2022_SEAI_C5
A comparative analysis between multi/many-objectives Evolutionary Algorithms

Students: Nicola Bicchielli and Lorenzo Biondi
Supervisor: Marco Cococcioni

June 7, 2022

0 Introduction

Evolutionary algorithms (EAs) have emerged as an important optimization and search technique in the last decades. They use a simulated evolution procedure to explore the solutions for complex problems and form a rich class of stochastic search methods. They are therefore best suited to the applications where it is not possible to use heuristic solutions and may however lead to inadequate results in some scenarios. The basic functioning principles of EAs are the following: a population of individuals is initialized and the members of this population are selected and reproduced according to the fitness values and their features. Genes are the units which control these features and the set of genes form the individual's chromosome. Only the fittest individuals survive in the consequent generations and their fittest genes are transmitted to their descendants during the process of recombination known as "crossover". In some rare cases, random mutations are applied to the genes of the individuals to guarantee a higher exploration rate of the search space. The population therefore changes dynamically until a maximum number of generations is reached or a termination condition is met. Fig. 1 shows an example of "crossover" and "mutation" operations when applied to a couple of chromosomes.

Clearly a number of hyperparameters need to be set before beginning the evolutionary process of the population of individuals. Some examples of these hyperparameters are the population size, the duration of the evolutionary process in terms of number of generations, the "crossover rate" (the probability of applying "crossover" to the chromosomes a couple of individuals of the current population) and the "mutation rate" (the probability of randomly mutating a gene of an individual). Designing an EA for a given application requires to select good values for these parameters. The values of all these hyperparameters can also be tuned throughout several runs of the evolutionary process. Unfortunately there is usually very little a-priori knowledge regarding the optimal

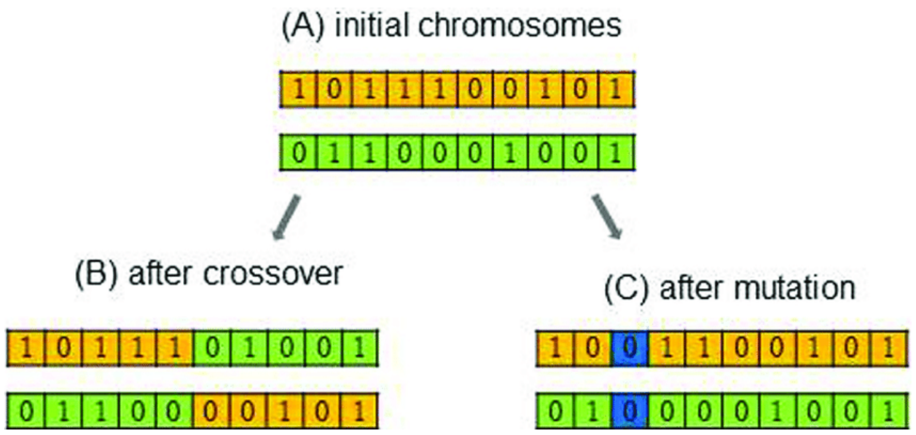


Figure 1: An visual example of ”crossover” and ”mutation” operations when applied to a couple of chromosomes. In this case, the genes are encoded using a binary encoding.

values to choose. This is because there is usually a large number of options, but little knowledge about the effect of the parameters on the EAs' performances.

There are some conventions regarding the choice of the hyperparameter values which are usually followed when using an EA:

- the number of iterations of the evolutionary process should always be rather large. Usually values within the range of 100 and 100,000 iterations are selected to be sure that the EA can reach full convergence of the fitness values of the individuals. Clearly this value is influenced by the speed at which an EA can select the individuals for the next iteration.

- the number of individuals to consider at each iteration should also be quite large. Values in the order of 100 or 1000 are usually preferred since they guarantee a wide variety of genetic material.

- the ”crossover rate” should be rather large in order to allow an intensive exploration of the search space. In the following experiments we always considered a crossover rate which was greater or equal to 70%. If a crossover was set to be executed in some experiments, 2 random chromosomes were chosen and we applied a ”convex crossover” which required to calculate the weighted average between the 2 chromosomes. We applied this type of crossover since the genes of all the individuals we considered were encoded using real numbers belonging to a constant range of values. Clearly also the weights of this weighted average needed to be set. We usually selected a value α randomly from an interval between $-\gamma$ and $+\gamma$, with γ equal to 10% and used α and $1 - \alpha$ as weights for the ”convex crossover” operation.

- the "mutation rate" is usually set to a rather low value in order to introduce a random component which may be useful in the first generations to enhance the exploration of the search space. In our case this rate was always smaller than or equal to 30%. If an individual was selected to have a mutation, this operation was usually by adding to the current chromosome a new chromosome whose genes were selected randomly and were always bounded by a maximum and a minimum values.

Now that the main principles of EAs have been explained, it is easy to understand why in the last decades EAs have been increasingly used. Their simplicity and flexibility allow them to solve a wide variety of complex problems and the fact that they do not require any derivative information allow them to speed up the search process. For these reasons, EAs have often been associated to multi/many-objective optimization problems ([9], [10]). In order to be solved, these problems usually require sets of Pareto-optimal solutions which need a further processing to arrive at a single preferred solution. To achieve the first task, it becomes quite natural to use an EA, since at each iteration it is able to simultaneously find multiple non-dominated solutions. These are solutions which provide trade-offs between different objectives: a solution that obtains high quality values for only one objective inevitably has a compromise for the other objectives. This means that, when dealing with optimization problems with more than one objective, we usually cannot accept a solution which is optimal with respect to only one of the objectives. We should instead aim at finding Pareto-optimal solutions with trade-offs among the various objectives but which are diverse enough to represent the entire range of the Pareto front. Fig. 2 provides a visual example of the Pareto front for a minimization problem with 2 objectives.

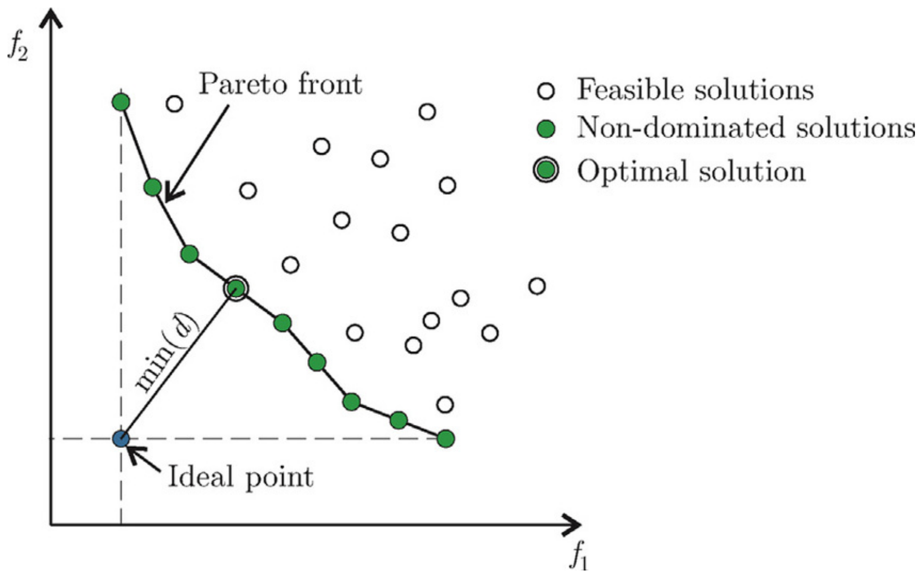


Figure 2: An example of a Pareto front for a minimization problem with 2 objectives.

When different algorithms are compared while solving the same optimization problems and using the same hyperparameters, different metrics can help the user or "Decision Maker" to select the best algorithms:

- the "Generational Distance" (GD) is defined as the distance between each point of the Pareto set found by the algorithm and the closest point in a reference set, averaged over the size of the Pareto set (see [6] for more details).
- the "Inverted Generational Distance" (IGD) is defined as the distance between each point of a reference set and the closest point of the Pareto set found by the algorithm, averaged over the size of the reference set (see [6] for more details).
- the "Delta" value is defined as the maximum value obtained between GD and IGD by an algorithm for each run
- the "Hypervolume" is a performance metric for indicating the quality of a non-dominated approximation set which was introduced by [13]. It measures the space between the points of the Pareto set and a reference point. Fig. 3 shows an example of the hypervolumes related to a couple of Pareto sets.

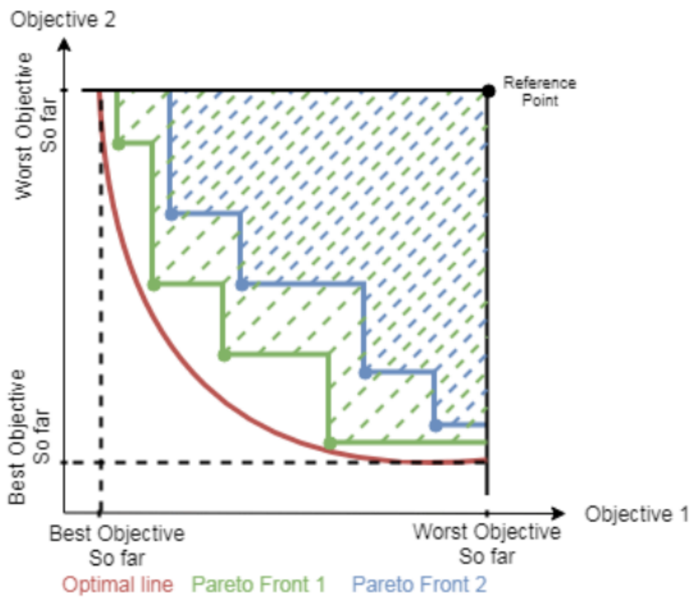


Figure 3: An example of how to calculate the hypervolume of 2 different Pareto sets.

> The areas highlighted with blue and green dashed lines in Fig. 3 indicate the hypervolume of the 2 Pareto sets which were being compared. In this case the green set is better than the blue one since it has a higher hypervolume.

We now moved on to explain more in detail some of the works that introduced the EAs we used in the following experiments.

