EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
EN-STI-FDA

Analysis of Various Multi-Objective Optimization Evolutionary Algorithms for Monte Carlo Treatment Planning System

Summer Student Report

Author
M. TYDRICHOVA

Supervisor
W. Kozłowska

August 4, 2017, Geneva

Introduction

The principal goal of this project was to analyze various multiobjective optimization algorithms (MOEA) for the Monte Carlo Treatment Planning System. After doing literature research on testing and quality evaluation methods, the comparison of available frameworks was done to select the most suitable one for our test needs. Then the software for testing was developed, we tried to provide a solid objective-oriented architecture to allow easy modifications in the future, another high priority was to create an user-friendly software.

From all available algorithms, candidates for testing were selected. Apart results from the literature that were taken into account to do this selection, we placed emphasis on a diverse scale of algorithms - the aim was not only to find the best performing algorithm, but also to describe the comportment, advantages and weak points of different types of algorithms, in hope that this output could be helpful and valuable for the further research.

In performed tests, we focused mostly on the quality of DOSE and LET values of obtained solutions, as the main goal of the problem was to minimize these quantities. Distribution of solutions and population size were also analyzed. Another important criterion was the reliability of the algorithm. That means, we studied if the algorithm performed the same way in all seeds and if it was independent on a given data set. To facilitate this task, the user friendly GUI for analysis was developed.

As the optimal solution is not known, the majority of available quality metrics could not be used. For this reason, we tried to define our own variants of some metrics independent on the optimal solution and primarily, a high importance was assigned to understanding of dependencies between different indicators to provide more complex and exhaustive results.

After analyzing all algorithms, the best of them were identified and recommended for use. Regarding the others, their weak points were identified and things to be improved in further work were proposed.

1 Methodology

1.1 Algorithm selection

In the literature, many studies on multiobjective evolutionary algorithms (MOEA) and their performance can be found. However, their results never provides the definitive answer about algorithm performance. In fact, algorithms are tested on multiobjective problem (MOP) test suites. Despite multiple attempts to create an exhaustive MOEA test suite, there are always some shortcomings. The test results only suggest that the algorithm would perform well on the type of problems included in the test suite. They do not assure the good performance of algorithm on a given "real-world" problem. On the other hand, any algorithm which is supposed not to perform very well can provide really good results in some particular case.

What is more, our problem is quite specific one, at least in the number of variables. While the commonly used, state-of-art MOPs have at maximum several dozens of variables, the both data sets of our problem used for tests have almost 7000 variables.

For this reason, while known information from the literature can give an indication of algorithm performance, we should not rely on it and should test as many algorithms as possible.

At the beginning, 15 algorithms were "pretested". Several runs of each algorithm with 10 iterations were performed to get the first idea about algorithm comportment on the given problem. The original idea was to order algorithms from the most promising to the least promising ones, and test them in this order. Unfortunately, due to the huge number of variables, tests are computationally expensive, and only 8 algorithms were tested.

That is why we finally decided to ensure the diversity of tested algorithms. The algorithms can be divided into several groups according to their strategies; for example, there are algorithms using evolutionary strategy, particle-swarm optimization based algorithms, classical Pareto-based algorithms etc. Within each group algorithms are often quite similar or there are different versions and improvements of the same original algorithm. Thus, we can expect that the comportment of algorithms will differ more between groups than within one group. The selection across all groups offers a possibility to observe the performance of different types of algorithms, to give us a general idea about their comportment on the problem and to help us to decide if we would like to test more similar algorithm in the future, or if this type of algorithms seems to be inappropriate for our purpose.

We selected two particle-swarm optimization based algorithms which are **OMOPSO** and **SMPSO**. The OMOPSO algorithm is inspired by the behavior of flocks of birds: a population is "flown" through the search space, guided by leader solutions which are selected from the non-dominated population using principally NSGA-II mechanisms. SMPSO was created to improve OMOPSO. The main difference between both algorithms is that, in SMPSO we constraint the dis-

tance traveled by a solution in one iteration. In fact, in OMOPSO, it happens that some solutions are moving too far in the space so they escape from the swarm and do not contribute on the swarm progress. According to the literature, these algorithms are among the best ones. This seemed to be true also in the executed pretests. There are no more PSO included in MOEAFramework, but there exist many others PSO-based approaches and variations of mentioned algorithms that could be implemented.

PAES was selected to represent algorithms using evolutionary strategy, despite the fact that this algorithm did not perform well in comparison with the others during pretests. On the other hand, it is very fast. It follows the (1+1) evolution strategy, meaning that a single parent generates a single offspring through mutation.

Classical MOEAs can be generally divided into three groups:

1. Pareto dominance-based approaches
2. Decomposition-based methods
3. Indicator-based algorithms

From the first group, **SPEA2** and **PESA-II** were selected. In PESA-II, the search space is divided into hyperboxes of fixed size in order to select parents from the low populated ones. This approach should provide more diversity. In the literature, this algorithm is considered to perform better than other algorithms from this group, such as SPEA or NSGA.

From the second group mentioned above, **MOEA/D** and **RVEA** were selected. These algorithms are based on decomposition. In MOEA/D, the MOP is decomposed into sub-problems. The sub-problems are then solved simultaneously by evolving the population of solution. In each generation, the population is composed by the best solutions founds so far for each sub-problem. There exists several variants of this algorithms, and especially many different strategies of decomposition. RVEA also decomposes the search space into a number of subspaces using the reference vectors. Our motivation is to optimize the problem in the direction of each vector. In out pre-tests, MOEA/D seemed to give the results slightly better than average, while RVEA performed very well. Other algorithms from this category are for example MOEA/DD, MOEA/D-M2M, DBEA or NSGA-III-OSD.

Finally, we selected **SMS-EMOA** to have a representative of the third group. In this algorithm, the current population is primarily evaluated in the same manner as NSGA-II. The individuals with the worst rank are then compared using the hypervolume metric, the worst among them is removed from the population. According to the literature, it seems that this algorithm can often outperform those using Pareto dominance-based approaches. The algorithm performed well in the pretests, despite the fact that the solutions tended to vary quite a lot. On the other hand, algorithm was very fast in comparison with its opponents. Other algorithms from this group are for example IBEA or SIBEA.

There are other approaches such as epsilon-dominance-based algorithms, differential strategy etc. None of these algorithms were tested.

1.2 Quality indicators selection

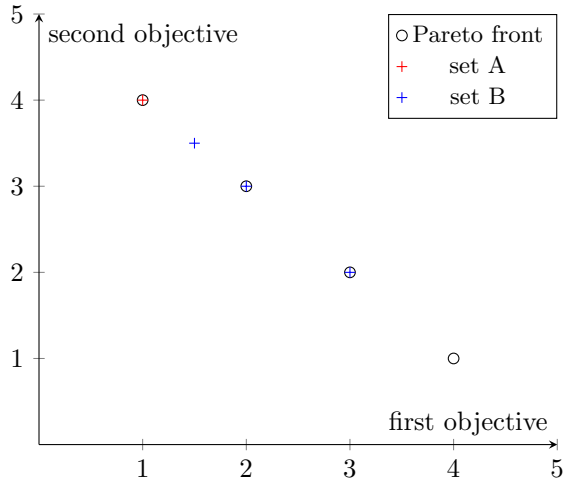
There are many available metrics to measure convergence or diversity of MOEAs. However, for the most of them, the knowledge of the true Pareto front is necessary. As we are facing a real-world problem, we do not know and we will probably never know the true Pareto front (if we knew the true Pareto front, there would be no need to optimize the problem using MOEAs). Thus, we are limited on a small number of quality indicators.

1.2.1 Relative generational distance

Regarding the convergence, we can not measure it without having a set we desire to converge to. Even though, we can obtain some general ideas about algorithm convergence comportment. We are inspired by the convergence metric called **generational distance**. This metrics calculates, after each iteration, the distance of the approximation set from the true Pareto front. The smaller its value is, the nearer the approximation set is to the Pareto front. More formally, the generational distance is calculated by formula

$$GD = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d_i^2}$$

where N is the population size and d_i the (most currently Euclidean) distance in objective space between the i-th solution and its closest neighbor from the Pareto front. We should remark that the zero value does not mean that we have found the Pareto front. It only means that the approximation set is contained in it. So, the smaller generational distance does not necessarily mean the better result, as shown on the following plot:



As said before, in our case, we can not measure the convergence to the Pareto front, but we can calculate the distance between two successive generations. This value is obtained by the same formula as the generational distance replacing the known fixed Pareto front by an another approximation set. That is why we called it **relative generational distance**. If the relative generational distance is near to zero, population has not changed from one iteration to the other. If the suite of relative generational distance values converge to zero, we can conclude that the population converges. However, it is important to say that we do not know if the set converges to its Pareto front or not, and there is no way to answer this question. We neither do not know if the population is improving during the execution - we can just see if it is moving, but we do not know towards which direction.

Nevertheless, this information can be helpful - for example, we can find out if there exists a sufficient number of iterations after which an algorithm usually converges. We can also describe the convergence type of an algorithm, which can help us to define new termination conditions.

To be accurate, here we should also mention, that if the relative generational distance (RGD) equals zero, it does not necessarily mean that the both populations are the same. In fact, one population can be included in another. We should see that even though this indicator is called distance, it is not really a distance from the mathematical point of view - there is not a symmetry. We calculate the relative generational distance for one population *with reference* to another. Let's have two sets, $A \subset B$. RGD value of A with reference to B is not the same as the RGD value of B with reference to A. Thus, we should always calculate the reference value of the current iteration population with the reference to the previous iteration population to really observe the progress, and also to prevent the case when the population is continuously growing, but we do not recognize this improvement due to the zero value of RGD.

1.2.2 Spacing

Regarding the diversity of solutions, we can use **spacing** metrics which does not require the Pareto front. This metrics tells us how uniformly the solutions are distributed in the objective space. In fact, it measures the variance of the distance of each solution to its closest neighbor. This can be formalized as:

$$s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{d} - d_i)^2}$$

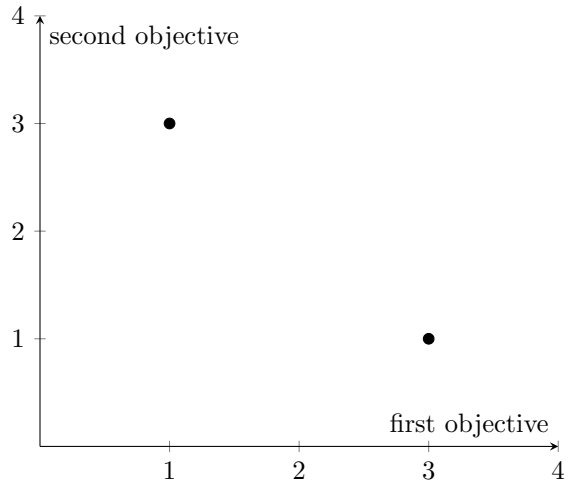
where

$$d_i = \min_{j \in \{1,\dots,N\},\ j \neq i} \|f^i - f^j\|_1$$

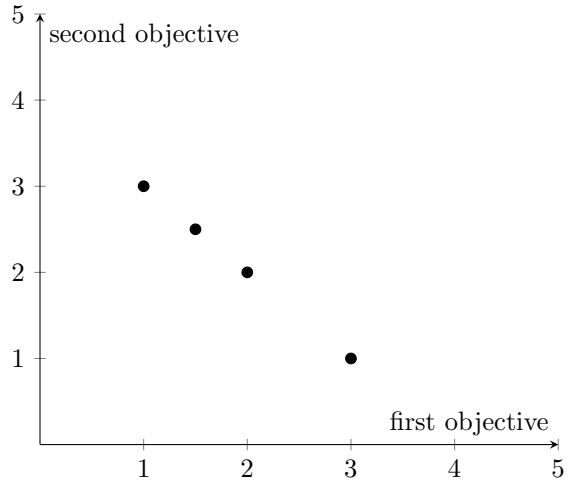
is the distance of i-th solution f^i to its closest neighbor and

$$\bar{d} = \frac{1}{N}\sum_{i=1}^{N} d_i$$

is an average distance from solution to its closest neighbor, N is the population size. As we recognize the normalized Euclidean distance in the first formula, this metric can be interpreted geometrically as the normalized distance between a vector $(\bar{d}, \dots, ...\bar{d})^T$ and $(d_1, \dots, d_N)^T$. The normalization should assure the independence on the population size. We see that if the solutions are perfectly uniformly distributed, $s = 0$. However, it does not always necessarily mean that a smaller spacing is better. For illustration, let's consider the following example:



Example 1: s = 0



Example 2: spacing > 0

In the second case, the spacing is bigger than 0. On the other hand, the first case solution is strictly included in the second one. For this reason, the second case should be considered as a better one. In particular, if the population consists of one or two individuals, the spacing will always be 0. As for generational distance, also spacing should be regarded in a wider context while analyzing results. As shown here, a small spacing can signify that an algorithm generates very small populations concentrated on a small space, so another algorithm generating bigger and more variable population can better fulfill our requirements.

