MASTER AIC

Indicator-Based Selection in Multi-objective Search

Presenté par :	*Email*
Amal TARGHI	amal.targhi@u-psud.fr
Anh Khoa NGO HO	anh-khoa.ngo-ho@u-psud.fr
Fella BELKHAM	fella.belkham@u-psud.fr
Mahmut CAVDAR	mahmut.cavdar@u-psud.fr
Trong Bach VU	jsbachvu@gmail.com

Présenté à :
Paris Sud University

10 octobre 2016

Table des matières

1 Introduction 2
2 Multi-objective Optimization 2
3 Evolutionary algorithms 3
4 Indicator-Based Selection in Multiobjective Search 4
4.1 Discussion 4
5 Progress 7
5.1 What have been done 7
5.2 Results 10
6 Problems faced 15
7 What remains to do ? 15
8 References 16

1 Introduction

This report serves as intermediate report in which we present our progress in the optimization project. In the following we will summarize the research we have done on the evolutionary algorithm and multi-objective optimization, discuss the article we've chosen and finally detail what has been done, what remain to do and describe the problems encountered during the work. For this project we have chosen to implement the algorithm 4 : Indicator-Based Selection in Search Multiobjective proposed using the Python language.

2 Multi-objective Optimization

Optimization problems are common in various disciplines,including mechanics, economics, logistics and biomolecular, that need to be solved. In optimization problems, we have to find best solutions which are optimal or near-optimal from all feasible solutions with respect to some constraints. We can represent an optimization problem with the following form :

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq b_i, \; i = 1, \ldots, m \\ & h_i(x) = 0, \; i = 1, \ldots, p. \end{aligned} \tag{1}$$

where f(x) is the objective function over the variable x, $g_i(x)$ and $h_i(x)$ are called constraints. The main difference between multi and mono-objective optimization problem is additional objective functions to be optimized. For representation of multi-objective optimization problem we need to change only first part of form :

$$\underset{x}{\text{minimize}} \quad (f_1(x), f_2(x), f_3(x), ..., f_k(x)) \tag{2}$$

where the integer $k \geq 2$ is the number of objective functions. We cannot apply our mono-objective optimization algorithm to only one objective, when others objectives are also important. All different objectives contain its own optimal solution. A solution can be optimal solution for an objective, on the other hand, it may require compromise for another objective. If we need to express it in a mathematical expression :

$$\underset{x}{\text{minimize}} \quad f_1(x) \quad \neq \quad \underset{x}{\text{minimize}} \quad f_2(x) \tag{3}$$

where $f_1(x)$ and $f_2(x)$ are two different objectives of same problem. And we can not say which solution is the best with respect to both objectives. Because each objective corresponds to different optimal solution. If that is the case, we have to keep both these solutions. So, solution set is composed by all optimal solution. No solution f rom the final set can be said to be better than others. This set of optimal solutions leads us to Pareto optimal solutions notion ;that is, solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives. (Figure1)

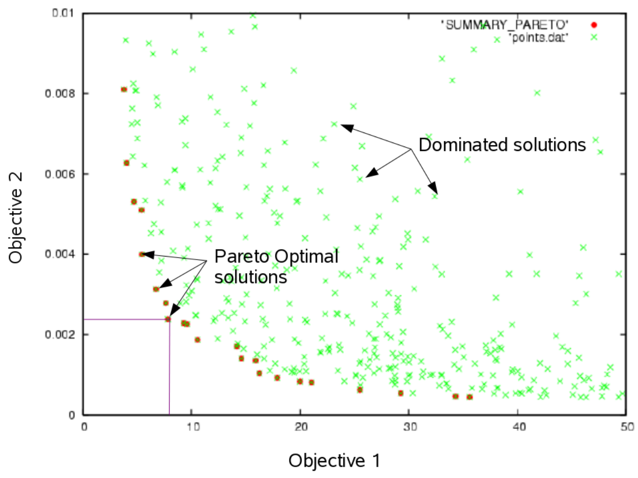


FIGURE 1 – Example of a Pareto frontier[Cenaero]

3 Evolutionary algorithms

Evolutionary algorithms attempt to solve complex problems by simulating the processes of Darwin. They develop a set of solutions to a problem with the aim of finding the best results. Since they use random processes iteratively, evolutionary algorithms are never guaranteed to find an optimal solution for any problem, but they will often find a good solution if one exists [1].
The main idea behind evolutionary algorithms is to evolve an initial population in a natural selection process to that ensure the rise in the fitness of the population. Based on the fitness, candidates are selected to seed the next generation by applying recombination and/or mutation to them, this selection process favor the solution with the best objective function value. Recombination operator will generate new candidate (children) using two or more candidates (parents), the children inherit the characteristic of the parent. Mutation, in the other side, will be applied to only one candidate and it will generate a new one. Once new candidate are created, and based on the new fitness values, the replacement operator will determine which individuals are to be removed from the population, keeping only the candidate with highest fitness [2].

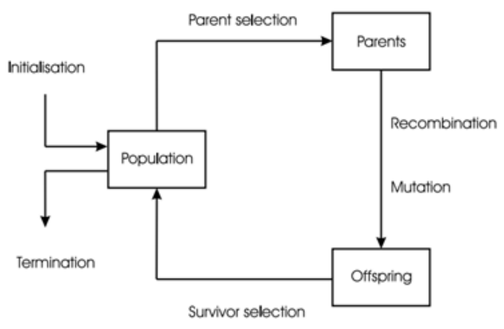


FIGURE 2 – Evolutionary algorithm have a number of components, procedures or operators [Reference 2]

Evolutionary algorithm have a number of components, procedures or operators [Reference 2] :

— Representation : definition of an individual ;
— Evaluation function (fitness function) ;
— Population ;
— Parent selection mechanism ;
— Variation operators, recombination and mutation ;
— Survivor selection mechanism (replacement).

