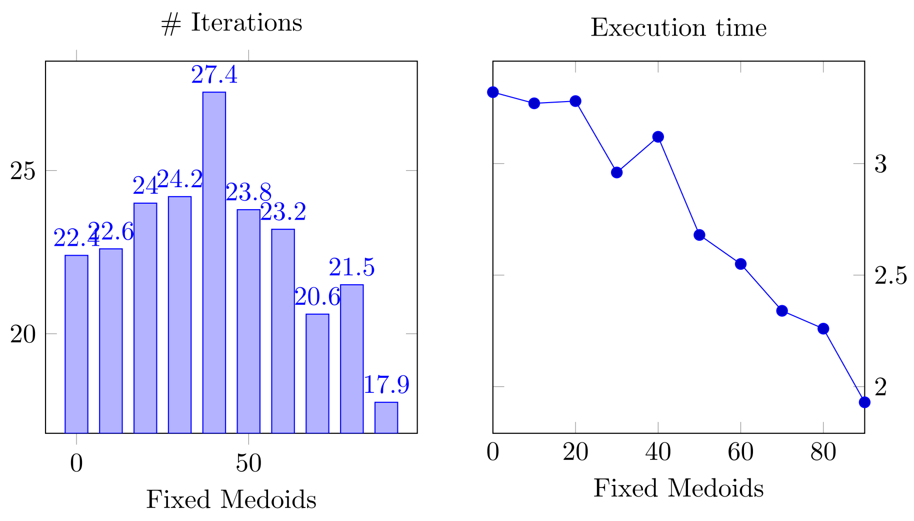

Figure 4: Experiment on 10.000 points in 2D space

3 NSGA-II

NSGA is a popular non-dominated based genetic algorithm for multi-objective optimization.
A modified version, NSGA-II, was developed, which has a better sorting algorithm, incorporates elitism and no sharing parameters needs to be chosen *a priori* [5].

3.1 General description of NSGA-II

The population is initialized as usual. Once initialized, the population is sorted based on non-domination into each front. The first front being completely non-dominant set in the current population and the second front being dominated by the individuals in the first front only and so on. Each individual in each front is assigned rank (fitness) values based on the front in which they belong to. Individuals in the first front are given a fitness value of 1 and individuals in the second one are assigned fitness value of 2 and so on.
In addition to fitness value, a new parameter called ***crowding distance*** is calculated for each individual. The crowding distance is a measure of how close an individual is with respect to its neighbors. Large average crowding distance will result in better diversity in the population.
Parents are selected from the population by using *binary tournament selection* based on the rank and crowding distance. An individual selected in the rank is lesser than the other if crowding distance is greater than the other (i.e., crowding distance is compared only if the ranks for both individuals are the same). The

selected population generate offsprings from *crossover* and *mutation* operators. The new population composed of the current population and current offsprings is sorted again based on non-domination and only the best N individuals are selected, where N is the population size.
The selection is based on the rank and on the crowding distance on the last front.

Algorithm 3 NSGA-II

```
Input:
  $N'$,$g$,$f_k(X)$ ▷ $N'$ members evolved $g$ generations to solve $f_k(X)$
    1. Initialize Population $P'$;
    2. Generate random population - size $N'$;
    3. Evaluate Objective values;
    4. Assign Rank based on Pareto - sort
    5. Generate Child Population;
    6. Binary Tournament Selection;
    7. Recombination and Mutation;
  for $i = 1$ to $g$ do
      for each Parent and Child in Population do
          Assign Rank based on Pareto - sort;
          Generate sets of non-dominated solutions;
          Determine Crowding distance
          Loop (inside) by adding solutions to next generation starting from the
          first front until N' individuals;
      end for
      Select points on the lower front with high crowding distance;
      Create next generation;
      Binary Tournament Selection;
      Recombination and Mutation;
  end for
```

3.2 Detailed description of NSGA-II

2.2.1 Population Initialization The population is initialized based on the problem range and constraints, if any.

2.2.2 Non-Dominated Sorting The initialized population is sorted based on non-domination (i.e., an individual is said to dominate another if its objective functions is no worse than the other and at least in one of its objective functions it is better than the other). The fast non-dominated sorting is described as follows:

- for each individual p in main population P
  - Initialize $S_p = 0$. This set would contain all the individuals that are being dominated by p.
  - Initialize $n_p = 0$. This would be the number of individuals that dominate p.
  - for each individual q in P
    - if p dominated q then
      - add q to the set S_p i.e., $S_p = S_p \cup \{q\}$
    - else if q dominated p then
      - increment the domination counter for p i.e., $n_p = n_p + 1$
    - if $n_p = 0$ i.e., no individuals dominate p then
      - p belongs to the first front; Set $p_{rank} = 1$
      - Update the first front set by adding p, i.e., $F_1 = F_1 \cup \{p\}$
- This is carried out for all the individuals in main population P.
- Initialize the front counter to one. $i = 1$.
- while the ith front is non-empty, i.e. $F_i \neq 0$
  - $Q = 0$. The set for storing the individuals for $(i+1)^{th}$ front.
  - for each individual p in front F_i
    - for each individual q in S_p (set of individuals dominated by p)
      - $n_q = n_q$ - 1, decrement the domination count for individual q.
      - if $n_q = 0$ then none of the individuals in the subsequent front would dominate q.
      - Hence set $q_{rank} = i + 1$
      - Update the set Q with individual q, i.e., $Q = Q \cup \{q\}$.
      - Increment the front counter by one, i.e., $i = i + 1$
      - Now the set Q is the next front. Hence $F_i = Q$.

This algorithm is better than the original NSGA since it utilizes the information about the set that an individual dominate (S_p) and the number of individuals that dominate the individual (n_p).

2.2.3 Crowding distance Once the non-dominated sort is complete, the crowding distance is assigned. Since the individuals are selected based on rank and crowding distance, all the individuals in the population are assigned a crowding distance value. Crowding distance is assigned front wise and comparing the crowding distance between two individuals in different fronts is meaningless.

2.2.4 Selection Once the individuals are sorted based on non-domination and with crowding distance assigned, the selection is carried out by using a ***crowded-comparison-operator***. The comparison is carried out based on:

- non-domination rank p_{rank}, i.e., the individuals in front F_i will have their rank as $p_{rank} = i$.
- crowding distance $F_i(d_j)$

The individuals are selected by using a binary tournament selection with crowded-comparison-operator.

2.2.5 Genetic operators Real coded GA's use ***Simulated Binary Crossover (SBX)*** operator for crossover and ***polynomial mutation***.

2.2.6 Recombination and Selection The offspring population is combined with the current generation population and selection is performed to set the individuals of the next generation. Since all the previous and current best individuals are added in the population, elitism is ensured. Population is now sorted based on non-domination. The new generation is filled by each front subsequently until the population size exceeds the current population size. If by adding all the individuals in front F_j the population exceeds N then individuals in front F_j are selected based on their crowding distance in the descending order until the population size is N. And hence the process repeats to generate the subsequent generations.

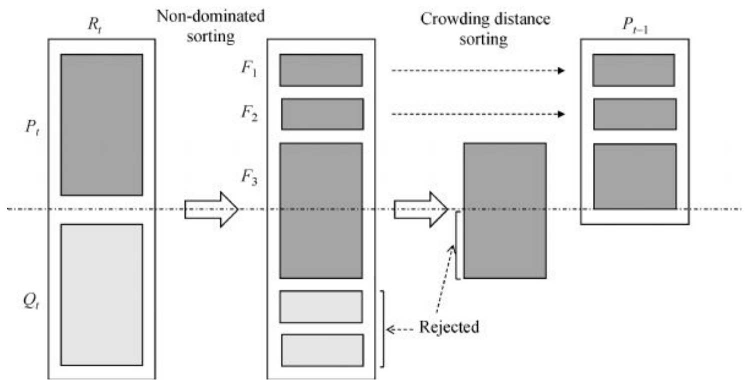

Figure 5: NSGA-II selection schema

4 ppNSGA-II

A different way to solve the problem of multi-objective optimization is to adapt the classic version of the NSGA-II algorithm.

In this new version called *Partially-Provided NSGA-II* (ppNSGA-II), the computation of the crowding distance has been completely eliminated. Remember that, in NSGA-II, the crowding distance is exploited both in the choice of parents during the binary tournament and for the selection of the best individuals, more precisely:

- The `selection policy` used by the classical NSGA-II selects n individuals at random (in general n is equal to 2). From these individuals only one is selected and is added to the mating pool according to two criteria. First, individuals with lower rank are selected. Secondly, if the rank of two individuals is the same, then the crowding distance is compared. Individuals with greater crowding distance are selected.
- After crossover and mutation, the new population is generated, so best individuals are chosen. Initially, until the population size is reached, each front is added one by one until the addition of a complete front which results in exceeding the population size. At this point the chromosomes in that front are added subsequently to the population based on the crowding distance.