2 Testing software

2.1 Available frameworks

There is quite a lot of software packages and frameworks to develop or test/compare MOEAs. In this section, we will first give the list of their main features, then try to choose the one which fits the best for our purpose. As the aim of the project is to compare MOEAs, software for the development (but not the comparison) of MOEAs will not be mentioned in details. These are for example Open Beagle or OPT4J.

2.1.1 MOEAFramework

Homepage: `http://moeaframework.org/`
Portability: written in Java, problems written in C/C++ can be defined.
Algorithms provided: The MOEAFramework has the largest collection of EA and MOEAs of any library. There are also others meta-algorithms, non-EA for multi-objective optimization. In addition to these pre-defined algorithms, new algorithms can be easily constructed using existing components.
MOPs: All major problems from the literature are available.
Performance indicators: All the main metrics are also implemented.
Comparison tools: Many tools for analyze and compare MOEAs, such as GUI for quickly comparing the performance, statistically compared results, sensitivity analysis tool etc.
Development tools: New algorithms can be easily constructed using existing components. Also, new MOPs written in Java or other languages can be easily incorporated.
Documentation: Very well documented, a lot of tutorials, active community.

Notes, remarks: still developed

2.1.2 JMetal 5

Homepage: `https://jmetal.github.io/jMetal/`
Portability: written in Java
Algorithms provided: Many (not only EA) multi-objective algorithms. These are NSGA-II, SPEA2, PAES, PESA-II, OMOPSO, MOCell, AbYSS, MOEA/D, GDE3, IBEA, SMPSO, SMPSOhv, SMS-EMOA, MOEA/D-STM, MOCHC, MOMBI, MOMBI-II, NSGA-III, WASF-GA, GWASF-GA
MOPs: All major problems families (ZDT, DTLZ, WFG, CEC2009, LZ09, GLT, MOP) such as classical problems (Kursawe, Fonseca, Schaffer, Viennet2, Viennet3) or problems, combinatorial and academic problems
Performance indicators: Classical metrics implemented, slightly less than in MOEAFramweork.
Comparison tools: Since version 5.1, jMetal incorporates support for making experimental studies, i.e. configuring an experiment where a set of algorithms solve a number of problems and, as a result, a number of output files (latex tables, R scripts) are generated.
Development tools: Classes, templates and patterns to ease the development of new algorithms.
Documentation: Very well documented, with code examples.

Notes, remarks: JMetalCpp and JMetalPy versions. JMetalCpp provide less algorithms than JMetal, almost the same MOPs and problems. JMetalPy is just being developed in these days. For instant, it does not provide many features.
JMetal was practicaly included in MOEAFramework.
The algorithms provided by JMetal but not by MOEAFrameworks are dMPOSO, MOEA/D-DE, pMOEA/D-DE, MOEA/D-DRA, ssNSGA-II, pNSGA-II (we constat that mostly, just a different variant of the same algorithm is implemented).

2.1.3 PISA

Homepage: `http://www.tik.ee.ethz.ch/sop/pisa/?page=pisa.php`
Portability: Fragmented code written mostly in C, Java, Matlab
Algorithms provided: SPAM, SHV, SIBEA, HyPE, SEMO, SEMO2, FEMO,SPEA2, NSGA2 ECEA, IBEA, MSOPS, EPSMOEA
MOPs: GWLAB, LOTZ, LOTZ2, Knapsack Problem, EXPO, DTLZ, BBV, MLOTZ, MMPN, SCD
Performance indicators: some indicators such as epsilon, R and hypervolume provided
Comparison tools: PISA PA (stands for Performance Assessment)provides the comparison of different algorithms on benchmark problems using appropriate indicators and statistical methods.
Development tools: Not found.
Documentation: quite brief and fragmented, but comprehensible. Almost without examples/tutorials
Notes, remarks: Modules are completely independent of the programming language used or the underlying operating system.
PISA library is mostly included in MOEA Framework

2.1.4 PARADISEO

Homepage: `http://paradiseo.gforge.inria.fr/`
Portability: C++
Algorithms provided: MOGA, NSGA, NSGA-II, SPEA2, IBEA, SEEA, and various non evolutionary algorithms
MOPs: ZDT and DTLZ families, some real-world problems (routing, scheduling, bioinformatics) and also combinatorial problems
Performance indicators: hypervolume, additive and multiplicative epsilon, contribution, entropy
Comparison tools: Statistical tools and performance indicators provided.
Development tools: A rich set of classes to facilitate the development. Very easy to create own problems (there is nothing to implement but the problem-specific evaluation function).
Documentation: Good documentation, clearly separated into sections. Examples and tutorials.
Notes, remarks: The framework perpetually evolves.

2.1.5 KEA (Kit for Evolutionary Algorithms)

Homepage: Not found. Should be available on `http://ls11-www.cs.unidortmund.de/people/schmitt/kea.jsp`, but this link seems to not work. Documentation can be found on
`https://www.researchgate.net/publication/216300191_KEA---A_Software_Package_for_Development_Analysis_and_Application_of_Multiple_Objective_Evolutionary_Algorithms`
Portability: written in Java, support for C/C++ functions
Algorithms provided: SPEA2, NSGA-II, MOPSO/DOPS (and also Simplex)
MOPs: Some classical problems as Binh, Kursawe, Murata or Shaffer, some problems from ZDT/DTLZ families.
Performance indicators: hypervolume, R-1 metric, R-2 metric and R-3 metric
Comparison tools: GUI for visualization, evaluation method to eliminate useless data (for analysis). Module for comparisons contains performance indicators
Development tools: KEA is object-oriented and extensible.
Documentation: Quite good, with technical details as well as theoretical background.
Notes, remarks:

2.1.6 HeuristicLab

Homepage: `http://dev.heuristiclab.com/trac.fcgi/wiki`
Portability: C#
Algorithms provided: NSGA-II and many population-based algorithms, such as Particle Swarm Optimization, Offspring Selection Evolution Strategy, CMA-ES and the others.
MOPs: Not many problems we could use.
Performance indicators: Not found
Comparison tools: Experiment designer - a GUI where different algorithms with different parameter settings and problems can be composed, executed and analyzed. Number of tools for graphically analyzing the results.
Development tools: Algorithm designer - a GUI where we can model algorithms in a graphical way without having to write a code.
Documentation: In form of video tutorials, quite clear, could be better organized.
Notes, remarks: Really many algorithms and problems, but the most of them does not fit well to the project ambit.

2.1.7 Guimoo

Homepage: `http://guimoo.gforge.inria.fr/`
Portability: C++
Notes, remarks: It is not really a framework, but a software providing visu-

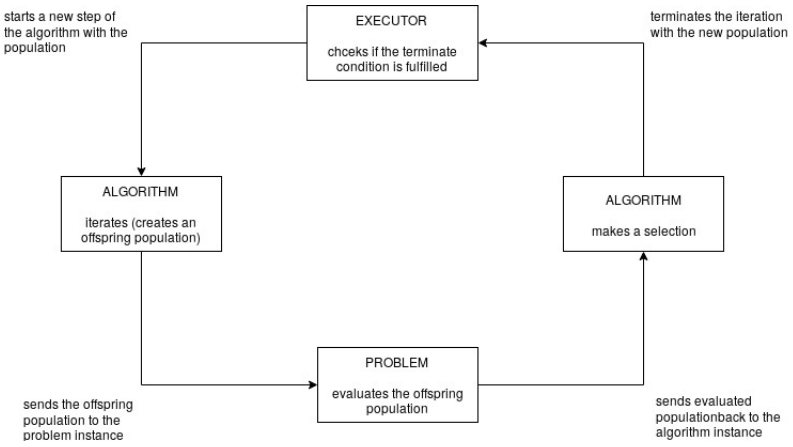
alization of Pareto front. It provides many metrics S-metric, R-metrics, contribution, entropy, generational distance, spacing, size of the dominated space, coverage of two sets and coverage difference. There are no MOEAs provided, some problems are supplied just for demonstration.

2.2 Communication between the MOEAFramework and an external problem

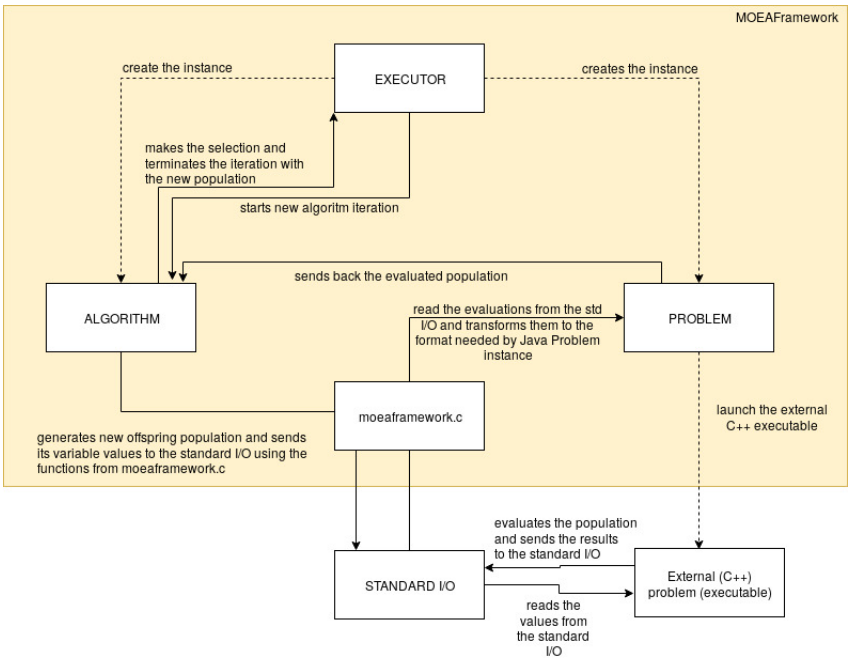
For our purpose, MOEAFramework was finally chosen for its various advantages such as a great number of provided algorithms, easily modifiable code or user-friendly documentation. Its architecture allows to test the chosen algorithm on the chosen problem. Normally, this is possible mostly thanks to three principal classes (or their inherited classes) of the framework. These are `Problem`, `Algorithm` and `Executor`. For each MOEA, the inherited class of the `Algorithm` abstract class is implemented. This class is responsible for performing the iterations, that means for generation offspring generations, making selections and so on.