4 Indicator-Based Selection in Multiobjective Search

4.1 Discussion

The Evolutionary algorithms « EA » are well adapted to optimization problems with multiple objectives (2 or more than objective), for many reasons as : i) the initial population is selected randomly ii) they are able to capture a number of solutions in the same time in a single run.

The main idea behind The paper «Indicator-Based Selection in Multi-objective Search » is to propose a general indicator-based evolutionary algorithm « IBEA » that can define the optimization goal in term of an indicator (measure) then use this measure in the selection process [1] which is one of the main steps of evolutionary algorithms.

The principal objective of multi-objective optimization is to find the set of Pareto (set of non-dominated solutions, see figure1). The good Pareto set depends on the user's pereferences, otherwise no objective can be improved without sacrificing at least one other objective.

The classical MOEAs algortihms as : Non-Dominated Sorting Genetic Algorithm (NSGA-II) [2] and Strength Pareto Evolutionary Algorithm (SPEA2) [3] have both advantages and disadvantages. The first has a high operating efficiency and good convergence, but poor distribution of Pareto solutions set obtained in optimizing the high-dimensional objective function ; the second one is able to obtain a well distributed Pareto optimal set,but is low running efficiency and time-consuming[4]. Therfrom the need to an algorithm who incorporate preference criteria in multi-objective optimization and in this case it's the IBEA.

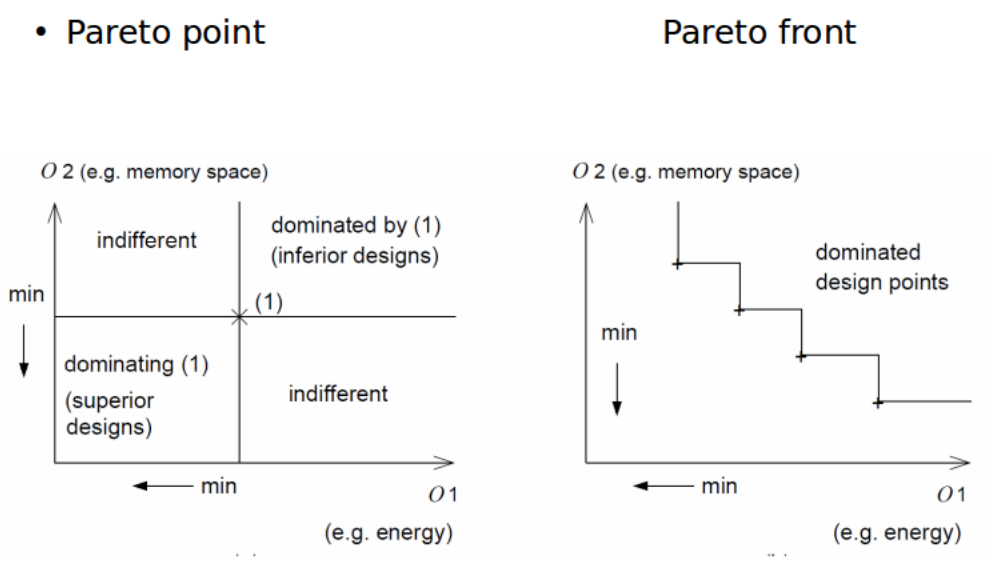


Figure 3 – Pareto points and Pareto front [Karl Lieberherr : University of Dortmund]

To sum up, our paper is an IBEA where the indicator is a hyper-volume indicator (quantitaive measure which gives the volume between the estimated Parteo and the reference point R) and where the preference criteria is always incorporated based-on dominance relations. The adaptive IBEA proposed contain the following six steps :

- **STEP1 :**
  The 1st step of the algorithm consist of generation an initial population of size α of n-dimension vectors denoted as x.
- **STEP2 :**
  In the 2nd step, we scale the objectives. For each objective ϕ, we determinate the lower and upper bounds :

  $$\underline{b_i} = min_{x\in P} f_i(x); \quad \overline{b_i} = max_{x\in P} f_i(x) \tag{4}$$

  We scale each objective to the interval [0,1].

  $$f_i^{'}(x) = (f_i(x) = -b_i)/(\overline{b_i} - \underline{b_i}) \tag{5}$$

  we use the new scaled objectives to calculate the indicators values and determine the maximum absolute indicator value.
  Finally, we calculate the fitness values :

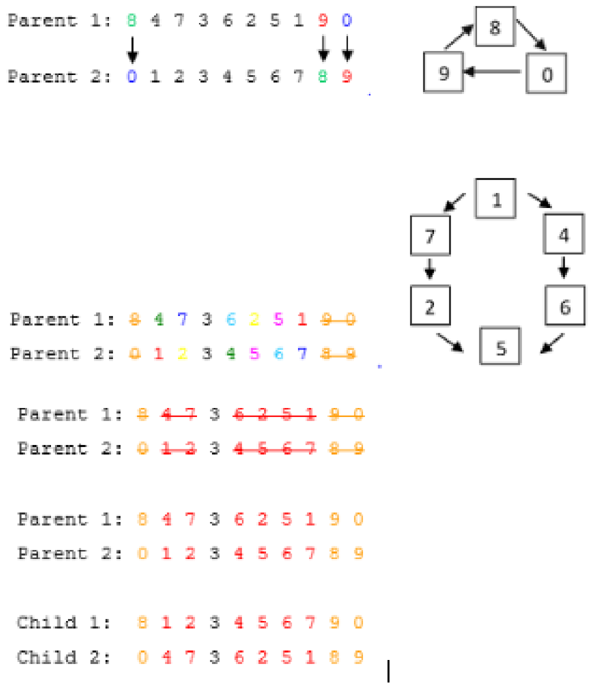
  $$\text{for all} \quad x^1 \in P, F(x^1) = \sum_{x^2 \in P/x^1} exp^{-i.(x^2.x^1)/exp^{-k}} \tag{6}$$
- **STEP 3 :**
  The third step is the environmental selection in which we eleminate the individuals with the a low fitness value and keep only the alph individuals with highest fitness. The fitness values are then updated fot the remaining individuals.