1 State of the Art

In this study we aimed at analysing some of the most known evolutionary multiobjective optimization algorithms (EMOA). One of the first proposed approaches was "Strength Pareto Evolutionary Algorithm" (*SPEA*) , that was introduced by Zitzler and Thiele ([15]) in 1998. Besides the usual population set, SPEA mantained an external set of individuals ("Pareto set") which contained the Pareto optimal solutions generated so far. This set was used to evaluate the fitness of an individual according to the Pareto dominance relationship. Moreover, a hierarchical agglomerative clustering method was exploited to reduce the Pareto set without destroying its characteristics. More details were provided in the following section, since we implemented this algorithm from scratch using Matlab. Three years later, the same authors proposed an improved version of SPEA called SPEA2 ([14]). This variation incorporated, in contrast to its predecessor, a fine-grained fitness assignment strategy, a density estimation technique and an enhanced archive truncation method based on the k-NN ("k-Nearest Neighbors") algorithm. This new truncation approach was far more efficient than the SPEA's clustering approach and guaranteed the preservation of boundary solutions. In 2002, Kalyanmoy Deb et al. [2] presented NSGA-II (where NSGA stands for "*Non-dominated Sorting Genetic Algorithm*"), a very popular multi-objective optimization algorithm that exploited elitism and did not require any external archive. As the name suggested, the algorithm consisted of a first phase of non-dominated sorting of the current population, followed by the crowding distance sorting. Some test studies ([8]) demonstrated that NSGA-II outperformed PAES and SPEA in terms of finding a diverse set of solutions and convergence to the true Pareto-optimal front. Moving to a many-objectives context (problems involving more than three objectives), the performances of the above algorithms deteriorated severely due to the increased computational cost for evaluating the objective functions and the number of non-dominated solutions in the population. Therefore, specific approaches has been designed to cop with these kind of problems. In 2007, Zhang et al. [12] presented MOEA/D a multiobjective evolutionary algorithm based on decomposition. It decomposed a multiobjective optimization problem into a number of scalar optimization subproblems and optimized them simultaneously. Each subproblem is optimized by only using information from its several neighboring subproblems, which made MOEA/D have lower computational complexity at each generation than NSGA-II. In 2014, Kalyanmoy Deb et al. [3] proposed NSGA-III, a reference-point-based many-objective evolutionary algorithm based on NSGA-II that emphasized population members that are non-dominated, yet close to a set of supplied reference points. In [7] Deb et al. performed further analysis to handle constrained problems.

2 SPEA implementation

Since we did not find a MATLAB implementation of the original Strength Pareto Evolutionary Algorithm (SPEA) algorithm, we decided to implement on our own the code related to SPEA following the original paper [15] written in 1998 and some reference repositories available on Github (https://github.com/rolmez/SPEA, https://github.com/jorgeramirez/AE). This implementation allowed us to apply SPEA to some benchmark problems we wanted to solve.

The basic principles of SPEA are the following:

- it stores the Pareto optimal solutions found so far externally in an "extended Pareto set" of size equal to the Population size used in each generation
- it uses the Pareto dominance of the solutions in order to assign scalar fitness values to individuals
- it performs clustering to reduce the number of non dominated solutions
- it considers also all the solutions in the "extended Pareto set" when performing the selection of the best individuals

In Fig. 4 we inserted an image taken from the original SPEA paper ([15]). It illustrates how the "extended Pareto set" can be used at each generation.

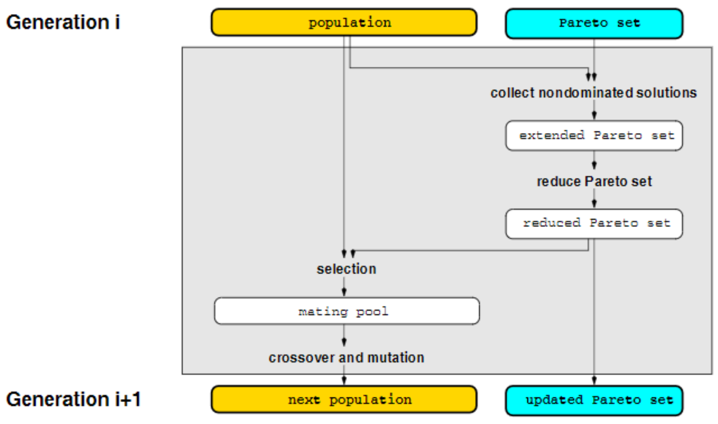


Figure 4: Image taken from the original SPEA paper ([15]) which shows how the external archive ("extended Pareto set") can be used at each generation.

More in detail, in the original paper, at each generation the members of the Pareto Set were assigned a "strength" value between 0 and 1. This value was proportional to the percentage of individuals in the current population that were dominated by the corresponding member of the Pareto set. We executed this part of the algorithm with the following MATLAB code which considered instead the inverse of this percentage, following the approach of https://github.com/jorgeramirez/AE. We did this since our problem required to minimize the objective functions and not maximize them as in [15].

```
nPop=size(Population,1);
fitness_PS=zeros(size(PS, 1),1);
dominated_individuals=zeros(size(PS, 1), nPop);
for i=1:size(PS, 1)
    for j=1:nPop
        if Dominates(PS(i, nVar+1:nVar+M),
                  Population(j, nVar+1:nVar+M))
            dominated_individuals(i, j)=1;
        end
    end
    fitness_PS(i)=1/(sum(dominated_individuals(i, :))/(nPop+1));
end
```

We then exploited the "strength" of the solutions in the Pareto set to calculate the "fitness" values of the individuals of the current Population as follows.

```
fitness_Pop=zeros(nPop,1);
for j=1:nPop
    PS_indexes=find(dominated_individuals(:, j));
    fitness_Pop(j)=1/(sum(1./fitness_PS(PS_indexes))+1);
end
```

In the original paper, for each individual of the current Population, the strengths of all external Pareto solutions by which it was covered were summed up to obtain the corresponding "fitness" value. A value of 1 was added to the result, therefore the "fitness" values of the individuals of the Population ranged between 1 and N. This was done in order to guarantee that Pareto solutions would be the most likely to be reproduced. This occurred because the individuals of the "extended Pareto set" would have a high chance of entering the mating pool if they had a low value of "fitness" or "strength". Since in our case we had to consider a minimization problem, individuals with a higher "fitness" or "strength" were inserted in the mating pool. We therefore summed the inverse values of the strengths of the Pareto set solutions following what was done in https://github.com/jorgeramirez/AE. The individuals therefore had a higher chance of being inserted in the mating pool if they had a high value of "strength" or "fitness". An important aspect to underline was that in case there were Pareto set solutions that dominated an individual of the current population, the "PS_indexes" array would have been empty and the sum of the following line would have correctly returned a value equal to 0.

Another interesting part of our implementation that we wanted to highlight was the function "reducePS" we called to reduce the size of the "extended Pareto set" whenever its size exceeded a given threshold ("max_PS_size"). Besides this threshold, the "extended Pareto set" ("extended_PS") and the number of variables ("nVar") and objectives ("M") were passed to this function. The "extended_PS" matrix contained a row associated to each individual of the "extended Pareto set". The first "nVar" columns contained the individual's chromosome while the last "M" columns contained the objective function values associated to that individual.

```
function PS =reducePS(extended_PS, max_PS_size, nVar, M)
    num_clusters=size(extended_PS,1);

    cluster_indexes=1:num_clusters;

    while num_clusters>max_PS_size
        minDistance=inf;
        cluster_to_merge1=0;
        cluster_to_merge2=0;
        for i=1:num_clusters
            for j=(i+1):num_clusters
                %the clusters are formed by considering the costs
                %associated to their individuals
                cluster1=extended_PS(cluster_indexes==i, nVar+1:nVar+M);
                cluster2=extended_PS(cluster_indexes==j, nVar+1:nVar+M);

                clusterDistance=
                calculate_cluster_distance(cluster1, cluster2);

                if clusterDistance<minDistance
                    minDistance=clusterDistance;
                    cluster_to_merge1=i;
                    cluster_to_merge2=j;
                end

            end
        end
        if cluster_to_merge1<cluster_to_merge2
            cluster_indexes(cluster_indexes==cluster_to_merge2)=
            cluster_to_merge1;
        else
            cluster_indexes(cluster_indexes==cluster_to_merge1)=
            cluster_to_merge2;
        end

        num_clusters=size(unique(cluster_indexes),2);
    end

    PS=zeros(max_PS_size, size(extended_PS, 2));
    row=1;
    i=1;
```

```
while row<=max_PS_size
     cluster_points=extended_PS(cluster_indexes==i, :);
     if size(cluster_points,1)==0
          i=i+1;
          continue
     end

     if size(cluster_points, 1)==1
          % avoids calling the k-medoids function when the cluster
          % is composed of only one individual
          PS(row,:)=cluster_points;
     else
          idx=kmedoids(cluster_points(:, nVar+1:nVar+M),1);
          if size(idx,1)==1
               PS(row, :)=cluster_points(idx, :);
          else
               PS(row, :)=cluster_points(idx(1), :);
          end
     end
     row=row+1;
     i=i+1;
end
PS=PS(1:row-1, :);
```

To prune the "extended Pareto set" we initially decided to follow the hierarchical agglomerative clustering approach of [15]:

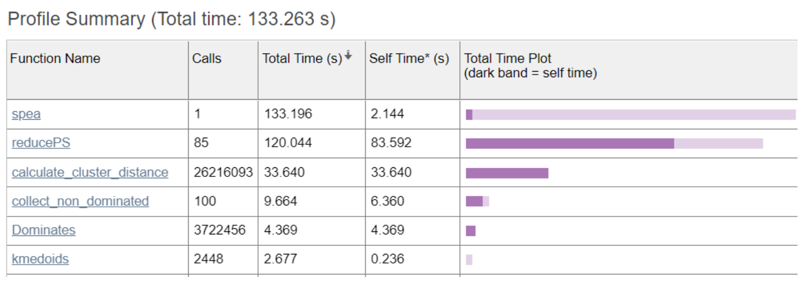
1. at first the clusters we considered were made up of only one element of the "extended Pareto set". Therefore the "points_cluster_indexes" array, which contained the index of the cluster to which each individual belonged to, was initialized with the integer values from 1 to the number of individuals of the "extended Pareto set".

2. we then aggregated the closest clusters together until the number of the clusters was equal to the maximum acceptable value ("max_PS_size"). The distance between two clusters was chosen to be equal to the average distance between pairs of individuals across the two clusters. Clearly this distance between clusters was the most expensive part of the entire algorithm since we had to consider the distances between all possible pairs of clusters. If N was the number of clusters, a total of N*(N-1)/2 distances between pairs of clusters had to be calculated at each iteration. The division by 2 was inserted given the symmetric property of the distances between clusters. Each of these distances required M^2 calculations of distances between points, where M is the average number of points belonging to a cluster at a certain iteration. All these operations had to be repeated for a significant number of iterations in our experiments. This was because the population size and the maximum Pareto set size

("max_PS_size") were always set to the same value (100 in the experiments of section 3). Therefore after the "Pareto set" reached its maximum size, the "extended Pareto set" would always start from a size which was greater than the maximum and then slowly be reduced back to the "max_PS_size" (see Fig. 4 as a reference).