The same, every concrete problem is represented by an inherited class of the `Problem` class. In this class the number of variables, constraints and objective functions of problem are specified, as well as the solution characteristics and objective functions that can calculate the fitness value of given solution.

The `Executor` class provide the communication between the problem and algorithm. Once the algorithm generates new population, it is sent to the problem to be evaluated, and this evaluation is then sent back to algorithm which makes the selection. For clearer comprehension, the graphical version of this process is given:



If we want to use an external problem, the `Problem` class launch an external executable which then communicate with the algorithm class basically via the standard I/O. The mechanism of this functionality is shown at the schema below:



However, the communication via standard I/O is not very robust. To ensure a good functionality, nothing but solutions and their evaluations can appear on the standard I/O. That is why the standard output stream `cout` was originally redirected to an external file. By default, the file is called `optimizer_moea_output.txt` and is created in the folder where the `MOEA_for_TPS` project is placed.

Lately, there was a need to run more tests at the same time. This was not possible with shared standard input and output. For this reason, this communication was replaced by a socket communication which solves the problem, is more robust in general and in addition, it seems to be faster than a standard I/O communication.

2.3 Adaptation of MOEAFramework for our purposes

Two big packages of classes were implemented - the package with testing tools, and the package for analyzing results. In this section, the key elements of implementation will be mentioned. The goal is not to describe the implementation in details, but to summarize what were our needs and how they were satisfied. The software documentation with more details and user manual is planned to be written.

2.3.1 The `InjectedAlgorithms` class

By default, the initial population is generated randomly by using the `StandardAlgorithms` class. However, the search space is so enormous that in 1000 of iterations the algorithms do not even approach good solutions with a completely random initial population - to be more concrete, while the DOSE cost function

value of the known solution is 6125, algorithms were able to find values of 1400000. This experience motivated the idea that we will generate initial population in a neighborhood of a known (called reference) solution. For this purpose, the mentioned class has been modified and saved under the name `InjectedAlgorithms`. This name comes from the fact that the framework provide a function `InjectedSolutions` which allow to initialize the algorithm with the list of known solutions.

The `InjectedAlgorithms` class provides all standard algorithms of the framework. To create the new instance of algorithm, firstly, the reference solution is loaded from the text file. Then, the initial population is generated randomly in a specified neighborhood of the reference solution, the algorithm is finally initialized with this population.

As not every algorithm provides the initialization from `StandardAlgorithms`, resp. `InjectedAlgorithms` class, some another classes needed to be adapted for this purpose. They can be recognized by the prefix "Injected" in their names - such as `InjectedSMPSO` or `InjectedPAES`.

2.4 Testing package

This package is responsible for the whole testing process - from loading run parameters through executing a test to displaying and saving results. While the running process is provided by framework standard classes (`Executor`, `Analyzer` and `Instrumenter`) and do not never change, the initialization part as well as the result presentation can vary. Thus, the `Tester` interface were created. There are two implementation - the GUI version and the XML one. In the default GUI version, run parameters are loaded from the GUI window. They are then passed to standard MOEAFramework classes that execute the test and send its results back to the window which display Pareto front plots, generational distance plot, as well as the text part of the solution. The plots and texts can be than manually saved to the hard disk.

To satisfy an user who do not want (or can not) use the GUI, the version using XML file is also provided. Run parameters are read from the `OptimizerMOEASettings.xml` file and passed to standard MOEAFramework classes. Test results that are than sent back to this implementation of `Tester` are automatically saved to the hard disk as images and text files.

2.5 Analysis package

This package provides a more complex analysis, such as comparison of different seed results or comparison of different algorithm performance. The GUI allows the user to choice if he wants analyze the results of different seeds of one algorithm run on one data set, or the comportment of multiple algorithms on one data set. The variant of analysis of one algorithm on multiple data sets is previewed, but not provided for instance.

The GUI load results (generated by functions of previous package) from a file and display desired plots and charts in the window. All the plots are saved

automatically on the hard disk, they can also be modified in the window and saved manually. At the moment, the XML version is not provided, but it is planned to be implemented in the future.

3 Analysis

In this section, we will analyze the results using different possibles point of view and user preferences. In the beginning, the global overview of all results will be done. Then we will focus in more details on each algorithm, we will discuss its strong and weak points and try to propose possible improvements or domain of use. Finally, some concluding remarks on obtained results and proposition for further testing will be given.

3.1 Global overview

For each algorithm, 20 tests with 200 iterations were run on two different data sets. We were observing the Pareto front, relative generational distance development during the run, spacing and population size. To get more complex information, the distribution of solutions within one seed was also studied. Elapsed time was not measured, as all the test were not performed in the same conditions. However, some remarks for giving a general idea can be done.

Following plots compare the performance of algorithms regarding the value of DOSE and LET functions respectively on the first and second data sets:

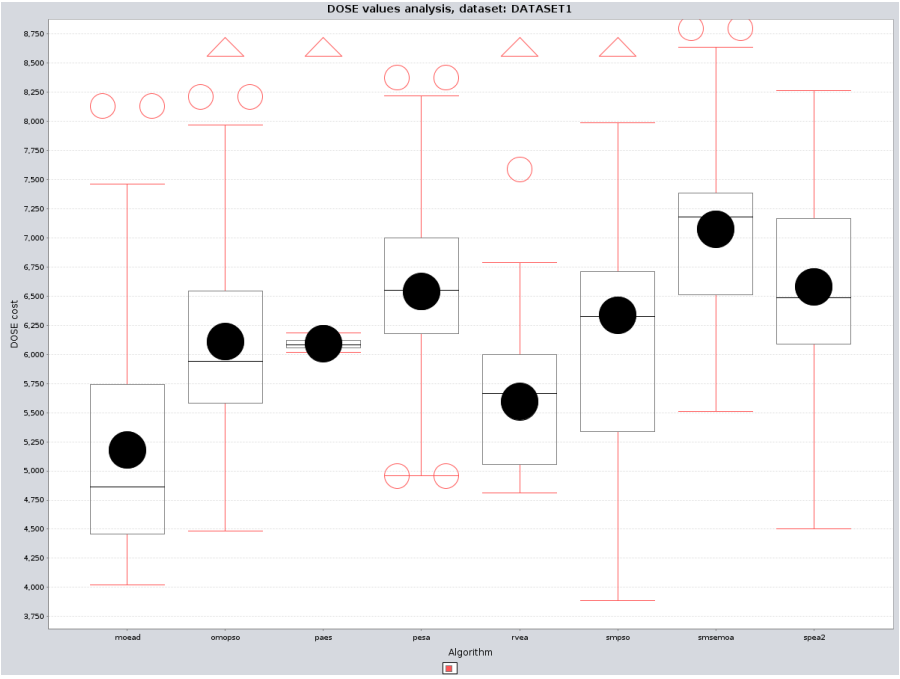


Figure 1: The first data set DOSE cost function characteristics

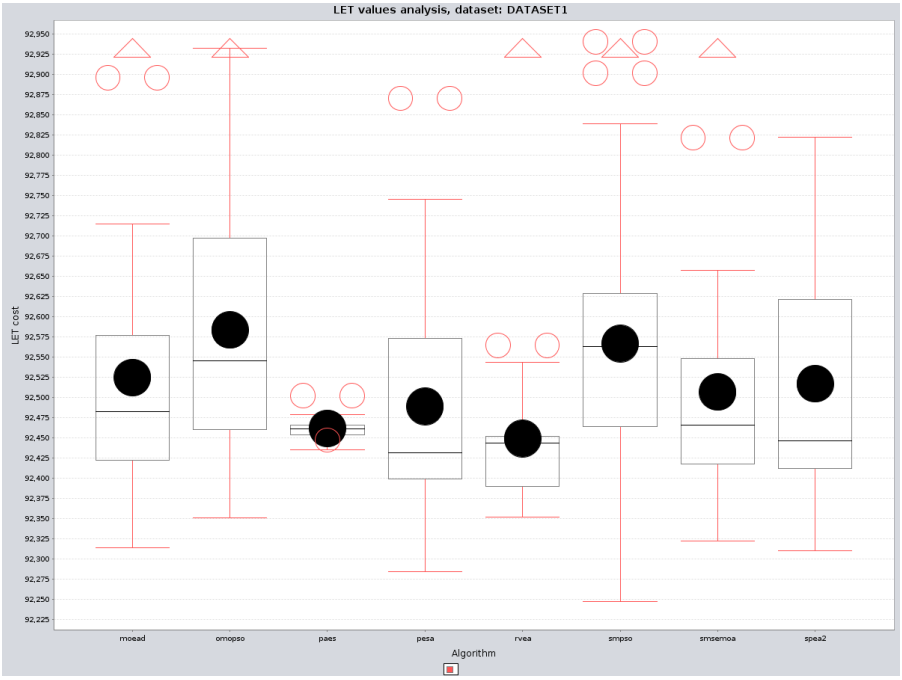


Figure 2: The first data set LET cost function characteristics

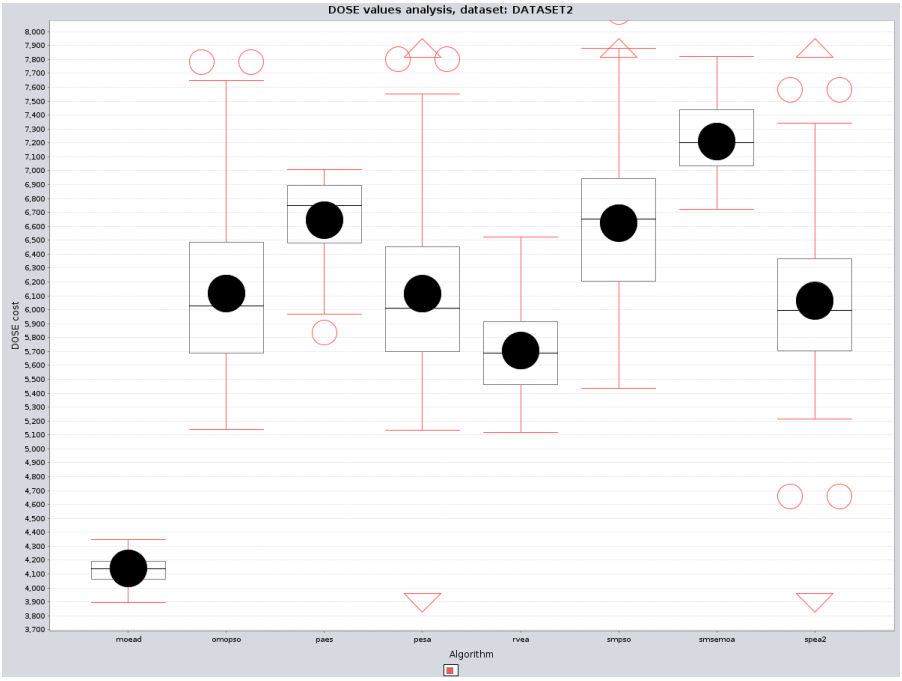


Figure 3: The second data set DOSE cost function characteristics

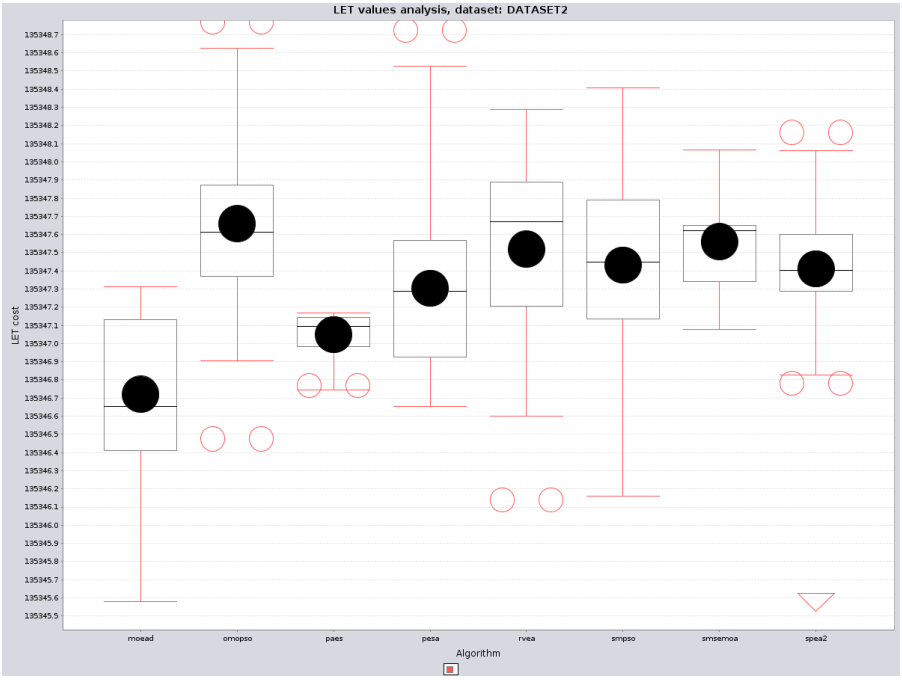


Figure 4: The second data set LET cost function characteristics

Regarding the LET cost function values, we note that they do not vary a lot in comparison with those of the DOSE cost function, which is especially true in the second case. The differences between algorithms are small in mean and average values. The algorithms only slightly differ in the scale of provided values. For this reason, the first quality evaluation and comparison of algorithms will be based on the DOSE cost function values.

Let we start by assuming that the algorithms do not perform exactly the same way on both data sets. On the first of them, the difference between the best and the worst algorithm is not as significant as on the second one. On the other hand, apart the extremes, the comportment of algorithms seems to be more variable on the first data set, while the relative differences between algorithms are less visible in the second case.

Although we observe some differences in comportment of algorithms on different test data sets, the basic information seems to be similar in the both cases. The algorithm which performs the worst is SMS-EMOA. On the other hand, we note that classical decomposition-based algorithms seems to perform better than the others. This is especially true for the MOEA/D algorithm which provides significantly better results, while the RVEA results stay comparable to the others algorithms. Classical Pareto dominance based algorithms (SPEA2 and PESA2) perform the same way. However, while on the first data set the algorithms are among the worsts, they could be classified as a "better average" on the second one.

Regarding the particle swarm optimization algorithms, they provide a very large spectre of solutions in both cases. In addition, OMOPSO is comparable to PESA2. Focusing on the median an average information, OMOPSO seems to be better than SMPSO which should not be the case as SMPSO was in fact developed as an improvement of OMOPSO. However, on the first data set, SMPSO reaches lower values than OMOPSO, and its first quantile values are even comparable to MOEA/D solutions. Contrary to classical Pareto dominance based algorithms, the particle swarm optimization algorithms could be classified as a better average on the first data set while on the second one they do not perform very well.

The last tested algorithm was evolution-strategy based PAES. We observe that this algorithm comport completely differently in both cases. While in the first case it provides a close scale of quite good solutions, and can be hence supposed as a really stable algorithm with a not-bad performance, the results obtained on the second data set are almost as poor as the SMS-EMOA ones.

However, the information is not complex. For example, we do not take into account the spacing and population size. Potentially, some algorithm with a big spacing and population can provide really large scale of solutions among that we can find several really very good solutions, but also a great number of poor solutions due to them it seems in global that the algorithm do not perform very well. In reality, this algorithm can be preferable to another one providing small population of close-scaled solution that are good, but not as good as in the case of previously mentioned algorithm.

Another problem that can appear is that the algorithm is not necessarily stable. We could prefer an algorithm that provides good solutions in every seed to an algorithm that provides usually a small population of poor solutions and rarely a huge population of very good solutions.

Last but not least, we would probably prefer an algorithm that is indifferent to a given data set. An algorithm that perform good on any data set is more reliable than another one which provides sometimes very well and sometimes very poor solutions.

So, let we now see various characteristics of different algorithms to can discuss its performance more in details.

3.2 Different algorithm analysis

To start, let we stay in case of comparison of different algorithms for a moment. We give, fort both data sets, the comparison plots of spacing and population size. Then we focus more punctually on these characteristics separately for each algorithm.