— **STEP 4 :**
In this step we check if any stopping criteria is satisfied, if so, set A to the set of decision vectors represented by the non-dominated individuals in P and stop.

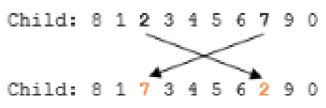
— **STEP 5 :**
In the selection process we determinate the individuals that will participate in the reproduction. Individuals with the best performance are selected more often than others because of their fitness. For this algorithm, we use the binary tournament selection that consist on picking 2 individuals randomly, the best individual in term of fitness will be placed in a temporary mating pool P'. We repeat this process until we obtain α parents. These parents will be used in in the follow step to generate new.

— **STEP 6 :**
In this last step we perform on the mating pool P' two operators : recombination (crossover) that produce one or more children from two (or more) parents and mutation which will reproduce a new individual from a single one. Those two operations help the algorithm to find better solution than those shown in its current population. However, no further details are mentioned in the paper about which algorithm to use for both operators. We found 22 crossover algorithm and 6 mutation algorithm, to proceed in this step we had to choose an algorithm for each operation :

1. crossover : we choose theTEXCycle Crossover (CX). This algorithm is known to preserves the information in both parents. In this algorithm, we have to define the cycle defined by the parent's chromosomes. Theses cycles are used to create the children, the 1st one is copied into the 1st child, the 2nd cycle is copied into the 2nd child and so on.


Parent 1: 8 4 7 3 6 2 5 1 9 0
Parent 2: 0 1 2 3 4 5 6 7 8 9
Parent 1: 8 4 7 3 6 2 5 1 9 0
Parent 2: 0 1 2 3 4 5 6 7 8 9
Parent 1: 8 4 7 3 6 2 5 1 9 0
Parent 2: 0 1 2 3 4 5 6 7 8 9
Parent 1: 8 4 7 3 6 2 5 1 9 0
Parent 2: 0 1 2 3 4 5 6 7 8 9
Child 1: 8 1 2 3 4 5 6 7 9 0
Child 2: 0 4 7 3 6 2 5 1 8 9

2. Mutation : we chose the 1-Opt ou Swap mutation. It consist on randomly picking to chromosomes from the individual and swap their positions.



5 Progress

5.1 What have been done

This section introduces our works and the steps completed in our optimization project.

First of all, we understood how Coco platform functions and how to run our algorithm by using the example **example_experiment.py**. In detail, we replaced the function **random_search** in this example and its call to our function **Hypervolume_Indicator_Based_Selection_MultiobjectiveSearch** and also post-processed the example data.

Secondly, we read our article and did researches about the algorithms presented in the paper. Besides Indicator Based Evolutionary Algorithm with Hypervolume concept, we researched Binary Tournament Selection Algorithm and selected the algorithms for Recombination and Mutation operators namely Cycle Crossover and Swap Mutation. Moreover, we chose their parameters based on documents found.

Thirdly, we implemented our algorithm which separated in one main function and eight sub-functions. Each step has its own functions :

Our main function is **Hypervolume_Indicator_Based_Selection_Multiobjective_Search** receiving bbob-biobj suite problems, their bounds and budget. We provided also population size depending on the bounds, fitness scaling factor k (0.05), maximum number of generations (10) and reference point Z(2,2). This function returns Pareto set approximation.

In first step Initialization, we generate initial population P with size “alphaValuePopulationSize” and set “mValueCounter” to 0. An array F is generated by Problem in bbob-biobj suite (fun()). We used F = func(P) with two objectives to calculate Pareto set.

```
initialPopulationP = lbounds + (ubounds - lbounds) $\times $ np.random.rand(
    chunk, dim); \\
F = [fun(x) for x in initialPopulationP]; \par

```

In second step Fitness Assignment, we calculated fitness values of individuals in P with function **fitness_assignment(F - pArray, reference point Z, fitness scaling factor)**”. Fitness value is calculated by using Hypervolume Indicator with function **indicator_value** (x1 , x2, reference point Z)”. The function “fitness_assignment” returns scaled array of F, hypervolume indicator values array, fitness value of each individual “fitnessArray” and the maximum absolute indicator value C.

```
initialPopulationP = lbounds + (ubounds - lbounds) * np.random.rand(chunk, dim
    );
```

```
F = [fun(x) for x in initialPopulationP];
```

python1.py

In the third step Environmental Selection, the function **environmental_selection**(initial Population P, F scaled array, hypervolume indicator values array, fitness values Array, alpha Value, maximum absolute indicator value C , fitness scaling factor) returns individuals having the best fitness values from F and P.

```
"
while (len(pArray) > alphaValue):
minFitnessIndex = np.argmin(fitnessArray);
pArray=np.delete(pArray,minFitnessIndex,0)
for i in range(len(fitnessArray)):
for j in range(len(fitnessArray)):
fitnessArray\[i\] = fitnessArray\[i\] + np.exp((-indicatorArray\[
    minFitnessIndex,j\]) / (cValueMaxIndicator * kValue)); initialPopulationP=
    np.delete(initialPopulationP,minFitnessIndex, 0); fitnessArray=np.delete(
    fitnessArray,minFitnessIndex, 0);
indicatorArray = np.delete(np.delete(indicatorArray,minFitnessIndex,1),
    minFitnessIndex,0);"
```

In fourth step Termination, we set the output of function "**non_dominated_selection(initial Population P, indicator Array)**" as Pareto set approximation, which means that our output are non-dominated individuals.

```
for i in range(indicatorArray.shape\[1\]):
if(indicatorArray.min(0)\[i\]<0):
listDominatedPoint.append(i)
paretoSetApproximation=np.delete(paretoSetApproximation,listDominatedPoint,0)
indicatorArray=np.delete(indicatorArray,listDominatedPoint,0);
indicatorArray=np.delete(indicatorArray,listDominatedPoint,1)
```