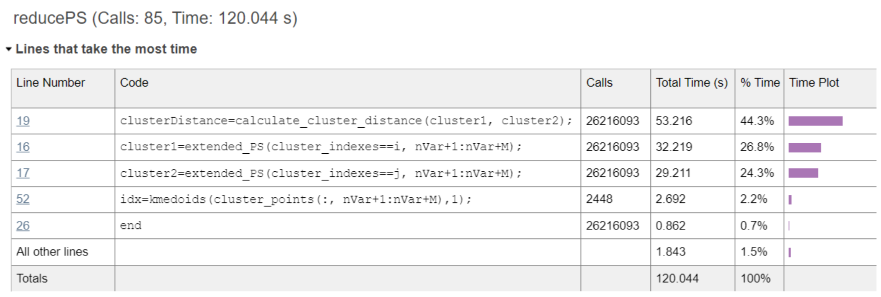
3. when the "max_PS_size" clusters had been found, we needed to select the most representative solution from each of the clusters. If the cluster was formed by only one element, we simply chose that element as the representative one. Otherwise, if the cluster was formed by more than one element, in this case we did not follow what was done in [15]. In fact, we decided to selected the most representative element of the cluster using the "k-medoids" algorithm applied to the cluster with a "k" value equal to 1. The most representative element therefore was not the "centroid", the point with minimal average distance to all the elements in the cluster, as in [15].

While making an initial set of experiments, SPEA was always the algorithm which required far more time to complete all the generations we required. This was in line with the intuitions we had described earlier. Surely calculating the distance between each pair of clusters was extremely expensive from a computational point of view. In the following images, we inserted the results obtained by exploiting MATLAB's "profile viewer" function which allowed us to measure the time required to execute each part of SPEA over the course of 100 generations while solving a simple problem ("SCH") with 1 variable and 2 objectives taken from [4]. The times reported by this profiler were slightly increased due to the overhead introduced by the profiler itself to make all the measurements.


Profile Summary (Total time: 133.263 s)

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
spea	1	133.196	2.144	
reducePS	85	120.044	83.592	
calculate_cluster_distance	26216093	33.640	33.640	
collect_non_dominated	100	9.664	6.360	
Dominates	3722456	4.369	4.369	
kmedoids	2448	2.677	0.236	

Figure 5: Execution times of the SPEA algorithm over a single run of 100 generations with a Population size and Pareto set size of 100 considering the "SCH" problem.


reducePS (Calls: 85, Time: 120.044 s)

▾ Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
19	`clusterDistance=calculate_cluster_distance(cluster1, cluster2);`	26216093	53.216	44.3%	
16	`cluster1=extended_PS(cluster_indexes==i, nVar+1:nVar+M);`	26216093	32.219	26.8%	
17	`cluster2=extended_PS(cluster_indexes==j, nVar+1:nVar+M);`	26216093	29.211	24.3%	
52	`idx=kmedoids(cluster_points(:, nVar+1:nVar+M),1);`	2448	2.692	2.2%	
26	`end`	26216093	0.862	0.7%	
All other lines			1.843	1.5%	
Totals			120.044	100%	

Figure 6: Execution times of the lines of the "reducePS" function we implemented for the SPEA algorithm. The times were evaluated over a single run of 100 generations with a Population size and Pareto set size of 100 while analysing the "SCH" problem.

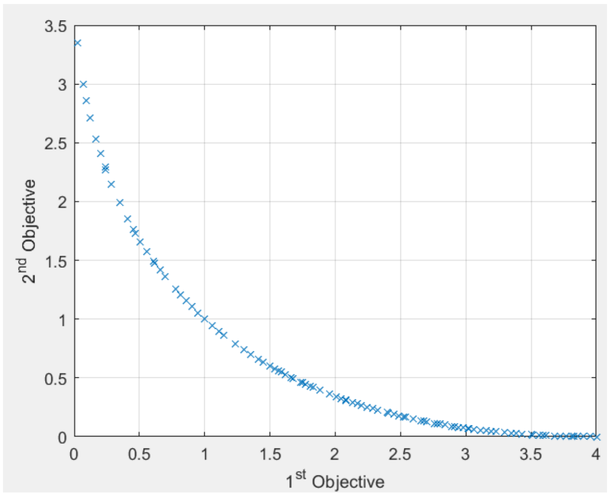
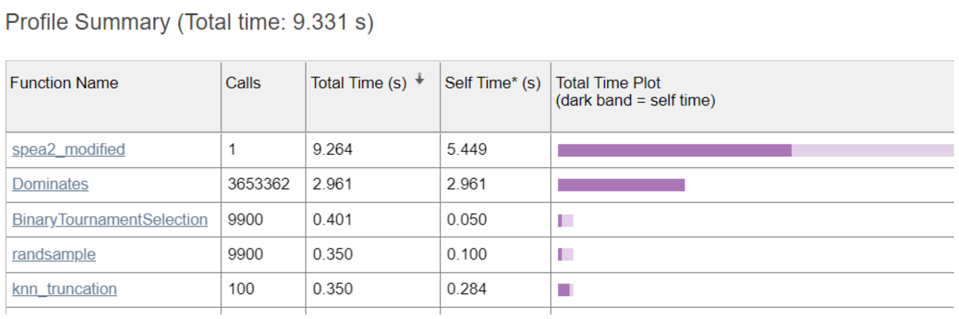
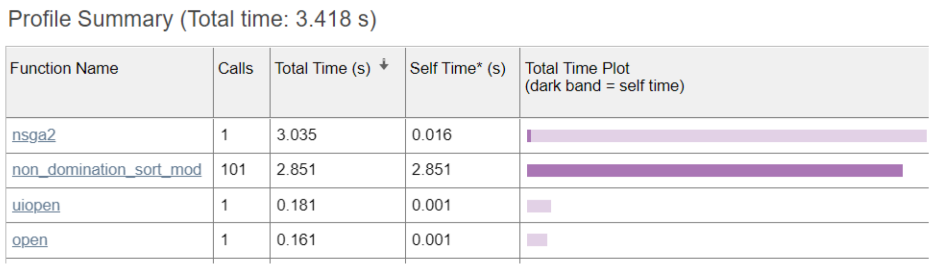


Figure 7: Pareto set found by the SPEA algorithm over a single run of 100 generations with a Population size and Pareto set size of 100 considering the "SCH" problem.


Profile Summary (Total time: 9.331 s)

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
spea2_modified	1	9.264	5.449	
Dominates	3653362	2.961	2.961	
BinaryTournamentSelection	9900	0.401	0.050	
randsample	9900	0.350	0.100	
knn_truncation	100	0.350	0.284	

Figure 8: Execution times of the SPEA2 algorithm over a single run of 100 generations with a Population size and Pareto set size of 100 considering the "SCH" problem.


Profile Summary (Total time: 3.418 s)

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
nsga2	1	3.035	0.016	
non_domination_sort_mod	101	2.851	2.851	
uiopen	1	0.181	0.001	
open	1	0.161	0.001	

Figure 9: Execution times of the NSGA-II algorithm over a single run of 100 generations with a Population size and Pareto set size of 100 considering the "SCH" problem.

From Fig. 5, we were able to understand that the entire run of 100 generations with a Population size and Pareto set size of 100 took over **130 seconds**. As a reference, SPEA2 and NSGA-II required less than 10 seconds (see Fig. 8 and 9). Out of the 133 seconds required by SPEA, **120 seconds were required to call the "reducePS" function** 85 times during the 100 generations. From Fig. 6 we noticed that over 26 million distances between clusters were calculated (**310,777 distance calculations/ function call**).

In this case, we followed the original guidelines of [15], however this number could probably have been significantly reduced if we had saved the distances between the clusters which were not aggregated at each iteration. At each iteration the only distances that must be recomputed are the ones related to the new aggregated cluster and all the other ones. For the remaining clusters, no changes have been carried out and their distances did not need to be recomputed. When the "reducePS" function was called, initially there could be at most 200 non-dominated solutions in the extended Pareto set (100 from the Population and 100 from the current Pareto set). Usually we noticed that instead of 200, the Pareto set at the beginning contained an average of 145 non-dominated individuals. Each of these points was assigned to a different cluster, therefore at the first iteration there were around 145 clusters which had to be reduced to 100. At the first iteration we could have saved the distances between the clusters in a matrix of size 145*145. Once clusters i and j (j>i) were selected to be aggregated together because they were the clusters with the smallest distance between them, we could have simply removed cluster j's corresponding row and column. At the following iteration, instead of calculating the distances between all the possible pairs of the 144 remaining clusters (144*(144-1)/2), we would just have to calculate the new distances between the cluster i and the other 143 clusters. The same considerations could be applied to the following iterations as well. Considering a starting size of the extended Pareto set of 145 and a target size of 100, the total number of distance calculations between clusters **at each call of the "reducePS" function** was approximately

$$\sum_{101 \leq x \leq 145} (x * (x-1))/2$$

, with x equal to the number of individuals of the "extended Pareto set" at the beginning of each generation before being reduced. This formula returned a value of 341,430 distances calculated, which was similar to the 310,777 distances calculated per function call value highlighted earlier. This value could be reduced to only

$$((145-1)^2)/2 + \sum_{101 \leq x \leq 144} (x-1)$$

(15,858) distances calculated/function call if we applied the simple modifications suggested earlier. This improvement would have brought a decrease of 95.36% of distances calculated, leading to a great boost in performance. Clearly the reduction could have been even greater if the Population size and the maximum size of the Pareto set were set to values that were much larger than 100.

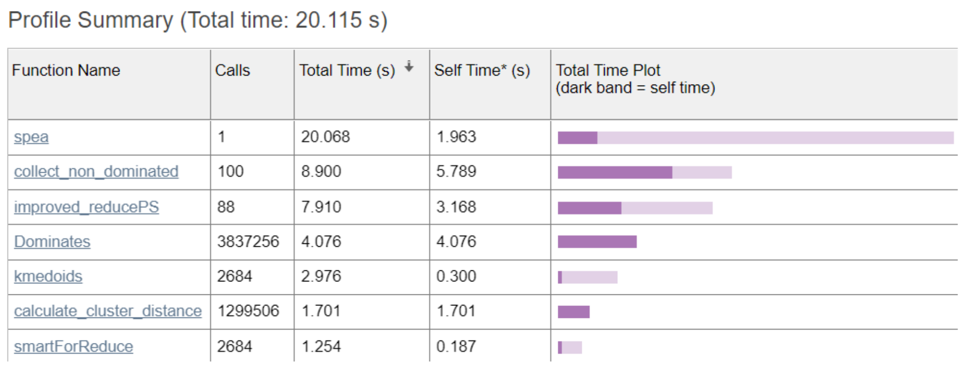
After making all these considerations, we decided to implement this modification to the original "reducePS" function we used with the SPEA algorithm. This was done in order to allow SPEA to have execution times that could have been comparable with those of SPEA2 and NSGA-II.