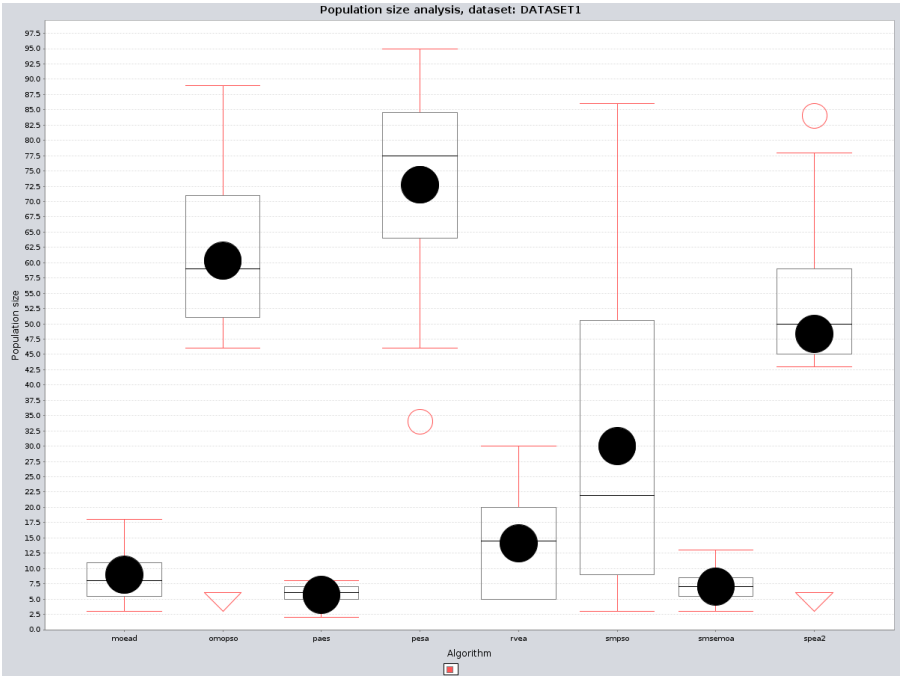


Figure 5: The first data set population size characteristics

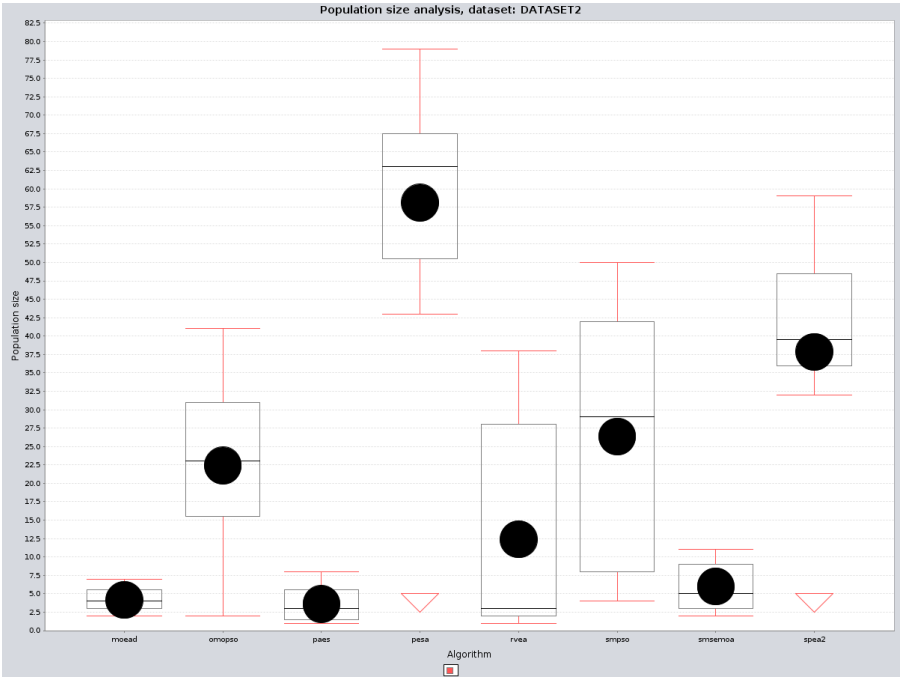


Figure 6: The second data set population size characteristics

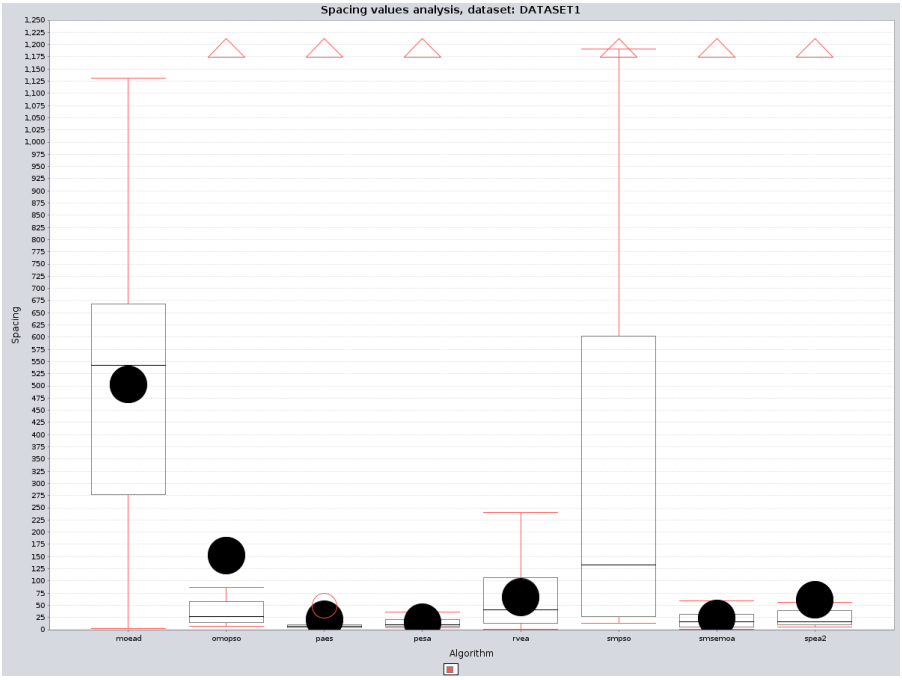


Figure 7: The first data set spacing characteristics

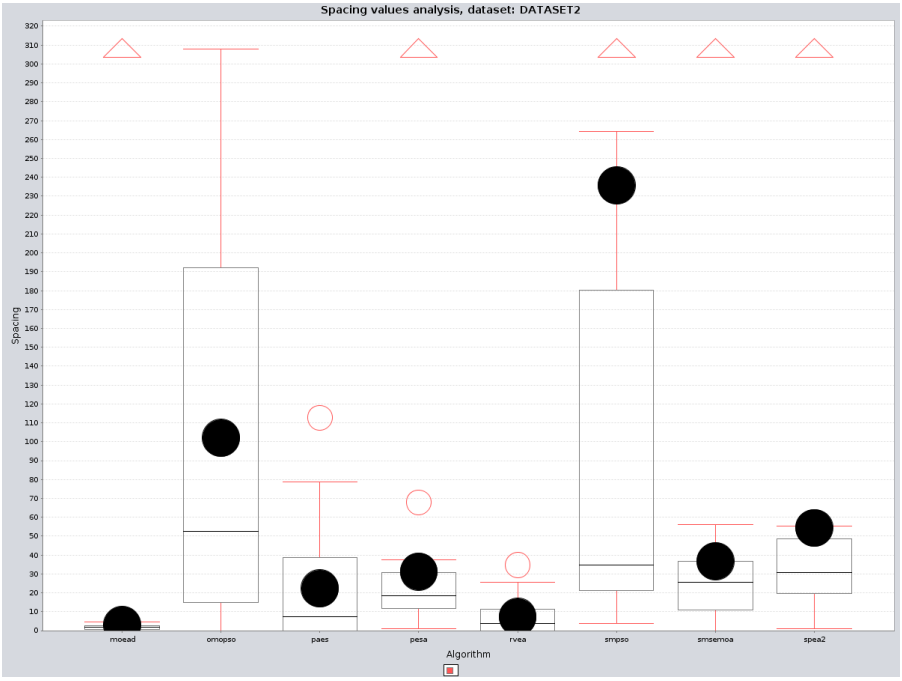


Figure 8: The second data set spacing characteristics

MOEA/D

As said previously, this algorithm seems to perform the best on our problem. We see that the characteristics change with data sets. In first case, the largest population had 17 individuals. It was only 7 individuals in the second case. Relatively to the others algorithms, the population size is small. It is even the smallest one in the second case. If we look at spacing, we observe that although the small population, it has the biggest spacing among all the algorithms in the first case. Analyzing the DOSE and LET values of different seeds and plotting their Pareto fronts, we note that there is always an isolated solution whose LET and especially DOSE values are relatively very far from the others solutions. Of course, as the nearest neighbor of this solution is far, it has an significant impact on the final spacing values. Despite of this fact, the others solutions have a really good quality.

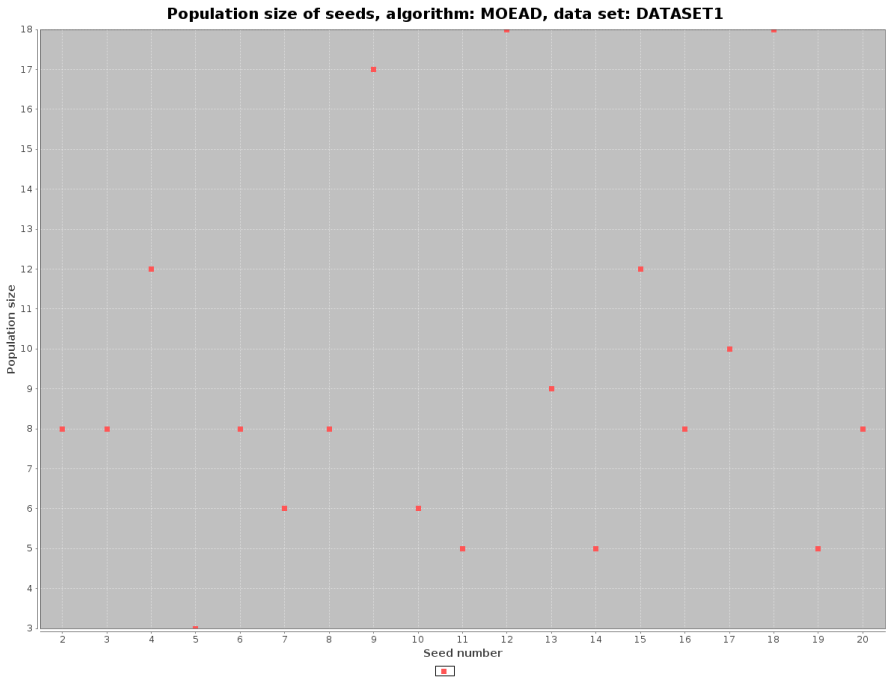


Figure 9: MOEA/D: the first data set population size by seed.

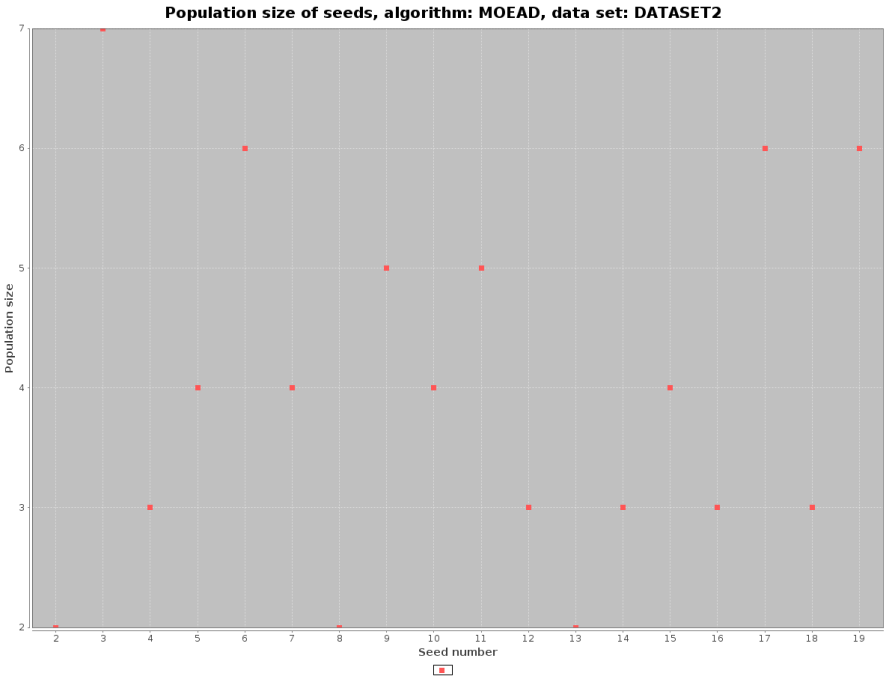


Figure 10: MOEA/D: the second data set population size by seed.

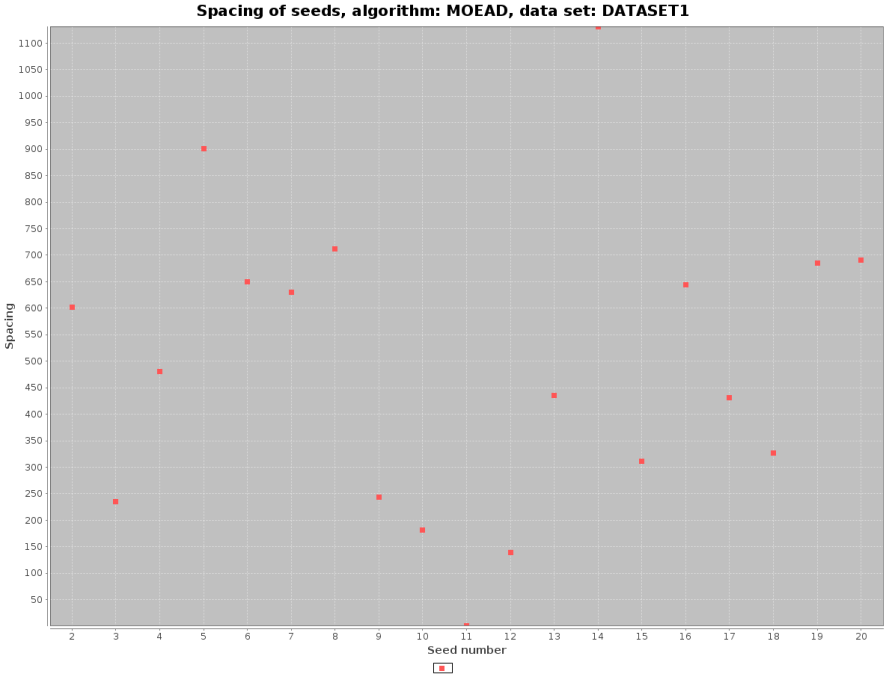


Figure 11: MOEA/D: the first data set spacing value by seed.

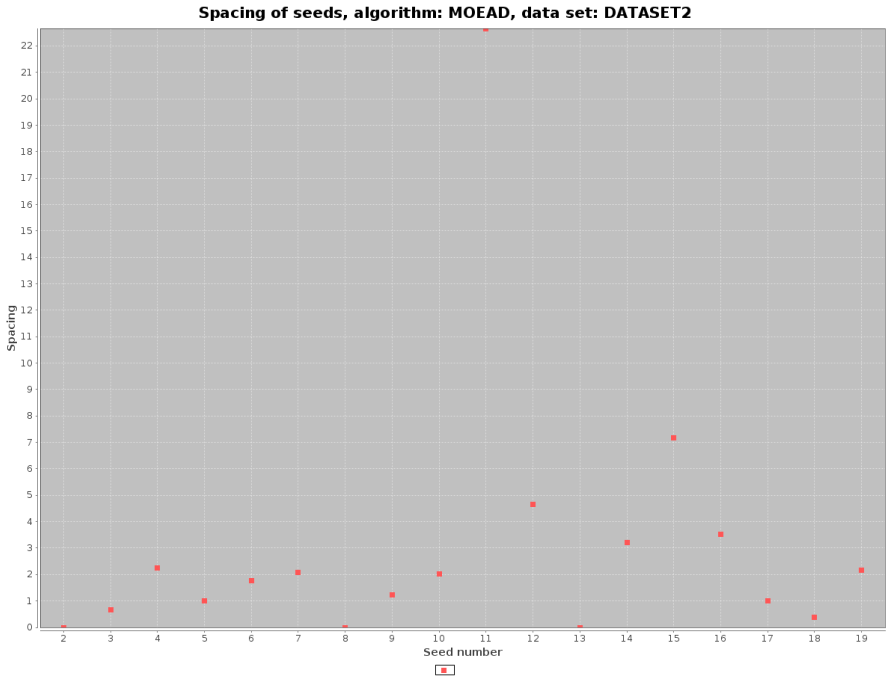


Figure 12: MOEA/D: the second data set spacing value by seed.

Regarding the first data set, in 13 tests of 20, at least one solution has DOSE value under 5000, in 7 tests, at least one solution has DOSE value under 4500 and the same number of tests gives some solution with a DOSE value between 4000 and 4500. From the 7 tests which do not give any solution with DOSE value under 5000, only one can be classified as poor with all its solutions above 6000. We also note that the solutions from the first three quantiles are quite close one to the others. The larger scale can be observed in the fourth quantile, which can be explained by the presence of isolated solutions. From this information, we can conclude that the most of provided solutions are of the really good quality.