In fifth step Mating Selection, the function "binary_tournament_selection(initial Population P, F Array, fitness Array)" selects two individuals for binary tournament, then the winner is put in Parent Population until alpha = 10 individuals chosen.

```

parentPopulationCounter = 0;

while parentPopulationCounter < maxParentPopulation:

parentIndex =  np.random.randint(len(fitnessArray), size= 2);

   if(fitnessArray\[parentIndex\[0\]\] <= fitnessArray\[parentIndex\[1\]\]):
     parentPopulation\[parentPopulationCounter\] = initialPopulationP\[
     parentIndex\[1\]\];
```

```
  else:

parentPopulation\[parentPopulationCounter\] = initialPopulationP\[parentIndex
    \[0\]\];

parentPopulationCounter += 1;

```

In sixth step Variation, the function “variation(initial Population P, parent Population)” has two sub-functions “recombination(parent Population)” and “mutation(baby Population)”. Two parents are mated for creating two new babies having the features of their parents (Cycle Crossover Algorithm). Each baby mutates by swapping randomly its two gene values (Swap Mutation Algorithm). The swapping possibility depends on a threshold chosen (Swapping possibility < 0.001). After variation step, we will comeback to the second step for re-calculating fitness value of these new individuals.

Recombination :

```

for i in range(len(parentPopulation)):
for j in range(i+1,len(parentPopulation)):
if (i != j) :
geneIndex = 0;
while geneIndex <= len(parentPopulation\[0\]) :
baby1\[geneIndex-1\] = parent1\[geneIndex-1\];
baby2\[geneIndex-1\] = parent2\[geneIndex-1\];
if (geneIndex % localisationValue) == 0 :
temporalParent = np.array(parent1);
parent1 = np.array(parent2);
parent2 = np.array(temporalParent);
geneIndex += 1;
recombinationBabyPopulation = np.append(recombinationBabyPopulation, \[baby1
    \], axis = 0);
recombinationBabyPopulation = np.append(recombinationBabyPopulation, \[baby2
    \], axis = 0);
```

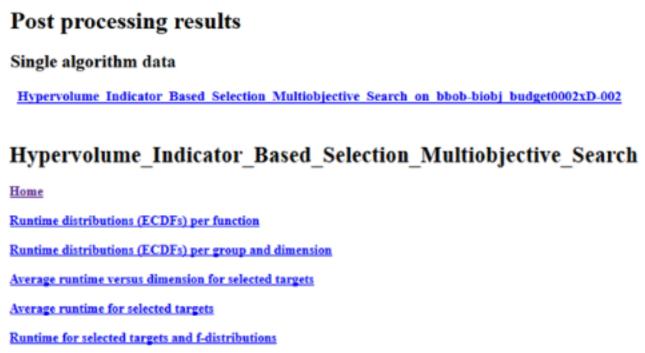
Mutation :

```
           "
possibilityThreshold = 0.001;
for baby in babyPopulation:
possibility = np.random.random();
if (possibility < possibilityThreshold) :
index1 = np.random.randint(len(babyPopulation\[0\]));
index2np.random.randint(len(babyPopulation\[0\]));
if(index1 != index2):
                 temporalValueBaby = baby\[index1\];  \\
                 baby\[index1\] = baby\[index2\]; \\
                 baby\[index2\] = temporalValueBaby
"
```

Finally, we did post-process our data received.

5.2 Results

After having done post-processing our data received, we have an html file “index.html” containing the post-processing results. In effect, we have “*.svg” files representing our data as images in our report and providing an intuitive view.



Post processing results

Single algorithm data

Hypervolume_Indicator_Based_Selection_Multiobjective_Search_on_bbob-biobj_budget0002xD-002

Hypervolume_Indicator_Based_Selection_Multiobjective_Search

Home

Runtime distributions (ECDFs) per function

Runtime distributions (ECDFs) per group and dimension

Average runtime versus dimension for selected targets

Average runtime for selected targets

Runtime for selected targets and f-distributions

FIGURE 4 – The interface of report file "index.html"