In the following, we included the first part of the code we used in the "improved_reducePS" function. The remaining part was not included for the sake of brevity since it was the same of the original "reducePS" function we showed earlier.

```
function PS =improved_reducePS(extended_PS, max_PS_size, nVar, M)
    num_clusters=size(extended_PS,1);

    points_cluster_indexes=1:num_clusters;

    % we filled the half of a square matrix with the
    % distances between clusters.
    distances=inf(num_clusters, num_clusters);

    for i=1:num_clusters
        for j=(i+1):num_clusters

            %we did not use the find function to obtain the points
            %belonging to the clusters
            cluster1=extended_PS(i, nVar+1:nVar+M);
            cluster2=extended_PS(j, nVar+1:nVar+M);

            distances(i,j)=calculate_cluster_distance(cluster1, cluster2);
        end
    end


    while num_clusters>max_PS_size

        %get clusters to merge based on their reciprocal distance
        [r,c]=find(distances==min(distances(:)));

        if size(r,1)>1
            %if more than a pair of clusters have the minimum distances
            cluster_to_merge1=r(1);
            cluster_to_merge2=c(1);
        else
            cluster_to_merge1=r;
            cluster_to_merge2=c;
        end


        if cluster_to_merge1<cluster_to_merge2
            %we did not use the find function to obtain the points
            %belonging to the clusters
            points_cluster_indexes(points_cluster_indexes==cluster_to_merge2)=
            cluster_to_merge1;

            %removes column and row belonging to the cluster_to_merge2
            distances(cluster_to_merge2,:)=inf;
            distances(:,cluster_to_merge2)=inf;

            %modifies row and column associated to cluster_to_merge1
            cluster1=extended_PS(points_cluster_indexes==cluster_to_merge1,
            nVar+1:nVar+M);

            for i=unique(points_cluster_indexes)
                if i==cluster_to_merge1
                    continue
                end

                cluster2=extended_PS(points_cluster_indexes==i,
                nVar+1:nVar+M);

                if i<cluster_to_merge1
                    distances(i,cluster_to_merge1)=
                    calculate_cluster_distance(cluster1, cluster2);
                else
                    distances(cluster_to_merge1,i)=
                    calculate_cluster_distance(cluster1, cluster2);
                end
            end
        else
            %we did not use the find function to obtain the points
            %belonging to the clusters
            points_cluster_indexes(points_cluster_indexes==cluster_to_merge1)=
            cluster_to_merge2;

            %removes column and row belonging to the cluster_to_merge1
            distances(cluster_to_merge1,:)=inf;
            distances(:,cluster_to_merge1)=inf;

            %modifies row and column associated to cluster_to_merge2
            cluster2=extended_PS(points_cluster_indexes==cluster_to_merge2,
            nVar+1:nVar+M);

            for i=unique(points_cluster_indexes)
                if i==cluster_to_merge2
                    continue
                end
                cluster1=extended_PS(points_cluster_indexes==i,
                nVar+1:nVar+M);
```

```
                if i<cluster_to_merge2
                    distances(i,cluster_to_merge2)=
                    calculate_cluster_distance(cluster1, cluster2);
                else
                    distances(cluster_to_merge2, i)=
                    calculate_cluster_distance(cluster1, cluster2);
                end
            end

        end

        num_clusters=size(unique(points_cluster_indexes),2);
    end
    ...
```

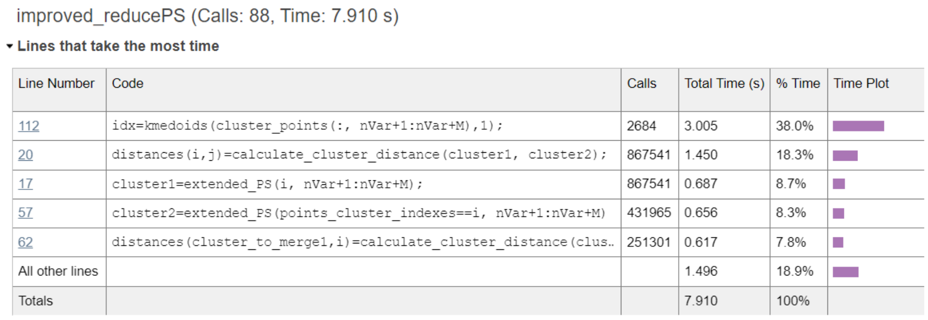
The inferior triangle of matrix containing the distances between clusters was filled with infinite values. This was done in order to not have the possibility of selecting the coordinates of any cell from this part of the matrix as the indices of the clusters which needed to be aggregated. Once clusters i and j were selected to be aggregated together, the elements belonging to cluster j were assigned to cluster i by modifying their corresponding values in the "points_cluster_index" array. The entries of the superior triangle of the "distances" matrix associated to the new cluster i were then updated. The profile of this version of SPEA that used the "improved_reducePS" function was the following.


Profile Summary (Total time: 20.115 s)

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
spea	1	20.068	1.963	
collect_non_dominated	100	8.900	5.789	
improved_reducePS	88	7.910	3.168	
Dominates	3837256	4.076	4.076	
kmedoids	2684	2.976	0.300	
calculate_cluster_distance	1299506	1.701	1.701	
smartForReduce	2684	1.254	0.187	

Figure 10: Execution times of the SPEA algorithm using the "improved_reducePS" function over a single run of 100 generations with a Population size and Pareto set size of 100 considering the "SCH" problem.

The total execution time measured by the profiler in Fig. 10 was much reduced compared to that of Fig. 5: it decreased significantly from 133 seconds to just 20 seconds. More importantly, the "improved_reducePS" function required


improved_reducePS (Calls: 88, Time: 7.910 s)

▾ Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
112	`idx=kmedoids(cluster_points(:, nVar+1:nVar+M),1);`	2684	3.005	38.0%	▬
20	`distances(i,j)=calculate_cluster_distance(cluster1, cluster2);`	867541	1.450	18.3%	▬
17	`cluster1=extended_PS(i, nVar+1:nVar+M);`	867541	0.687	8.7%	▬
57	`cluster2=extended_PS(points_cluster_indexes==i, nVar+1:nVar+M)`	431965	0.656	8.3%	▬
62	`distances(cluster_to_merge1,i)=calculate_cluster_distance(clus…`	251301	0.617	7.8%	▬
All other lines			1.496	18.9%	▬
Totals			7.910	100%	

Figure 11: Execution times of the lines of the ”improved_reducePS” function we implemented for the SPEA algorithm. The times were evaluated over a single run of 100 generations with a Population size and Pareto set size of 100 while analysing the ”SCH” problem.

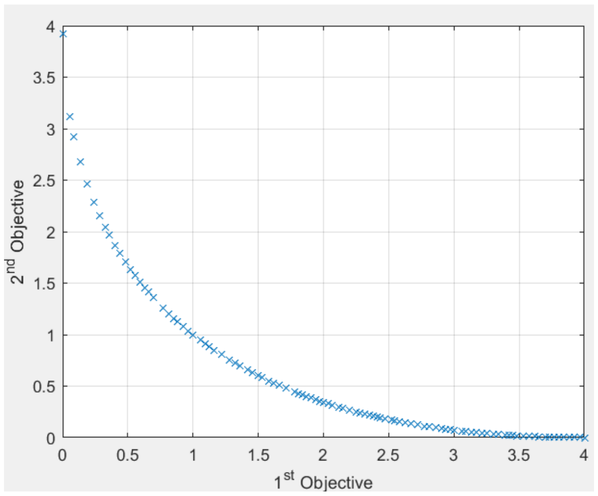


Figure 12: Pareto set found by the SPEA algorithm while using the ”improved_reducePS” functipn over a single run of 100 generations with a Population size and Pareto set size of 100 considering the ”SCH” problem. This Pareto set was extremely similar to the one shown in Fig. 7

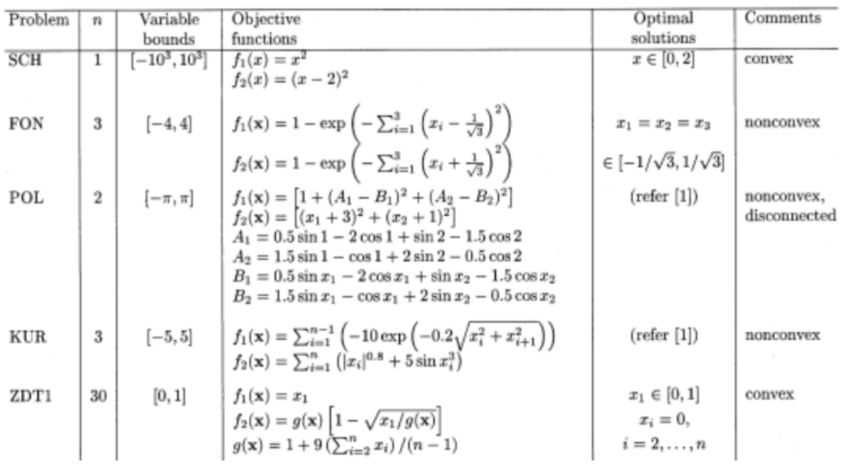
less than 8 seconds to be executed 88 times while the original ”reducePS” function required more than 120 seconds to be executed 85 times (see Fig. 6). With this modification, we were therefore able to reduce the total execution time by around 85%. As we had predicted this considerable drop of the execution times was due to the fact that less than 1.3 million distances between clusters were calculated (see Fig. 10), while in Fig. 5 we saw that this value was 20 times higher (over 26 million cluster distances calculated). The average number of cluster distances calculated for each call of the "improved_reducePS" function was only 14,767. This value was actually lower than the 15,858 we had predicted in the previous paragraph. **The number of cluster distances calculated was decreased by more than 95.2%**, as we had predicted in the previous paragraphs. In Fig. 12, we could also see that even if the "improved_reducePS" function had been used, the Pareto set obtained in this case was extremely similar to the one shown in Fig. 7.

Having analysed all the benefits of the modification we suggested for the SPEA algorithm, we now moved on to perform some experiments to compare our version of SPEA with SPEA2 and NSGA-II when analysing some benchmark problems.

3 Experiments: 1st phase

The first set of experiments we decided to execute regarded 3 different Evolutionary Algorithms (SPEA, SPEA2 and NSGA-II). We decided to implement on our own the code related to SPEA and described how we did this in section 2. We exploited a Matlab version of NSGA-II which was based on the paper written by the research at the Kanpur Genetic Algorithm Laboratory [1] (see also [4]). We also decided to modify the SPEA2 version written by Mostapha Kalami Heris and the Yarpiz team in 2015 (https://yarpiz.com/74/ypea122-spea2) in order to test it on some predefined problems.

These algorithms were used to solve different multi-objective problems. More specifically, the problems we decided to test the algorithms on were taken from [4]. We chose to analyse the first 5 problems out of the 9 that were defined by *Deb et al.*. These problems had different numbers of variables, types of objectives to minimize and variable bounds. Fig.13 was taken directly from [4] and illustrated these problems.