From the plot comparing the LET characteristics of different seeds, we note that within one seed, the LET is practically constant. That means, obtaining the results of one seed, the decision about what solution will be used can be really made on the basis of DOSE cost function value, any additional trade-offs between DOSE and LET are not necessary. In global overview, the most of LET values of all seeds is contained between 92400 and 92600. We also remark the presence of an isolated solution with a very low value of LET. However, we see from the Pareto front plots that its DOSE value is poor.

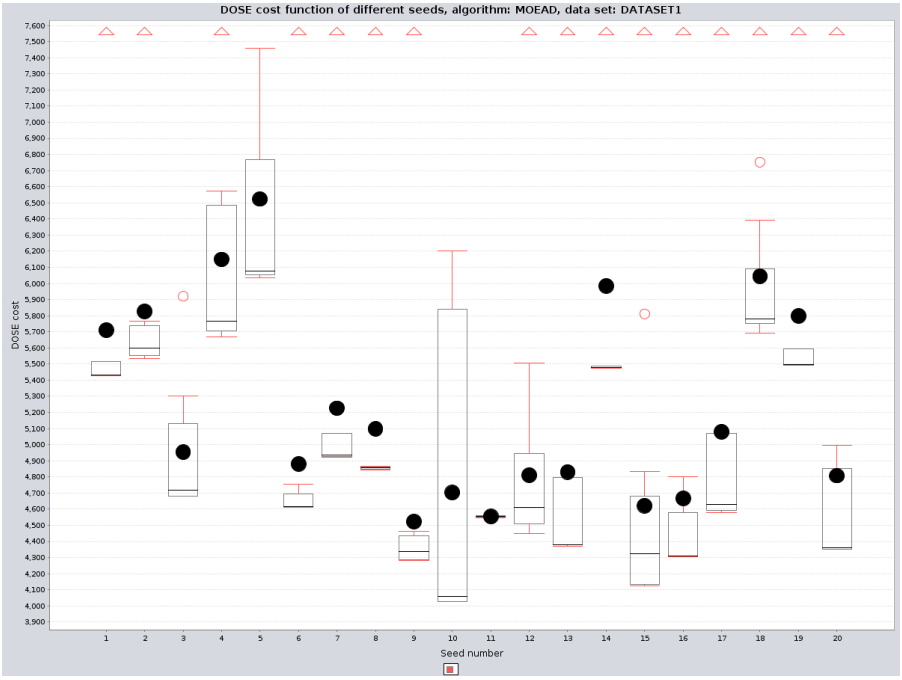


Figure 13: MOEA/D: the first data set DOSE cost characteristics by seed.

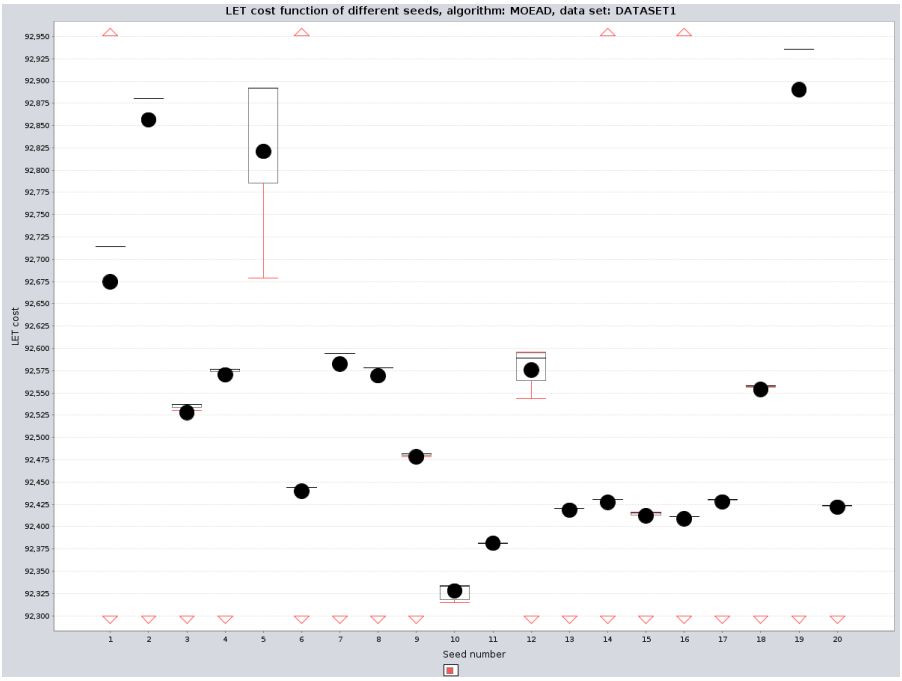


Figure 14: MOEA/D: the first data set LET cost characteristics by seed.

Regarding the second data set, the solutions are even better. Already the pre-

viously displayed plot that compares different algorithms DOSE characteristics shows that every solution provided by this algorithm has its DOSE value under 4400. In fact, there are only two different algorithms, PESA and SPEA2, providing a solution that reaches the value in this range. In addition, we see that the scale of DOSE values is very closed, which assure the reliability of this algorithm. The analysis of different seeds corresponds to the small spacing. Each seed provides a close set of solutions, there is only one seed containing an isolated solution. This observation can be partially caused by a small size of populations.

Also in this case, LET values within one seed are practically identical. We can also remark that the LET value scale is really negligible, going from 135345 to 135347, which correspond with the fact that the LET cost function almost do not change on the second data set.

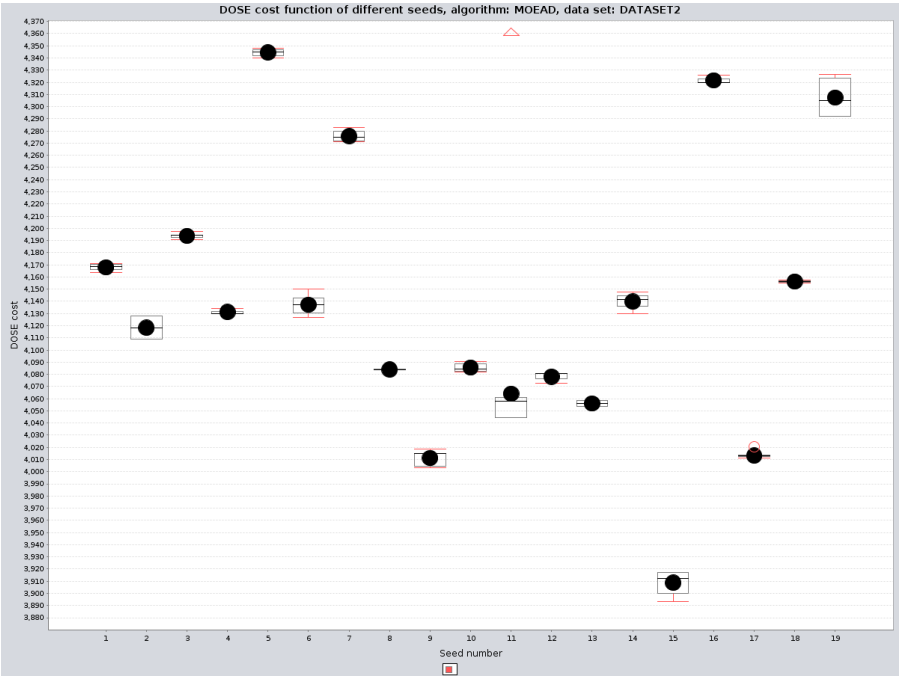


Figure 15: MOEA/D: the second data set DOSE cost characteristics by seed.

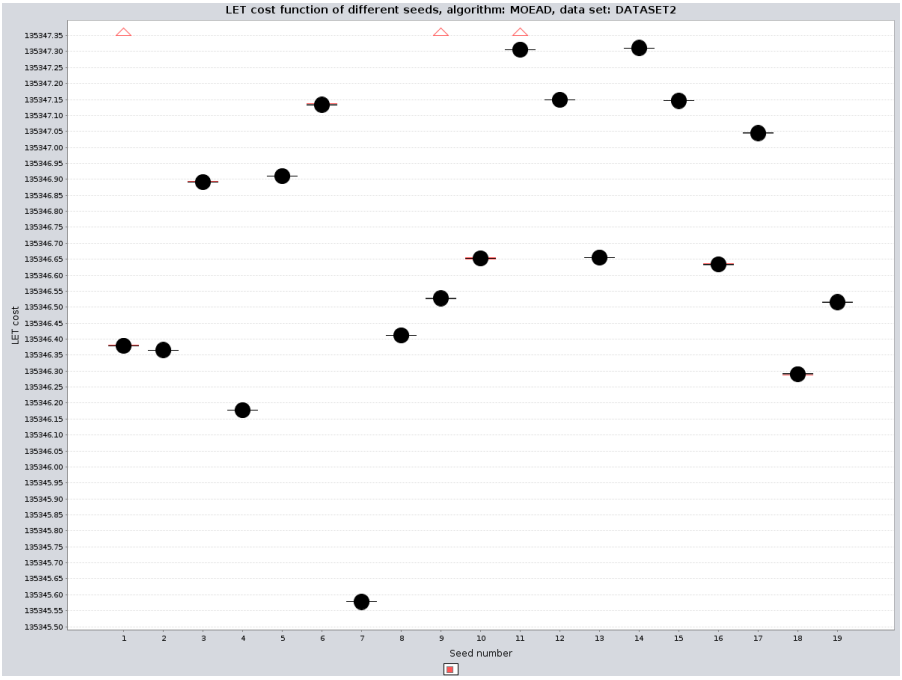


Figure 16: MOEA/D: the second data set LET cost characteristics by seed.

In conclusion, the strong point of this algorithm is indisputably the quality of the DOSE cost function values. In the second case, we should mention its high-level of reliability. In the first case, the algorithm is less reliable than in the second one, but still enough to promote the existence of very good solutions with a high probability. Its weak point can be the small population size. Also, this algorithm seems to belong among the slowest ones.

Finally, according to the observations based on relative generational distance, the algorithm does not seem to converge on the threshold of the 200th iteration. In fact, any tendency of convergence or slow down of progress is observed. As the relative generational distance only indicates the movement of generations (whether it is to better or not), we can not predict if the population would keep improving if we increase the number of iterations.