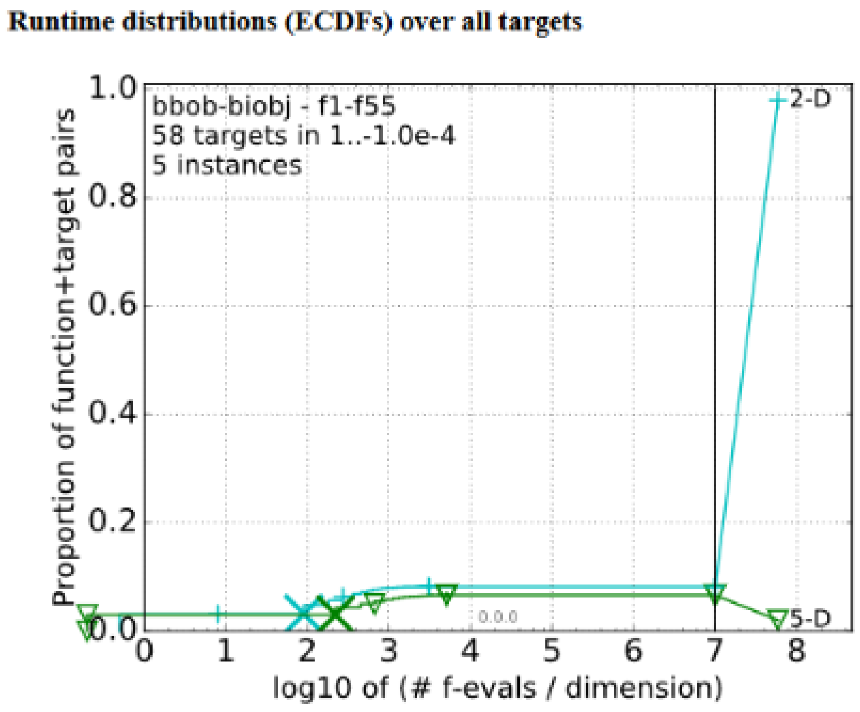


FIGURE 5 – Runtime distributions (ECDFs) over all targets

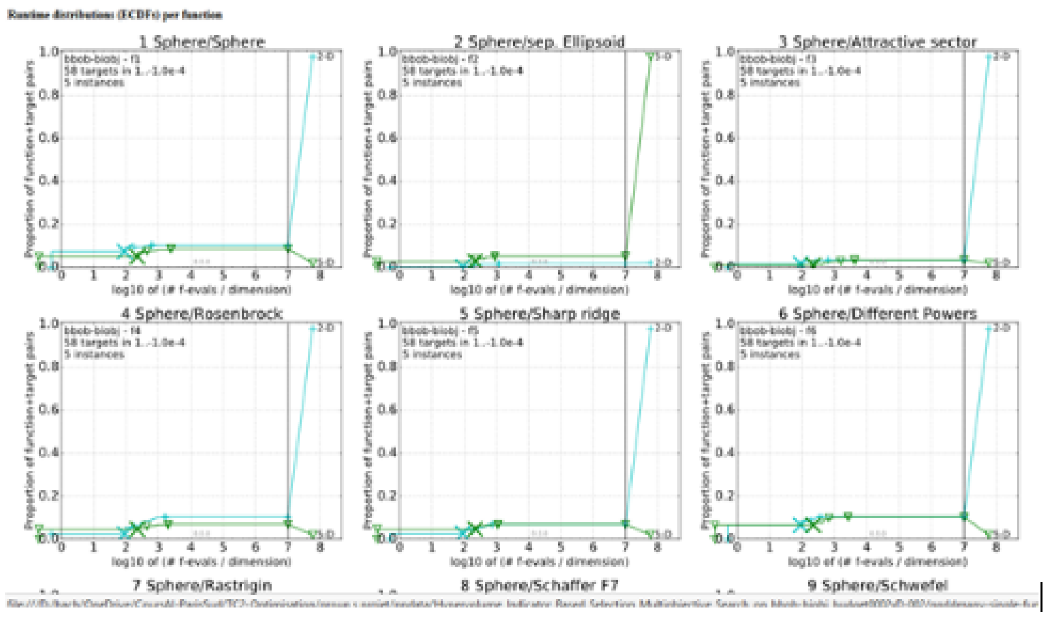


FIGURE 6 – Runtime distributions (ECDFs) per function