Problem	n	Variable bounds	Objective functions	Optimal solutions	Comments
SCH	1	$[-10^3, 10^3]$	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$x \in [0, 2]$	convex
FON	3	$[-4, 4]$	$f_1(\mathbf{x}) = 1 - \exp\left(-\sum_{i=1}^3 \left(x_i - \frac{1}{\sqrt{3}}\right)^2\right)$ $f_2(\mathbf{x}) = 1 - \exp\left(-\sum_{i=1}^3 \left(x_i + \frac{1}{\sqrt{3}}\right)^2\right)$	$x_1 = x_2 = x_3$ $\in [-1/\sqrt{3}, 1/\sqrt{3}]$	nonconvex
POL	2	$[-\pi, \pi]$	$f_1(\mathbf{x}) = \left[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2\right]$ $f_2(\mathbf{x}) = \left[(x_1+3)^2 + (x_2+1)^2\right]$ $A_1 = 0.5\sin 1 - 2\cos 1 + \sin 2 - 1.5\cos 2$ $A_2 = 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2$ $B_1 = 0.5\sin x_1 - 2\cos x_1 + \sin x_2 - 1.5\cos x_2$ $B_2 = 1.5\sin x_1 - \cos x_1 + 2\sin x_2 - 0.5\cos x_2$	(refer [1])	nonconvex, disconnected
KUR	3	$[-5, 5]$	$f_1(\mathbf{x}) = \sum_{i=1}^{n-1}\left(-10\exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right)\right)$ $f_2(\mathbf{x}) = \sum_{i=1}^{n}\left(\|x_i\|^{0.8} + 5\sin x_i^3\right)$	(refer [1])	nonconvex
ZDT1	30	$[0, 1]$	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = g(\mathbf{x})\left[1 - \sqrt{x_1/g(\mathbf{x})}\right]$ $g(\mathbf{x}) = 1 + 9\left(\sum_{i=2}^n x_i\right)/(n-1)$	$x_1 \in [0, 1]$ $x_i = 0,$ $i = 2, \ldots, n$	convex

Figure 13: Table taken from the original paper of *Deb et al.* ([4]) which describes the problems considered in the first phase of Experiments

All the problems considered in this first phase were all rather simple since they had a limited number of variables to consider and only 2 objective functions. We decided to analyse these simple problems to understand if there could be any evident differences between the algorithms we considered when using them in these circumstances.

In order to compare the algorithms we exploited the different metrics such as the "Generational Distance" (GD), "Inverted Generational Distance" (IGD), the "Delta" value, the "Hypervolume" (see section 0 for the definitions of these

[1] https://it.mathworks.com/matlabcentral/fileexchange/10429-nsga-ii-a-multi-objective-optimization-algorithm

first 4 metrics) and the execution time to obtain the final results. We took the average values of all these metrics over various iterations (for simplicity we set this value to 10). To calculate the GD and IGD values, we needed to compare the Pareto set points found by each algorithm at each iteration to a set of reference points of the problem that was being analysed. We therefore decided to exploit all the 3 algorithms to obtain these reference points of the problems they were being tested on: each of these EAs was run for a number of times (for simplicity we set also this value to 10) and the Pareto set points of all the algorithms over all the iterations were then aggregated to obtain the reference Pareto set.

Before showing the results we obtained from the various experiments of this first phase, in Table 1 we inserted the values we assigned to the hyperparameters that needed to be set. Low values for certain hyperparameters had to be chosen due to limited computational resources.

Hyperparameter	**Value**
Population size, Archive/ Pareto set size	100
Number of Generations	100
Crossover rate	70% (SPEA, SPEA2), 90% (NSGA-II)
Mutation rate	30% (SPEA, SPEA2), 10% (NSGA-II)
Archive dimension (SPEA2)	100
Number of iterations to obtain reference Pareto set	10
Number of iterations to obtain algorithms' Pareto set points	10

Table 1: Hyperparameter values for the first set of Experiments. Low values had to be chosen due to limited computational resources.

Using the hyperparameters of table 1, the following metric values were obtained when comparing SPEA, SPEA2 and NSGA-II. The execution times of each run, which consisted of 100 generations, were obtained using MATLAB's utility function "cputime".

Problem (Variables)	**Algorithm**	**GD**		**IGD**		**Delta**		**Hypervolume**		**Execution time (s)**	
		Avg.	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**
SCH (1)	SPEA	**0.0011**	**3.4771e-04**	0.0359	0.0052	0.0359	0.0052	12.7746	0.7688	12.4672	1.5225
SCH (1)	SPEA2	0.0013	3.0009e-04	**0.0278**	**0.0027**	**0.0278**	**0.0027**	**13.1612**	**0.2697**	4.1959	1.1363
SCH (1)	NSGA-II	60.5860	151.4779	60.4576	150.6239	60.8333	151.3725	9.9735	5.6101	**4.0647**	**4.9249**
FON (3)	SPEA	**0.0066**	**8.0989e-04**	**0.0164**	**9.4449e-04**	**0.0164**	**9.4449e-04**	**0.3063**	**0.0030**	10.0906	1.9895
FON (3)	SPEA2	0.0128	0.0014	0.0205	0.0011	0.0205	0.0011	0.3009	0.0096	4.1099	1.4201
FON (3)	NSGA-II	0.0109	0.0022	0.0214	0.0025	0.0214	0.0025	0.3061	0.0083	**0.6146**	**0.1554**
POL (2)	SPEA	17.3942	36.6470	0.0908	0.0099	17.3942	36.6470	**3.0669 e+04**	**6.3963 e+04**	12.2094	1.9588
POL (2)	SPEA2	0.0168	0.0031	0.1099	0.0153	0.1099	0.0153	321.2492	15.7605	4.2379	1.2583
POL (2)	NSGA-II	**0.0155**	**0.0017**	**0.0777**	**0.0058**	**0.0777**	**0.0058**	352.8877	0.8801	**0.3774**	**0.0128**
KUR (3)	SPEA	0.0389	0.0051	**0.1606**	**0.0308**	**0.1606**	**0.0308**	33.1226	1.1360	8.7984	1.5972
KUR (3)	SPEA2	**0.0367**	**0.0051**	0.1660	0.0196	0.1660	0.0196	32.3246	0.7223	6.3279	3.2378
KUR (3)	NSGA-II	0.3092	0.5228	0.4443	0.3733	0.4750	0.4370	**35.5053**	**9.5307**	**1.3861**	**0.1288**
ZDT1 (30)	SPEA	**4.3593e-04**	**1.8054e-04**	0.1476	0.0645	0.1476	0.0645	0.0957	0.0684	12.2328	0.6809
ZDT1 (30)	SPEA2	6.9022e-04	1.4006e-04	**0.0129**	**0.0085**	**0.0129**	**0.0085**	0.5327	0.1352	4.5850	0.4338
ZDT1 (30)	NSGA-II	1.1564	0.1029	0.9698	0.0565	1.1564	0.1029	**1.0301**	**0.1986**	**0.6370**	**0.0765**

Table 2: Metric values obtained by SPEA, SPEA2 and NSGA-II when applied to our first set of Experiments. For each problem, there were only 2 objective functions to consider and the best results for each metric were put in bold.

Table 2 gave us the opportunity to make some interesting considerations:

- Considering the GD, IGD, Delta and Hypervolume, there was no clear winner amongst the 3 algorithms when applied to all the benchmark problems. In the first problem, SPEA2 was the one which generally obtained the best results. NSGA-II in this case obtained very low quality results compared to the other algorithms. Instead when solving the "FON" problem, it was immediately clear that SPEA was the algorithm which obtained the best results.
- The "POL" problem offered very interesting results regarding the GD, IGD, Delta and Hypervolume metrics: SPEA was the one which obtained the worst results regarding the GD and Delta metrics but obtained a Hypervolume which was several orders of magnitude greater than that of SPEA2 and NSGA-II. We also verified that this value was not an error by making several runs.
- When considering the "ZDT1" problem, the one with 30 variables and 2 objective functions, all 3 algorithms achieved the best results in at least one metric: SPEA obtained the best GD values, SPEA2 obtained the best IGD and Delta values while NSGA-II had the best Hypervolumes and Execution times.
- The metric with the most significant difference between the 3 algorithms was the Execution time: NSGA-II was always the fastest algorithm when analysing all of the 5 problems we had selected. Even with our improvement of the SPEA algorithm, it still was not able to be as fast as SPEA2 or NSGA-II.

In the following images, we inserted for each problem of Fig. 13 the plot of the Pareto set obtained by the algorithm which obtained the best results in Table 2.

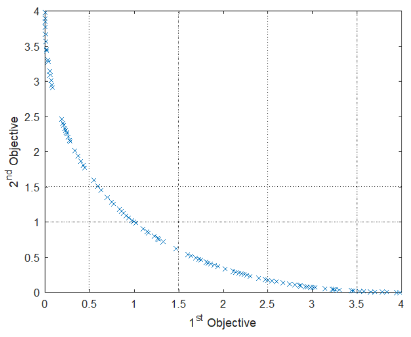


Figure 14: Pareto set obtained by SPEA2 for the "SCH" problem.

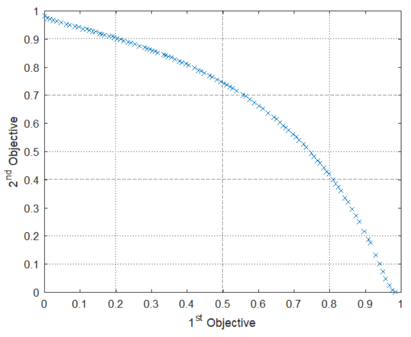


Figure 15: Pareto set obtained by SPEA for the "FON" problem.

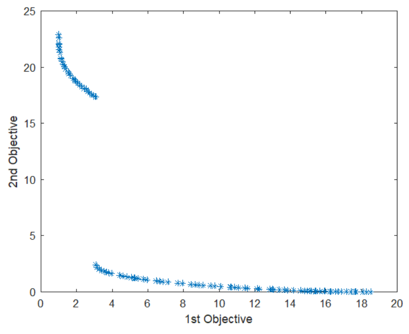


Figure 16: Pareto set obtained by NSGA-II for the "POL" problem.

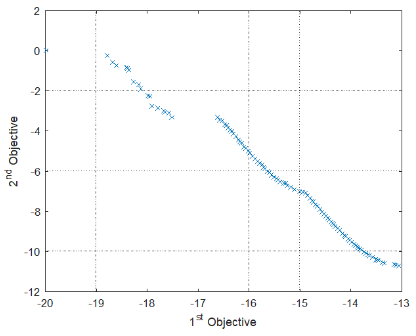


Figure 17: Pareto set obtained by SPEA for the "KUR" problem.

All these results were in line with the Pareto sets shown in the original paper ([4]) in which the problems were defined and in [5]. We now moved on to perform a second set of experiments which analysed more complex problems to see if the evolutionary algorithms could still be efficient.