In this new version, the crowding distance is not used and it is substituted by the Partially-Provided K-Medoids algorithm. For what concerns the tournament selection of parents, if two individuals have the same rank, one of them will be selected randomly. Instead, during the selection of the best individuals, the chromosomes in the last front to be added will be chosen directly from the proposed algorithm considering the extremes of the front itself as fixed medoids.

5 Results

The benchmark called FON has been used to evaluate the goodness of the proposed solution. The problem is formulated in the following way $\mathbb{R}^3 \rightarrow \mathbb{R}^2$:

$$\begin{aligned} f_1(x) &= 1 - \exp(-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}})^2) \\ f_2(x) &= 1 - \exp(-\sum_{i=1}^{3}(x_i + \frac{1}{\sqrt{3}})^2) \end{aligned} \tag{2}$$

The Equation (2) defines two objects with three inputs in the range $x_i \in [-4, 4]$. The configuration used for this test is the following:

- `population` of 50 individuals
- 200 `generations`

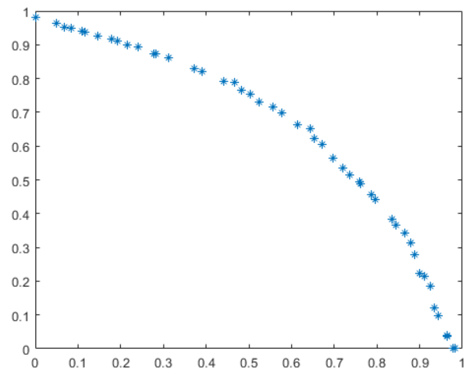

Figure 6: Efficient set obtained with the original NSGA-II

Figure 6 shows the efficient set obtained from the original NSGA-II. As you can see, the points in the objective space are equally distributed in the range $[0, 1]$. Indeed, NSGA-II tries to explore all the objective space.

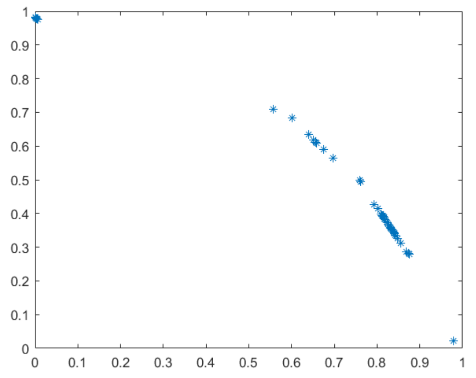

Figure 7: Efficient set obtained with the ppNSGA-II

In figure 7 it is possible to see the efficient set obtained by applying ppNSGA-II. The results obtained are not satisfactory since the solutions obtained are not equally distributed in the space of the objectives. On the contrary, they seem to move towards the extremes.

6 Conclusions

In this work we proposed a variant of the NSGA-II algorithm that exploits the proposed modified version of the K-Medoids clustering algorithm trying to

maintain the same idea of the crowding distance and so to obtain medoids that are equally distributed within the considered front. The algorithm considers a set F of fixed centroids and the remaining (free) centroids are optimized in order to minimize a cost function.
As we can see from the experiments shown above representing the efficient sets obtained, respectively, with the classic version of NSGA-II and with the modified version ppNSGA-II, the results obtained show that it would not be a good idea to introduce the K-Medoids approach for the selection of the best individuals in the considered front in the NSGA-II algorithm since the solutions are not equally distributed as happens in the original version of the algorithm.

References

[1] A. Yerpude and Dubey, "Colour image segmentation using k − medoids clustering," *International Journal of Computer Techology and Applications*, vol. 3, 01 2012.

[2] P. Arora, Deepali, and S. Varshney, "Analysis of k-means and k-medoids algorithm for big data," *Procedia Computer Science*, vol. 78, pp. 507–512, 2016, 1st International Conference on Information Security and Privacy 2015.

[3] M. M. Madbouly, S. M. Darwish, N. A. Bagi, and M. A. Osman, "Clustering big data based on distributed fuzzy k-medoids: An application to geospatial informatics," *IEEE Access*, vol. 10, pp. 20 926–20 936, 2022.

[4] M. Cococcioni, B. Lazzerini, and P. F. Lermusiaux, "Adaptive sampling using fleets of underwater gliders in the presence of fixed buoys using a constrained clustering algorithm," in *OCEANS 2015 - Genova*, 2015, pp. 1–6.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.