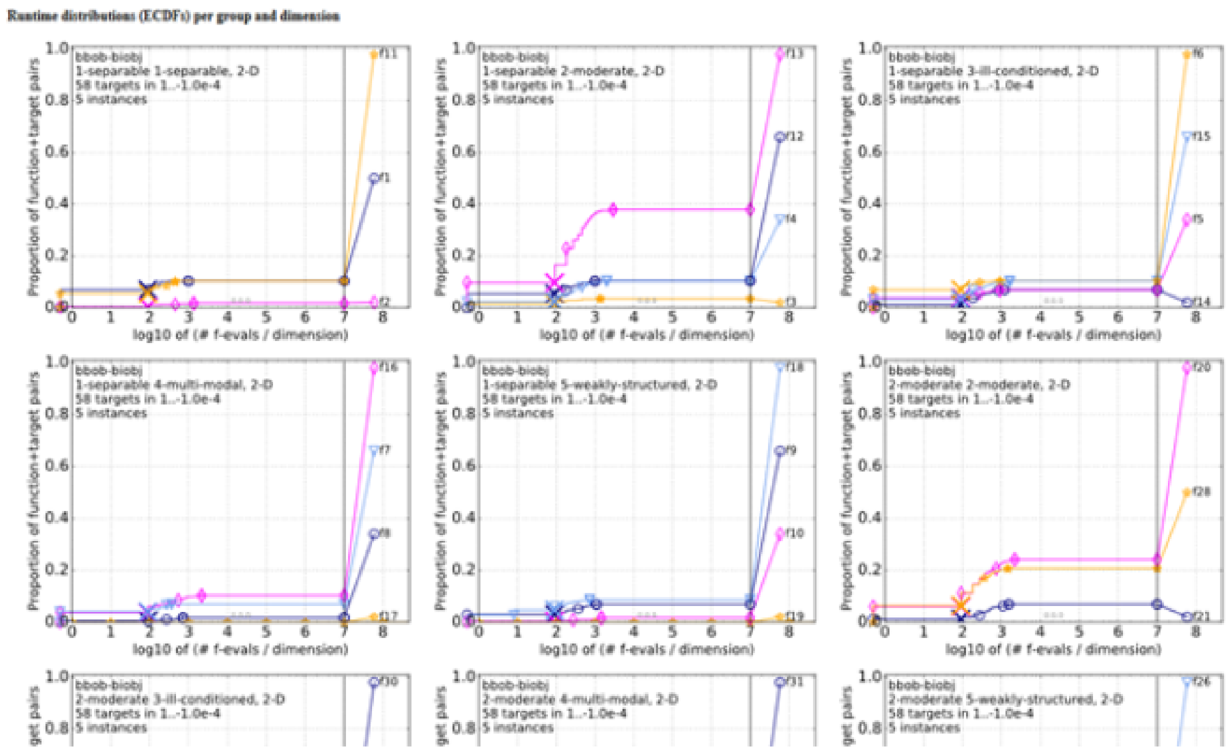


FIGURE 7 – Runtime distributions (ECDFs) per group and dimension

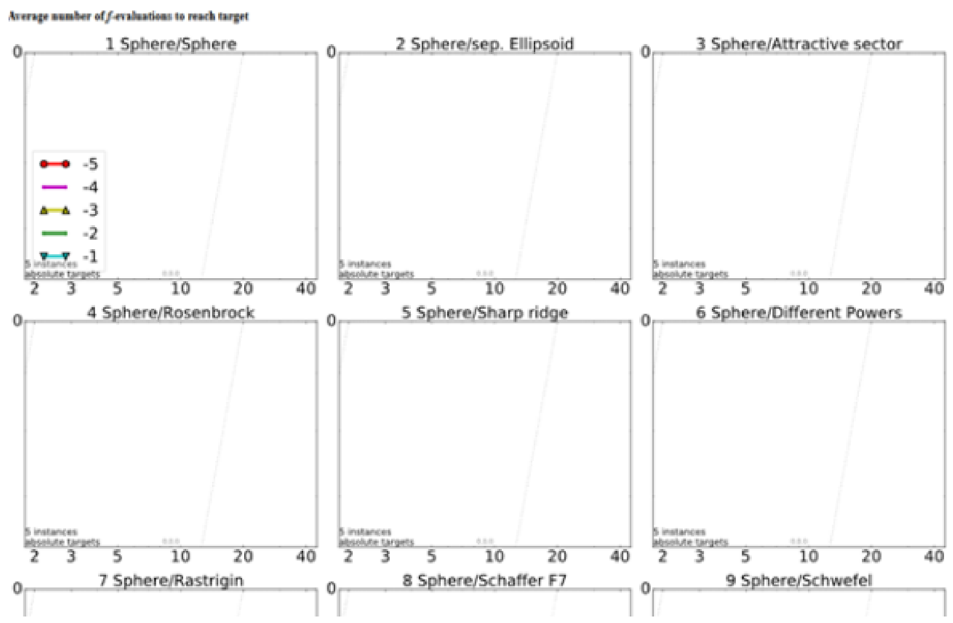
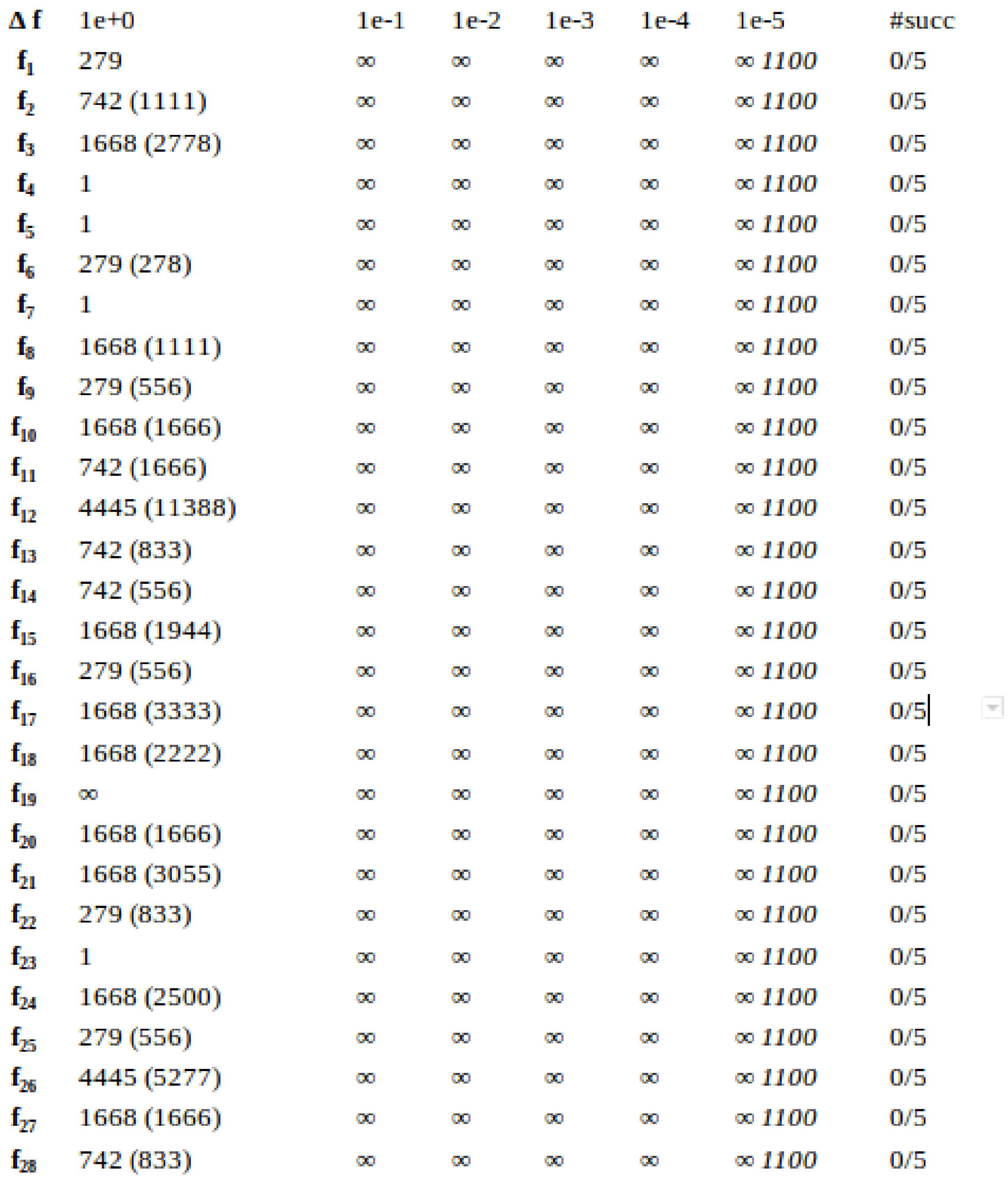