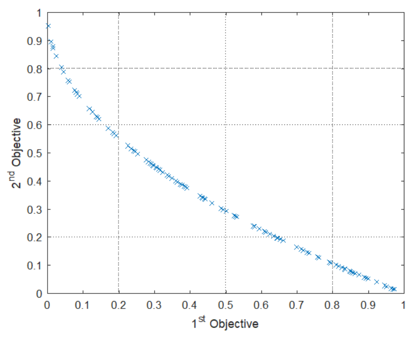
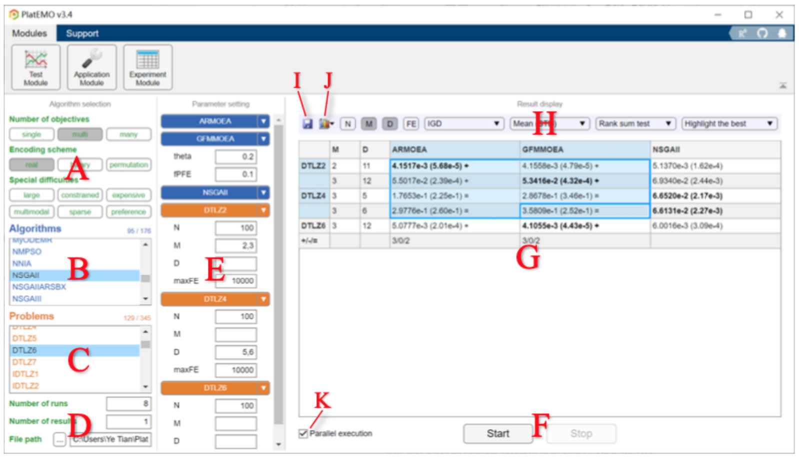


Figure 18: Pareto set obtained by SPEA2 for the "ZDT1" problem.

4 Experiments: 2nd phase

In this second set of experiments, we carried out some comparative analyses between the state of the art algorithms for many-objectives optimization (namely, "NSGA-III" and "MOEA/D") and the ones discussed in chapter 3, in order to highlight strengths and weaknesses of these two sets of algorithms. To perform these experiments we exploited PlatEMO ([11]), a MATLAB platform that provided a variety of algorithms for solving multi/many-objective optimization problems in a very convenient manner. To this aim, users should define the optimization problem to solve, select an algorithm, and set the parameter values. The remarkable quality of PlatEMO can also be deduced from the fact that a user is allowed to add new types of algorithms, problems and operators at will. In our work we used the PlatEMO GUI that can be launched calling the function "*platemo()*" without any parameters. More specifically, we exploited the *"Experiment Module"* to statistically analyze the performances of multiple algorithms on multiple benchmark problems.

Referring to Fig. 19, users should first determine in Region A the general set-up properties of their experiments: if the algorithms were designed to tackle single-multi-many objective optimization problems, an Encoding scheme and other possible properties. Users could then select one of the many available algorithms in Region B, a benchmark problem in Region C, configure the experimental settings in Region D. Finally, the users had to set the parameter values in Region E, where the number of objectives "M" and the number of variables "D" could also be vectors. Then, the optimization process could be initiated and controlled in Region F, while the statistical results were listed in Region G. The statistical results to be listed could be customized in Region H. By pressing Button I ,the user could save the table to a preferred file format, and Button J could display the results in the selected cells of the table in a new figure. Button K determined whether the experiment was performed on a single



Number of objectives: multi
Encoding scheme: real
Algorithms 95 / 176
Problems 129 / 345
Number of runs: 8
Number of results: 1
File path: C:\Users\Ye Tian\Plat

ARMOEA
GFMMOEA: theta 0.2, fPFE 0.1
NSGAII
DTLZ2: N 100, M 2,3, D, maxFE 10000
DTLZ4: N 100, M, D 5,6, maxFE 10000
DTLZ6: N 100, M, D

N M D FE IGD Mean (std) Rank sum test Highlight the best

	M	D	ARMOEA	GFMMOEA	NSGAII
DTLZ2	2	11	**4.1517e-3 (5.68e-5) +**	4.1558e-3 (4.79e-5) +	5.1370e-3 (1.62e-4)
	3	12	5.5017e-2 (2.39e-4) +	**5.3416e-2 (4.32e-4) +**	6.9340e-2 (2.44e-3)
DTLZ4	3	5	1.7653e-1 (2.25e-1) =	2.8678e-1 (3.46e-1) =	**6.6520e-2 (2.17e-3)**
	3	6	2.9776e-1 (2.60e-1) =	3.5809e-1 (2.52e-1) =	**6.6131e-2 (2.27e-3)**
DTLZ6	3	12	5.0777e-3 (2.01e-4) +	**4.1055e-3 (4.43e-5) +**	6.0016e-3 (3.09e-4)
+/-/=			3/0/2	3/0/2	

✓ Parallel execution

Figure 19: An example use of the PlatEMO GUI for the "Experiment Module"

CPU (in sequence) or multiple CPUs (in parallel). All the results were saved to MAT files in the folder specified in Region D. For the complete list of available algorithms and benchmark problems, we advise to new users refer to the official documentation of PlatEMO.

Moving on to the discription of our own experiments, we came up with the following setup:

Hyperparameter	**Value**
Algorithm	NSGA-II, SPEA 2, NSGA-III, MOEA/D
Benchmark problem	MaF5, MaF6, MaF7
Number of objectives (Population size)	5 (125), 10 (250), 15 (375)
Number of generations	10.000
Number of runs	30

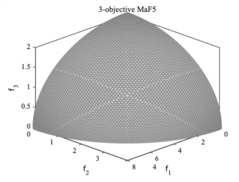
Table 3: The hyperparameters used for the experiments on the PlatEMO platform.

We referred to [1] to produce the setup showed in Tab. 3. Indeed, the authors suggested to set the *"Population size"* to a value of 25 x M, where "M" was the number of objectives (5,10 or 15). Note that the *"Number of runs"* referred to the number of executions for a single <Algorithm, Problem, Number of objectives> tuple and PlatEMO provided us with the mean and standard deviation of the *"Number of runs"* results obtained. Among the 15 benchmark problems (from MaF1 to MaF15) we selected MaF5, MaF6 and MaF7 to be solved by the 4 algorithms. The definitions and the Pareto fronts of such problems were reported in Fig. 20, 21, 22.

$$\min \begin{cases} f_1(\mathbf{x}) = a^M \times \left[\cos\left(\frac{\pi}{2} x_1{}^{\alpha}\right) \dots \cos\left(\frac{\pi}{2} x_{M-2}^{\alpha}\right) \cos\left(\frac{\pi}{2} x_{M-1}^{\alpha}\right) (1+g(\mathbf{x}_M))\right]^4 \\ f_2(\mathbf{x}) = a^{M-1} \times \left[\cos\left(\frac{\pi}{2} x_1{}^{\alpha}\right) \dots \cos\left(\frac{\pi}{2} x_{M-2}^{\alpha}\right) \sin\left(\frac{\pi}{2} x_{M-1}^{\alpha}\right) (1+g(\mathbf{x}_M))\right]^4 \\ \quad \dots \\ f_{M-1}(\mathbf{x}) = a^2 \times \left[\cos(\frac{\pi}{2} x_1^{\alpha}) \sin(\frac{\pi}{2} x_2^{\alpha})(1 + g(\mathbf{x}_M))\right]^4 \\ f_M(\mathbf{x}) = a \times \left[\sin(\frac{\pi}{2} x_1{}^{\alpha})(1 + g(\mathbf{x}_M))\right]^4 \end{cases}$$

with

$$g(\mathbf{x}_M) = \sum_{i=M}^{|\mathbf{x}|} (x_i - 0.5)^2$$



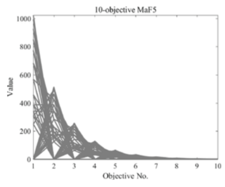
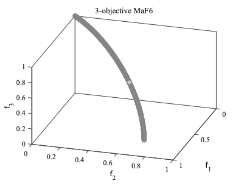


Figure 20: MaF5 problem definition and Pareto front with 3 and 10 objectives shown by Cartesian coordinates and parallel coordinates, respectively.

$$\min \begin{cases} f_1(\mathbf{x}) = \cos(\theta_1) \dots \cos(\theta_{M-2}) \cos(\theta_{M-1})(1 + 100g(\mathbf{x}_M)) \\ f_2(\mathbf{x}) = \cos(\theta_1) \dots \cos(\theta_{M-2}) \sin(\theta_{M-1})(1 + 100g(\mathbf{x}_M)) \\ \quad \dots \\ f_{M-1}(\mathbf{x}) = \cos(\theta_1) \sin(\theta_2)(1 + 100g(\mathbf{x}_M)) \\ f_M(\mathbf{x}) = \sin(\theta_1)(1 + 100g(\mathbf{x}_M)) \end{cases}$$

$$\theta_i = \begin{cases} \frac{\pi}{2} x_i \quad \text{for } i = 1, 2, \dots, I-1 \\ \frac{1}{4(1+g(\mathbf{x}_M))}(1 + 2g(\mathbf{x}_M)x_i) \quad \text{for } i = I, \dots, M-1 \end{cases}$$

$$g(\mathbf{x}_M) = \sum_{i=M}^{|\mathbf{x}|} (x_i - 0.5)^2$$



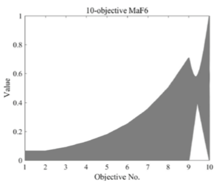
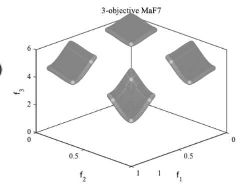


Figure 21: MaF6 problem definition and Pareto front with 3 and 10 objectives shown by Cartesian coordinates and parallel coordinates, respectively.

$$\min \begin{cases} f_1(\mathbf{x}) = x_1 \\ f_2(\mathbf{x}) = x_2 \\ \quad \dots \\ f_{M-1}(\mathbf{x}) = x_{M-1} \\ f_M(\mathbf{x}) = h(f_1,\ f_2,\ \dots,\ f_{M-1},\ g) \times (1 + g(\mathbf{x}_M)) \end{cases}$$

with

$$\begin{cases} g(\mathbf{x}_M) = 1 + \frac{9}{|\mathbf{x}_M|} \sum_{i=M}^{|\mathbf{x}|} x_i \\ h(f_1, f_2, \dots, f_{M-1}, g) = M - \sum_{i=1}^{M-1} \left[\frac{f_i}{1+g}(1 + \sin(3\pi f_i))\right] \end{cases}$$



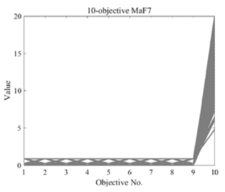


Figure 22: MaF7 problem definition and Pareto front with 3 and 10 objectives shown by Cartesian coordinates and parallel coordinates, respectively.

Using the hyperparameters of table 3, the results reported in Tab. 4, 5 and 6 were obtained. Note that for each problem, we highlighted in bold the best results for the five considered metrics (this information was provided by PlatEMO applying the Wilcoxon signed-rank test, which considered MOEA/D as the reference algorithm).