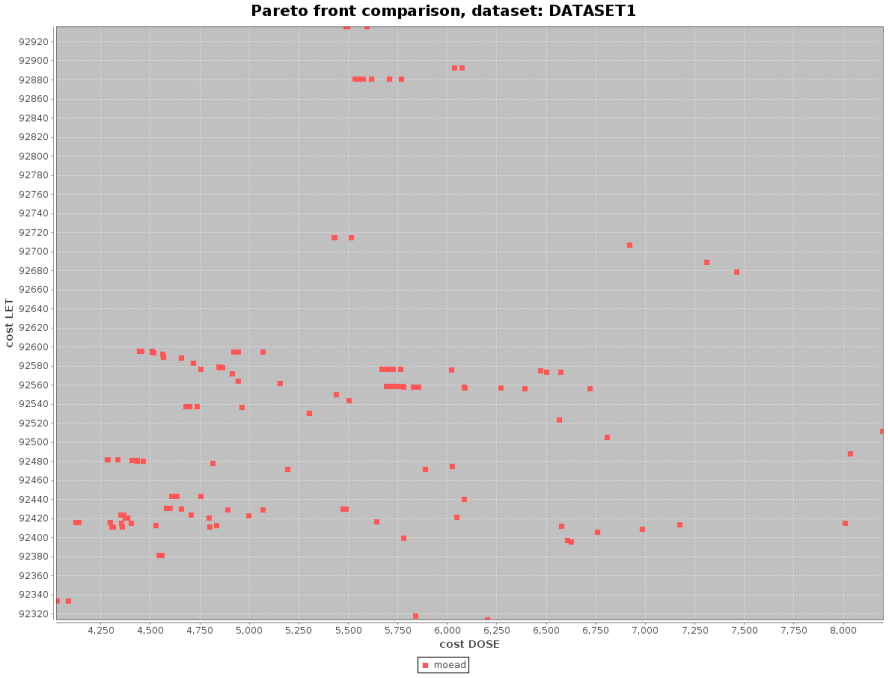


Figure 17: MOEA/D: the first data set solutions of all seeds plot together.

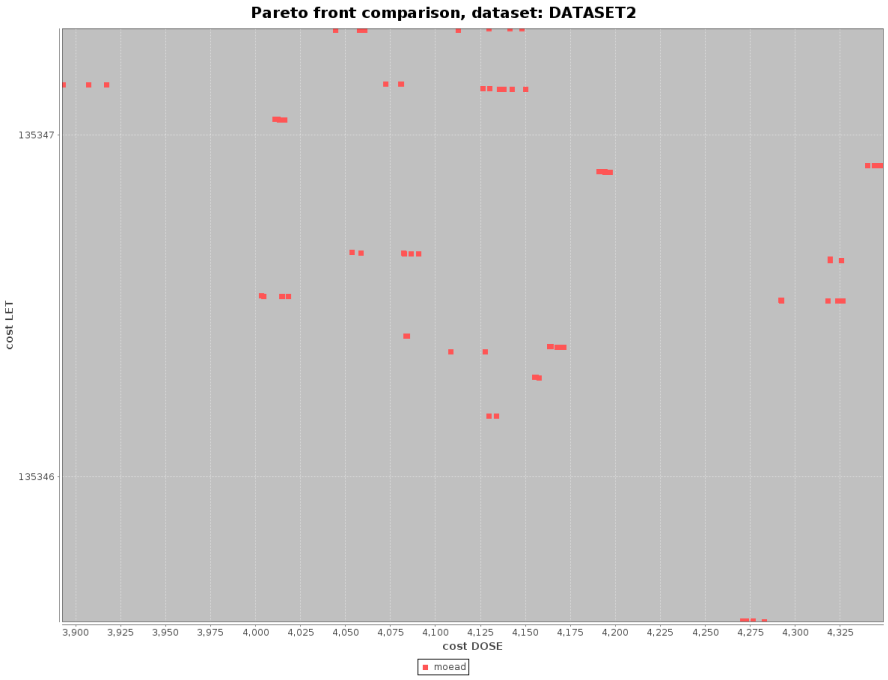


Figure 18: MOEA/D: the second data set solutions of all seeds plot together.

RVEA

This algorithm also demonstrated a good performance relatively to its opponents, even though the gap between RVEA and MOEA/D stays considerable, and in favour of MOEA/D. In many cases, the population size does not exceed 10 individuals. This is in particular true on the second data set where 13 of 20 populations contain only from 3 to 5 individuals. On the first data set, the situation is better - they are 9 populations with less than 10 solutions. The other populations provides in general about 20 solutions. So, despite the fact that there are few well-populated results on the second data set, in general, the algorithm provides bigger populations on the first data set, which is in agreement with the case of MOEA/D. We remark that spacing values cover a large spectre of values in the case of the first data set, while they are stably small in the second case. In principle, smaller spacing should mean more uniformly distributed solutions. However, regarding the population sizes of the second data set results, in this case it probably means that solutions are concentrated on a small area in the case of the second data set. On the other hand, we see on the Pareto front plots of the first data set that solutions are well dispersed in the second case, even though the distribution is not really uniform.

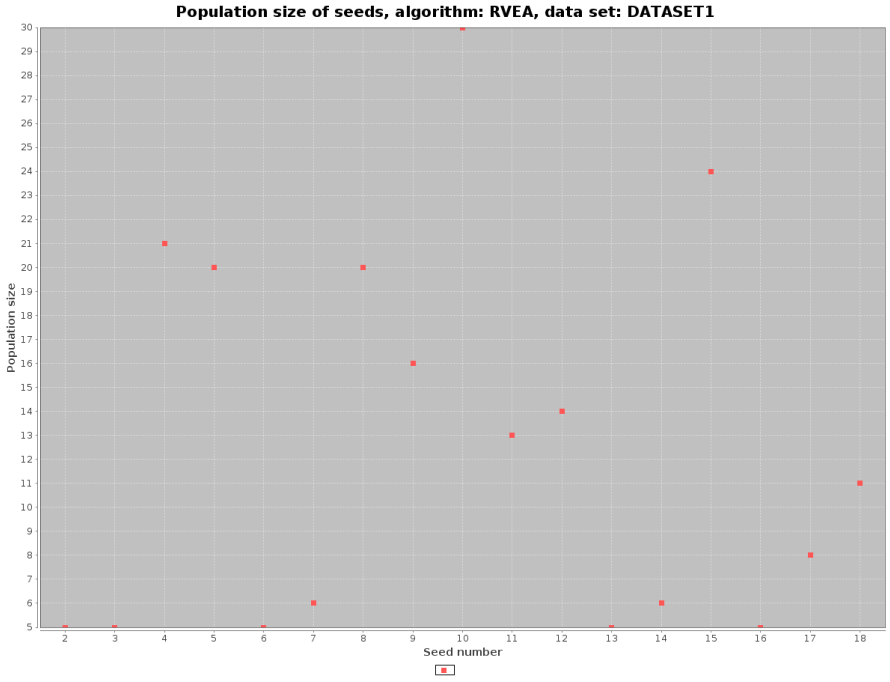


Figure 19: RVEA: the first data set population size by seed.

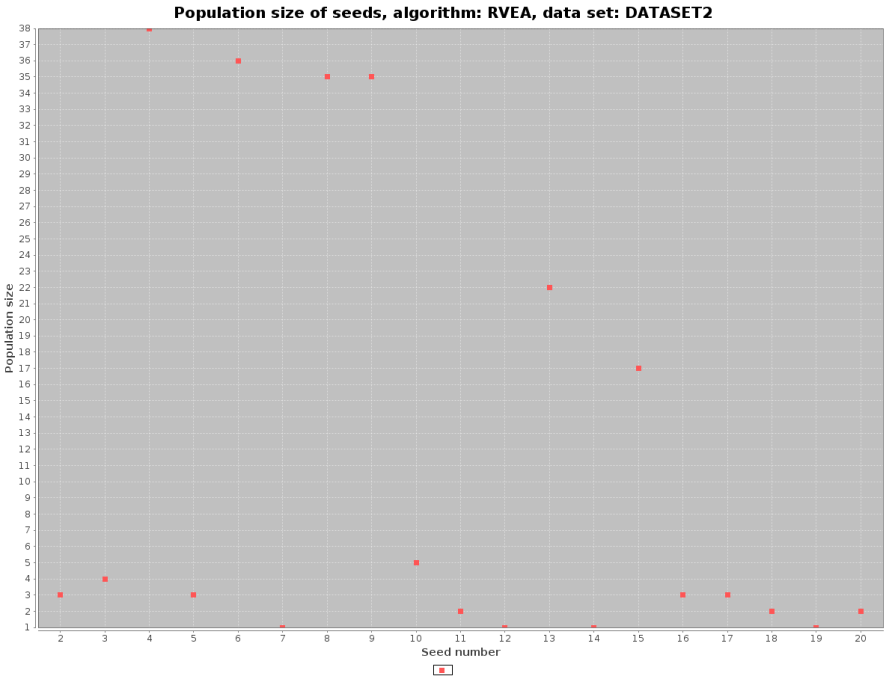


Figure 20: RVEA: the second data set population size by seed.

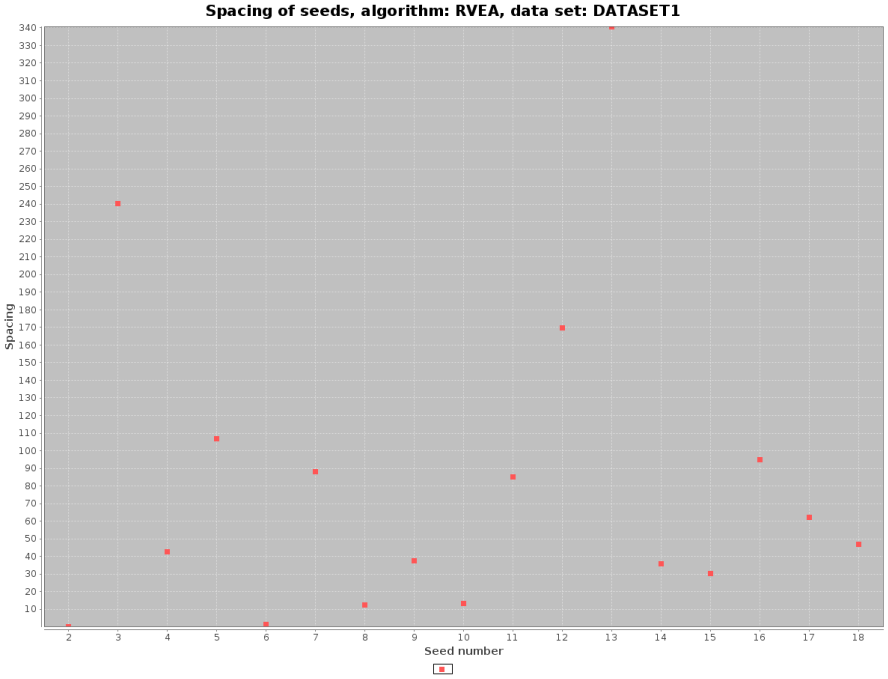


Figure 21: RVEA: the first data set spacing value by seed.

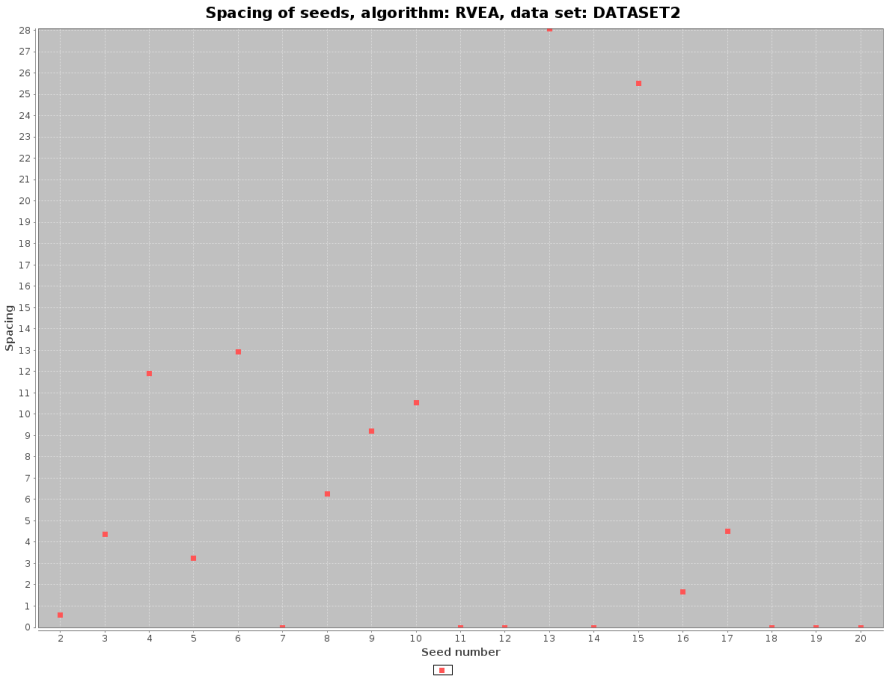


Figure 22: RVEA: the second data set spacing value by seed.