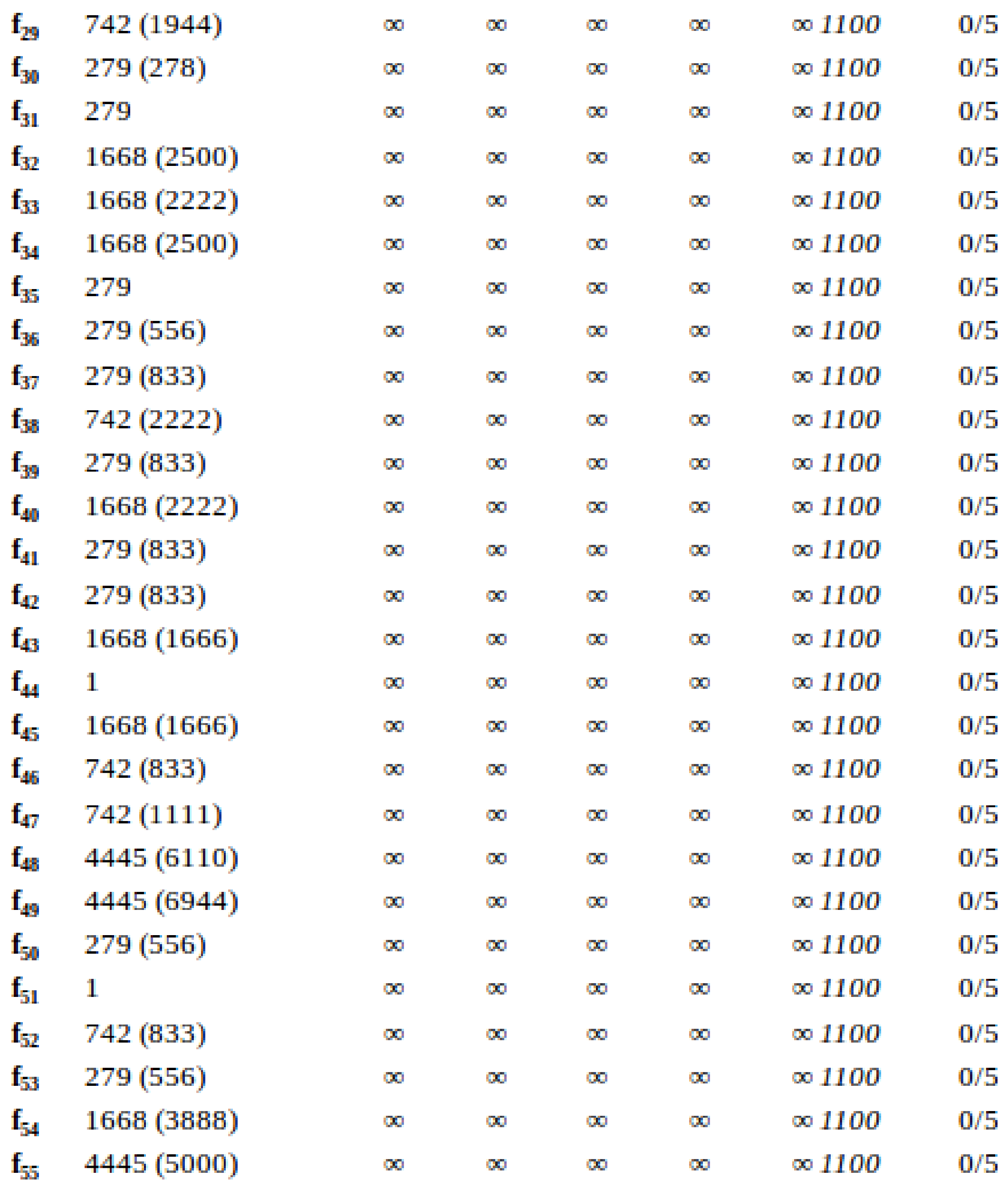
FIGURE 8 – Average number of f-evaluations to reach target

aRT in number of function evaluations

Average runtime (aRT) to reach given targets, measured in number of function evaluations. For each function, the aRT and, in braces as dispersion measure, the half difference between 10 and 90 %-tile of (bootstrapped) runtimes is shown for the different target Δf-values as shown in the top row. #succ is the number of trials that reached the last target $I^{red} + 10^{-5}$. The median number of conducted function evaluations is additionally given in italics, if the target in the last column was never reached.



Δ f	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
f_1	279	∞	∞	∞	∞	∞ *1100*	0/5
f_2	742 (1111)	∞	∞	∞	∞	∞ *1100*	0/5
f_3	1668 (2778)	∞	∞	∞	∞	∞ *1100*	0/5
f_4	1	∞	∞	∞	∞	∞ *1100*	0/5
f_5	1	∞	∞	∞	∞	∞ *1100*	0/5
f_6	279 (278)	∞	∞	∞	∞	∞ *1100*	0/5
f_7	1	∞	∞	∞	∞	∞ *1100*	0/5
f_8	1668 (1111)	∞	∞	∞	∞	∞ *1100*	0/5
f_9	279 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{10}	1668 (1666)	∞	∞	∞	∞	∞ *1100*	0/5
f_{11}	742 (1666)	∞	∞	∞	∞	∞ *1100*	0/5
f_{12}	4445 (11388)	∞	∞	∞	∞	∞ *1100*	0/5
f_{13}	742 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{14}	742 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{15}	1668 (1944)	∞	∞	∞	∞	∞ *1100*	0/5
f_{16}	279 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{17}	1668 (3333)	∞	∞	∞	∞	∞ *1100*	0/5
f_{18}	1668 (2222)	∞	∞	∞	∞	∞ *1100*	0/5
f_{19}	∞	∞	∞	∞	∞	∞ *1100*	0/5
f_{20}	1668 (1666)	∞	∞	∞	∞	∞ *1100*	0/5
f_{21}	1668 (3055)	∞	∞	∞	∞	∞ *1100*	0/5
f_{22}	279 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{23}	1	∞	∞	∞	∞	∞ *1100*	0/5
f_{24}	1668 (2500)	∞	∞	∞	∞	∞ *1100*	0/5
f_{25}	279 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{26}	4445 (5277)	∞	∞	∞	∞	∞ *1100*	0/5
f_{27}	1668 (1666)	∞	∞	∞	∞	∞ *1100*	0/5
f_{28}	742 (833)	∞	∞	∞	∞	∞ *1100*	0/5