Problem (Variables)	**Algorithm**	**GD**		**IGD**		**Delta**		**Hypervolume**		**Execution time (s)**	
		Avg.	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**
MaF5 (14)	NSGA-II	0.23211	0.136	2.688	0.188	2.6879	0.1844	0.47655	0.059	**0.40867**	**0.0436**
MaF5 (14)	SPEA2	0.14262	0.0449	**2.4753**	**0.725**	**2.4753**	**0.7124**	0.6332	0.0327	6.3967	0.271
MaF5 (14)	NSGA-III	0.057324	0.00454	2.6146	1.18	2.6145	1.1638	**0.73609**	**0.0491**	0.78554	0.102
MaF5 (14)	MOEA/D	**0.03761**	**0.0097**	10.922	2.84	10.9218	2.7954	0.29616	0.115	5.8403	0.724
MaF6 (14)	NSGA-II	2.5444e-4	8.3e-5	0.0063382	0.00123	0.006	0.0012	**0.12748**	**8.99e-4**	**0.40122**	**0.0344**
MaF6 (14)	SPEA2	2.7145e-4	8.47e-5	**0.00548**	**0.0011**	**0.0054**	**0.0011**	0.12709	7.68e-4	2.0015	0.131
MaF6 (14)	NSGA-III	3.6011e-4	9.99e-5	0.024878	0.00737	0.0248	0.0072	0.12462	0.00115	0.71102	0.097
MaF6 (14)	MOEA/D	**1.4154e-5**	**1.05e-5**	0.32416	0.203	0.32415	0.1998	0.063292	0.0551	5.6233	0.276
MaF7 (24)	NSGA-II	0.11367	0.0545	0.71417	0.0883	0.7141	0.0867	0.046528	0.0202	**0.42743**	**0.0373**
MaF7 (24)	SPEA2	0.067292	0.0221	**0.4753**	**0.0941**	**0.4752**	**0.09255**	0.10255	0.0302	5.9238	0.211
MaF7 (24)	NSGA-III	0.045419	0.0131	0.5218	0.114	0.5215	0.1116	**0.15134**	**0.0227**	0.78107	0.0283
MaF7 (24)	MOEA/D	**0.00667**	**0.00129**	1.123	0.143	1.1230	0.14105	0.0076691	0.0122	5.5223	0.123

Table 4: Metric values obtained by NSGA-II, SPEA2, NSGA-III and MOEA/D when considering **5 objectives** for the three benchmark problems

Problem (Variables)	**Algorithm**	**GD**		**IGD**		**Delta**		**Hypervolume**		**Execution time (s)**	
		Avg.	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**
MaF5 (19)	NSGA-II	37.033	1.96	88.499	6.32	88.4993	6.2153	0.0025384	0.00795	**0.3910**	**0.0411**
MaF5 (19)	SPEA2	59.518	1.66	153.16	12.4	153.1559	12.1516	0.0121	0.0167	53.041	4.51
MaF5 (19)	NSGA-III	4.0247	0.592	**83.426**	**4.79**	**83.4255**	**4.7136**	**0.8060**	**0.0379**	0.84587	0.0762
MaF5 (19)	MOEA/D	**0.01945**	**0.018**	305.14	1.27	305.1442	1.2497	0.1745	0.0876	6.5852	0.502
MaF6 (19)	NSGA-II	10.173	1.98	4.3213	2.08	10.1742	1.9454	0.0028	0.0156	**0.40916**	**0.0417**
MaF6 (19)	SPEA2	10.388	0.787	7.4418	3.09	10.6511	0.9753	0	0	21.824	2.2
MaF6 (19)	NSGA-III	6.2434	1.84	3.8846	2.01	6.4418	1.9051	0.00285	0.0109	0.7451	0.0538
MaF6 (19)	MOEA/D	**7.6065e-6**	**7.76e-6**	**0.26305**	**0.251**	**0.2630**	**0.2469**	**0.0593**	**0.0454**	6.1639	0.058
MaF7 (29)	NSGA-II	2.4013	0.0783	12.739	1.97	12.738	1.9325	0	0	**0.4566**	**0.0186**
MaF7 (29)	SPEA2	2.5296	0.0995	7.6502	1.79	7.6502	1.7562	1.639e-8	8.98e-8	46.996	0.842
MaF7 (29)	NSGA-III	0.69627	0.0876	8.7353	0.981	8.7352	0.9644	3.1153e-5	9.9e-5	0.83123	0.0318
MaF7 (29)	MOEA/D	**0.01914**	**0.00239**	**2.5865**	**0.536**	**2.5864**	**0.5268**	**2.0515e-4**	**4.52e-4**	6.2738	0.352

Table 5: Metric values obtained by NSGA-II, SPEA2, NSGA-III and MOEA/D when considering **10 objectives** for the three benchmark problems

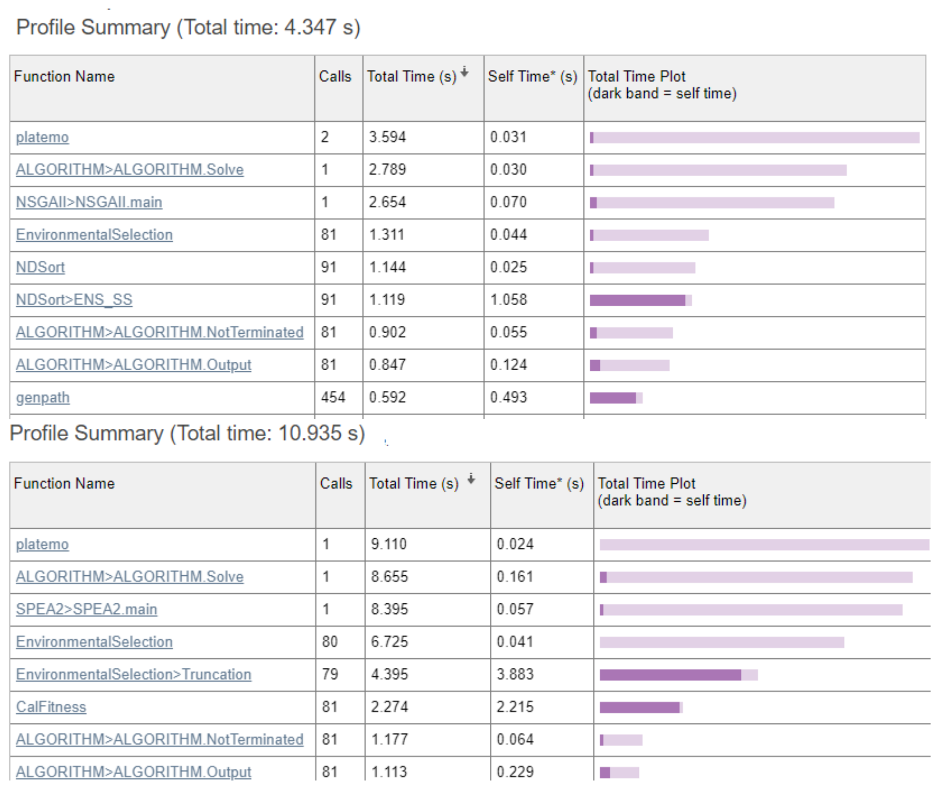
Problem (Variables)	**Algorithm**	**GD**		**IGD**		**Delta**		**Hypervolume**		**Execution time (s)**	
		Avg.	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**	**Avg.**	**Std. Dev.**
MaF5 (19)	NSGA-II	752.11	39.4	1656.5	94.6	1656.517	93.005	2.6724e-4	7.25e-4	**0.41131**	**0.0286**
MaF5 (19)	SPEA2	1514.9	29.6	4128	396	4127.98	389.157	0.010792	0.012	141.04	1.82
MaF5 (19)	NSGA-III	123.13	13.2	**1607.3**	**115**	**1607.31**	**112.756**	**0.9066**	**0.0228**	0.87236	0.0481
MaF5 (19)	MOEA/D	**0.043405**	**0.083**	7323	7.15	7323.02	7.0289	0.14726	0.0556	6.483	0.333
MaF6 (19)	NSGA-II	7.7948	0.381	5.9292	1.84	7.94621	0.6068	0	0	**0.41927**	**0.0318**
MaF6 (19)	SPEA2	9.2073	0.329	17.329	8.44	17.5581	8.0337	0	0	69.876	3.48
MaF6 (19)	NSGA-III	5.2798	0.636	5.792	2.24	6.4680	1.6902	0	0	0.86714	0.028
MaF6 (19)	MOEA/D	**7.4581e-6**	**7.13e-6**	**0.2892**	**0.194**	**0.2892**	**0.1910**	**0.03655**	**0.0447**	6.4129	0.422
MaF7 (29)	NSGA-II	3.1373	0.0819	27.326	3.88	27.3262	3.8142	0	0	**0.57974**	**0.0339**
MaF7 (29)	SPEA2	3.1905	0.0834	17.19	3.07	17.1898	3.0233	0	0	152.92	29
MaF7 (29)	NSGA-III	1.6224	0.162	23.752	2.44	23.7516	2.3954	0	0	1.367	0.0525
MaF7 (29)	MOEA/D	**0.06133**	**0.0117**	**3.0872**	**0.466**	**3.0872**	**0.4586**	**7.9081e-6**	**2.18e-5**	9.3206	0.17

Table 6: Metric values obtained by NSGA-II, SPEA2, NSGA-III and MOEA/D when considering **15 objectives** for the three benchmark problems

By analysing Tab. 4, 5 and 6, some interesting observations were drawn:

- Coherently with the experiments carried out in section 3, we reported the hypervolume metric regarding these experiments as well. Unfortunately, when dealing with high-dimensional objective spaces, this metric could be not so effective and significant in reporting the goodness of a GA. Moreover, the computation of such volume was very costly and in some cases ended up with a value of 0, especially in Tab. 6 (15 objectives).

- When observing the *Execution time*, we immediately understood that NSGA-II outperformed all the other algorithms in every experiment. On the other hand, SPEA2 proved to be the slowest, except for MaF6 with 5 objectives where MOEA/D performed worse. When comparing SPEA2 with NSGA-II, we could immediately notice that the former algorithm needed a running time much higher than the latter, with a factor of 15 for the MaF5 and MaF7 problems on 5 objectives (on MaF5, SPEA2 seemed to reach fairly good performances). When the number of objectives were increased, the execution times of SPEA2 increased even more, while NSGA-II kept a constant running time. To better understand this behavior, we profiled SPEA2 and NSGA-II during a single execution with 10.000 generations on the MaF7 problem, considering 5 objectives. The results were shown in Fig. 23 and 24.



Profile Summary (Total time: 4.347 s)

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
platemo	2	3.594	0.031	
ALGORITHM>ALGORITHM.Solve	1	2.789	0.030	
NSGAII>NSGAII.main	1	2.654	0.070	
EnvironmentalSelection	81	1.311	0.044	
NDSort	91	1.144	0.025	
NDSort>ENS_SS	91	1.119	1.058	
ALGORITHM>ALGORITHM.NotTerminated	81	0.902	0.055	
ALGORITHM>ALGORITHM.Output	81	0.847	0.124	
genpath	454	0.592	0.493	

Profile Summary (Total time: 10.935 s)

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
platemo	1	9.110	0.024	
ALGORITHM>ALGORITHM.Solve	1	8.655	0.161	
SPEA2>SPEA2.main	1	8.395	0.057	
EnvironmentalSelection	80	6.725	0.041	
EnvironmentalSelection>Truncation	79	4.395	3.883	
CalFitness	81	2.274	2.215	
ALGORITHM>ALGORITHM.NotTerminated	81	1.177	0.064	
ALGORITHM>ALGORITHM.Output	81	1.113	0.229	

Figure 23: The profiling results for NSGA-II and SPEA2 on MaF7 with 5 objectives

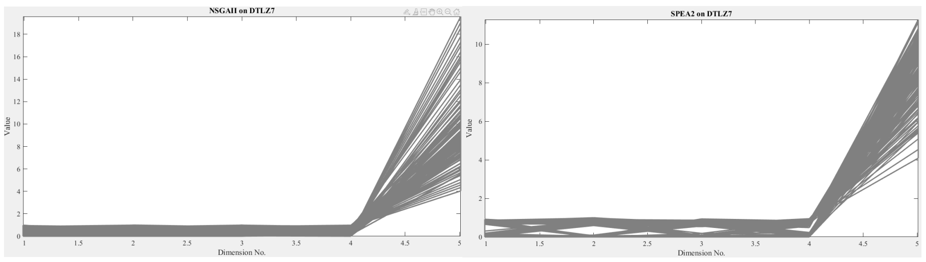


Figure 24: The Pareto fronts found by NSGA-II and SPEA2 on MaF7 with 5 objectives

The times reported in Fig. 23 were higher than the ones showed in Tab. 4 since the MATLAB profiler included all the overhead related to the profiling into the total execution time. However, in relative terms, SPEA2 was still proved to be much slower than NSGA-II. Looking at the bottom table in Fig. 23, the main time bottleneck for SPEA2 was the archive truncation procedure, that was called 79 times and took 3.8 seconds to execute. Instead, the most significant bottleneck for NSGA-II was the Non-Dominated sorting which required 1.058 seconds. When analysing Tab. 5 and 6, we observed that the execution time of SPEA2 increased with the number of objectives, demonstrating the SPEA2's inability of scaling in a many-objective context. Fig. 25 highlighted this behavior by reporting the execution time (in seconds) for NSGA-II and SPEA2 when solving ZDT1 (2 objectives) and MaF7 (with 5,10 and 15 objectives). For further comparisons, we reported also the times required by NSGA-III and MOEA/D. To make the plot more readable, the y-axis values were plotted in a logarithmic scale.

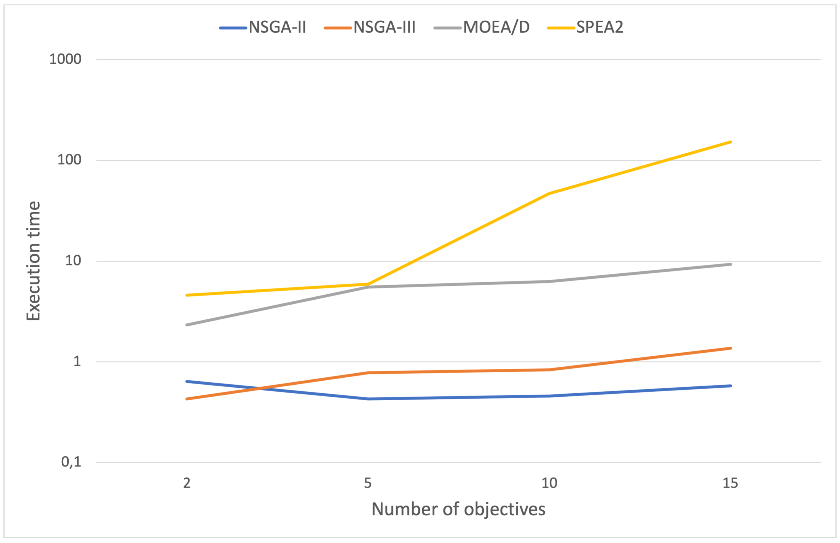


Figure 25: The execution times of NSGA-II, SPEA2, NSGA-III and MOEA/D when solving the "ZDT1" problem (2 objectives) and the "MaF7" problem (considering 5, 10 and 15 objectives).

- By analysing the Delta metric values, which were taken as the maximum between the GD and IGD values, of the previous tables we could notice how the algorithms that were specifically designed for many-objectives optimization provided the best results. Tab. 5 and 6 showed that the best algorithms are NSGA-III and MOEA/D. However, it was worth to notice that, in case of 5 objectives (Tab. 4), the best values for the Delta metric were obtained by SPEA 2, with an execution time that was comparable to the one required by MOEA/D. The reason for this could be that when coping with "not so many" objectives, NSGA-III and MOEA/D did not provide any specific advantage compared to other "classical" EAs. Instead, with an increasing number of objectives, the latter became unable to deal with such high-dimensional spaces. When looking for example at Tab. 6, we could appreciate the great advantages provided by the application of MOEA/D. For instance, when solving problem MaF6, MOEA/D obtained a Delta of 0.2892, while SPEA2 reached a value of 17.329 . To have a complete overview of the Delta values for the different algorithms, we produced the plot in Fig. 26.

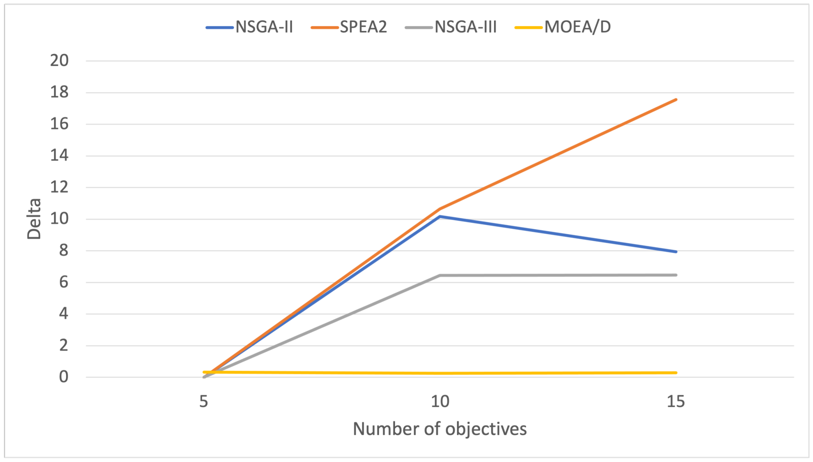


Figure 26: The Delta value obtained by NSGA-II, SPEA2, NSGA-III and MOEA/D when solving the "MaF6" problem with 5, 10 and 15 objectives.

The picture clearly showed the effectiveness of MOEA/D, that always obtained Delta values lower than 0.4 . On the other hand, SPEA2 demonstrated a monotonic increasing behavior with respect to the number of objectives. Finally, NSGA-III obtained stable and fairly good results.

5 Conclusions

In the first part of this work we described in detail several Evolutionary Algorithms (”EAs”) and illustrated the original papers from which the EAs were taken. Since we did not find an online version of the SPEA algorithm in MATLAB, we implemented it on our own, using [15] and several Github repositories as a reference. We also suggested some modifications to the original SPEA algorithm and verified that they could provide significant improvements in the efficiency of the algorithm, with a reduction of the execution time of around 85%.

After this, we made a first set of experiments in which we compared the performances of our modified version of SPEA with SPEA2 and NSGA-II. In these experiments, the algorithms were tested on 5 different minimization problems (with only 2 objectives) that were taken from [4]. We made an in-depth analysis regarding the results obtained by the 3 algorithms and verified that the Pareto sets they found corresponded to those found in our reference paper.

In the second set of our experiments, we used the PlatEMO platform to compare the performances and scalability of SPEA2, NSGA-II, NSGA-III and MOEA/D while considering more many-objective problems taken from [1]. From here, we noticed that SPEA2 had significant scalability issues which were not present in NSGA-II. Nevertheless, we were able to verify that the best GD, IGD and Delta values were almost always obtained by NSGA-III and MOEA/D. This occurred especially when the problems became more and more complex (10/15 objectives). We could therefore confirm that algorithms designed to solve many-objectives problems were capable of achieving by far the best results in these circumstances.

References

[1] Ran Cheng et al. “A benchmark test suite for evolutionary many-objective optimization”. English. In: *Complex Intelligent Systems* 3.1 (Mar. 2017), pp. 67–81. DOI: 10.1007/s40747-017-0039-7.

[2] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[3] Kalyanmoy Deb and Himanshu Jain. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints”. In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 577–601. DOI: 10.1109/TEVC.2013.2281535.

[4] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[5] V Huang et al. “Differential Evolution with External Archive and Harmonic Distance-Based Diversity Measure”. In: (July 2008).

[6] Hisao Ishibuchi et al. “Modified distance calculation in generational distance and inverted generational distance”. In: *International conference on evolutionary multi-criterion optimization.* Springer. 2015, pp. 110–125.

[7] Himanshu Jain and Kalyanmoy Deb. “An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach”. In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 602–622. DOI: 10.1109/TEVC.2013.2281534.

[8] Xinyou Lin, Zhili Lin, and Shenshen Wei. “Multi-objective optimized driving strategy of dual-motor EVs using NSGA-II as a case study and comparison of various intelligent algorithms”. In: *Applied Soft Computing* 111 (2021), p. 107684. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2021.107684. URL: https://www.sciencedirect.com/science/article/pii/S1568494621006050.

[9] Kay Chen Tan, Tong Heng Lee, and Eik Fun Khor. “Evolutionary algorithms for multi-objective optimization: Performance assessments and comparisons”. In: *Artificial intelligence review* 17.4 (2002), pp. 251–290.

[10] Lothar Thiele et al. “A Preference-Based Evolutionary Algorithm for Multi-Objective Optimization”. In: *Evolutionary Computation* 17.3 (2009), pp. 411–436. DOI: 10.1162/evco.2009.17.3.411.

[11] Ye Tian et al. “PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization [Educational Forum]”. In: *IEEE Computational Intelligence Magazine* 12.4 (2017), pp. 73–87. DOI: 10.1109/MCI.2017.2742868.

[12] Qingfu Zhang and Hui Li. “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”. In: *IEEE Transactions on Evolutionary Computation* 11.6 (2007), pp. 712–731. DOI: 10.1109/TEVC.2007.892759.

[13] Eckart Zitzler and Simon Künzli. “Indicator-based selection in multiobjective search”. In: *International conference on parallel problem solving from nature*. Springer. 2004, pp. 832–842.

[14] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. “SPEA2: Improving the strength pareto evolutionary algorithm”. In: 2001.

[15] Eckart Zitzler and Lothar Thiele. *An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach*. 1998.