f_{29}	742 (1944)	∞	∞	∞	∞	∞ *1100*	0/5
f_{30}	279 (278)	∞	∞	∞	∞	∞ *1100*	0/5
f_{31}	279	∞	∞	∞	∞	∞ *1100*	0/5
f_{32}	1668 (2500)	∞	∞	∞	∞	∞ *1100*	0/5
f_{33}	1668 (2222)	∞	∞	∞	∞	∞ *1100*	0/5
f_{34}	1668 (2500)	∞	∞	∞	∞	∞ *1100*	0/5
f_{35}	279	∞	∞	∞	∞	∞ *1100*	0/5
f_{36}	279 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{37}	279 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{38}	742 (2222)	∞	∞	∞	∞	∞ *1100*	0/5
f_{39}	279 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{40}	1668 (2222)	∞	∞	∞	∞	∞ *1100*	0/5
f_{41}	279 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{42}	279 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{43}	1668 (1666)	∞	∞	∞	∞	∞ *1100*	0/5
f_{44}	1	∞	∞	∞	∞	∞ *1100*	0/5
f_{45}	1668 (1666)	∞	∞	∞	∞	∞ *1100*	0/5
f_{46}	742 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{47}	742 (1111)	∞	∞	∞	∞	∞ *1100*	0/5
f_{48}	4445 (6110)	∞	∞	∞	∞	∞ *1100*	0/5
f_{49}	4445 (6944)	∞	∞	∞	∞	∞ *1100*	0/5
f_{50}	279 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{51}	1	∞	∞	∞	∞	∞ *1100*	0/5
f_{52}	742 (833)	∞	∞	∞	∞	∞ *1100*	0/5
f_{53}	279 (556)	∞	∞	∞	∞	∞ *1100*	0/5
f_{54}	1668 (3888)	∞	∞	∞	∞	∞ *1100*	0/5
f_{55}	4445 (5000)	∞	∞	∞	∞	∞ *1100*	0/5

Empirical cumulative distribution functions (ECDF)

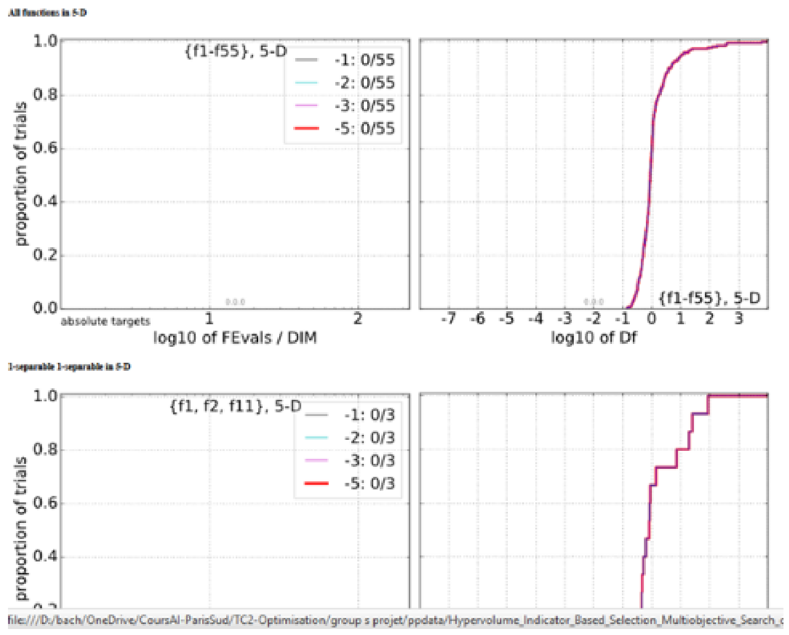


FIGURE 9 – Empirical cumulative distribution functions (ECDF)

6 Problems faced

— IUnderstanding all the concepts included in the article like multiobjective optimization. It took us some time an couple of meeting for us to be on the same wavelength.
— The paper is not very detailed about what recombination and mutation to use. As stated before, we found 20 algorithms for recombinition and 5 for mutation and it's hard to test all of them so we had to discuss them and pick one in each category.

7 What remains to do ?

— Try to test our algorithm using an other recombinting algorithm if possible
— Run more tests and update our algorithm for better results
— Prepar the final paper and presentation

8 References

[1] Mishra, B. S. P., Dehuri, S., Kim, E. (Eds.). (2016). Techniques and Environments for Big Data Analysis : Parallel, Cloud, and Grid Computing (Vol. 17). Springer.

[2] Eiben, Agoston E., and James E. Smith. Introduction to evolutionary computing. Vol. 53. Heidelberg : springer, 2003.

[3] Zitzler, E., Künzli, S. (2004, September). Indicator-based selection in multiobjective search. In International Conference on Parallel Problem Solving from Nature (pp. 832-842). Springer Berlin Heidelberg.

[4] Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., Schwefel, H. P. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization : NSGA-II, Paris, France, 2000.

[5] Zitzler, E., Laumanns, M., Thiele, L. (2001, May). SPEA2 : Improving the strength Pareto evolutionary algorithm. In Eurogen (Vol. 3242, No. 103, pp. 95-100).

[6] Liu, H. C., Sung, W. P., Yao, W. (Eds.). (2014). Computer, intelligent computing and education technology. CRC Press.