Analyzing the DOSE cost function values of different seeds of the first data set, we observe that there are 3 tests of 20 that do not provide any solution with a DOSE value under 6000, these test results can be classified as poor ones. There are only 5 seeds reaching the DOSE value smaller than 5500, only 2 of them go under 5000.

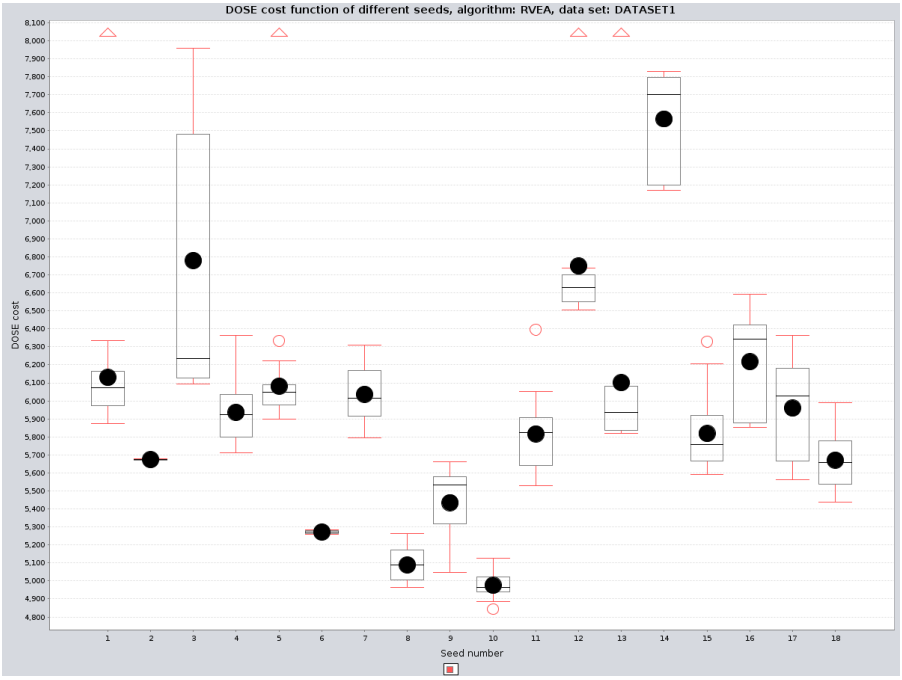


Figure 23: RVEA: the first data set DOSE cost characteristics by seed.

Regarding the second data set, only one test does not provide any solution with DOSE value under 6000, and there are any test reaching a value under 5100. We also observe that in the 13 cases of 20, the scale of dose values is very close, which strengthens the argument of small populations with the solutions concentrated on a small area.

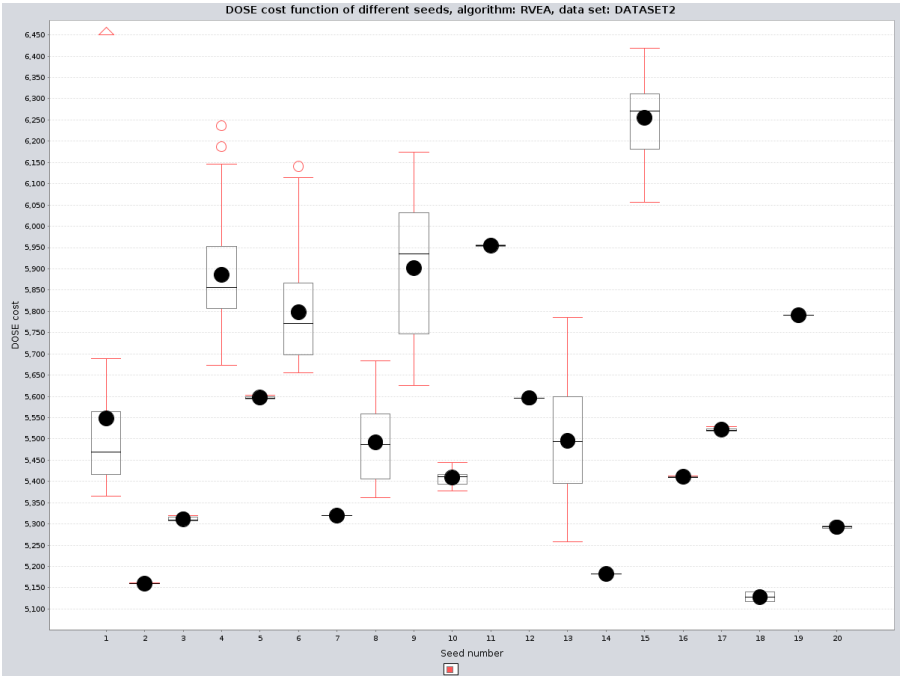


Figure 24: RVEA: the second data set DOSE cost characteristics by seed.

In conclusion, this algorithm can not outperform MOEA/D in DOSE cost function values, even neither in LET cost function values. However, the algorithm performance is quite good in both cases, so seems to be indifferent to data set. Although it provides a well-populated and well-dispersed results on the first data set, we can not say that it is the strong point of the algorithm, as on the second data set provided populations are usually small and closed. The strong point could be the algorithm time efficiency. In our tests, it performed approximately twice faster than all the others algorithms except SMS-EMOA and PAES. Regarding the relative generational distance, the algorithm do not converge on the threshold of 200th iteration. Any special progress type was observed - contrarily, the evolution seems to be really variable. There were some cases when the algorithm moved a lot during several first iterations and than it became stable (which we could consider as a convergence). In other cases, the algorithm was varying considerably all the time or during some larger interval of iterations. As in the case of MOEA/D, we can not predict if the population would keep improving with the increasing number of iterations.

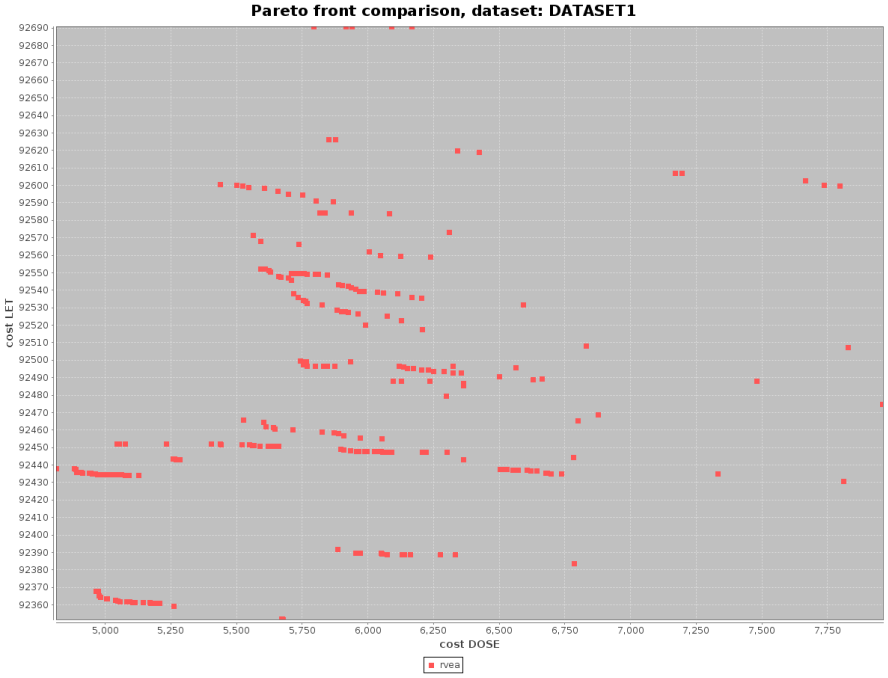


Figure 25: RVEA: the first data set solutions of all seeds plot together.

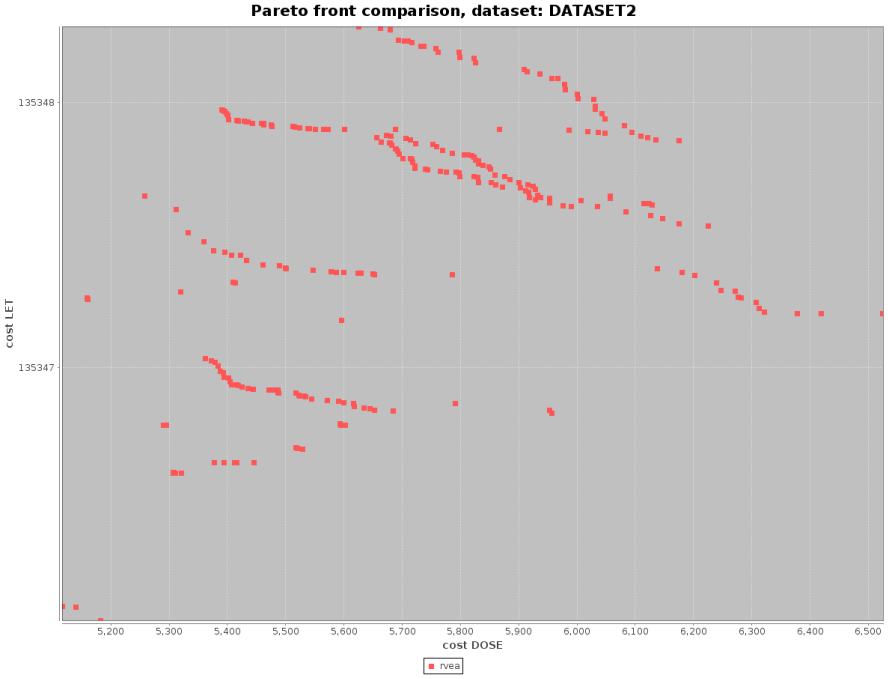


Figure 26: RVEA: the second data set solutions of all seeds plot together.

OMOPSO

As the previously mentioned algorithms, OMOPSO provides bigger populations on the first date set than on the second one. The population consists usually of about 50 individuals in the first case. In the second case, population contains the most often from 20 to 40 individuals. In the first case, the spacing is relatively small. If we regard the Pareto front plots, we note that there is practically always an isolated solution. This solution is usually the best one in LET and the worst one in DOSE. Contrary to MOEA/D, it does not influence considerably OMOPSO spacing values, as it provides a huge number of solutions which are well-distributes, except that isolated one. The spacing values are much worse (bigger) on the second data set. However, Pareto front plots show the same characteristics as in the first case - well-distributed solutions except the isolated one which is still present. As in this case populations are about twice as small as in the first case, the impact of the isolated solution on the final spacing value is more important.

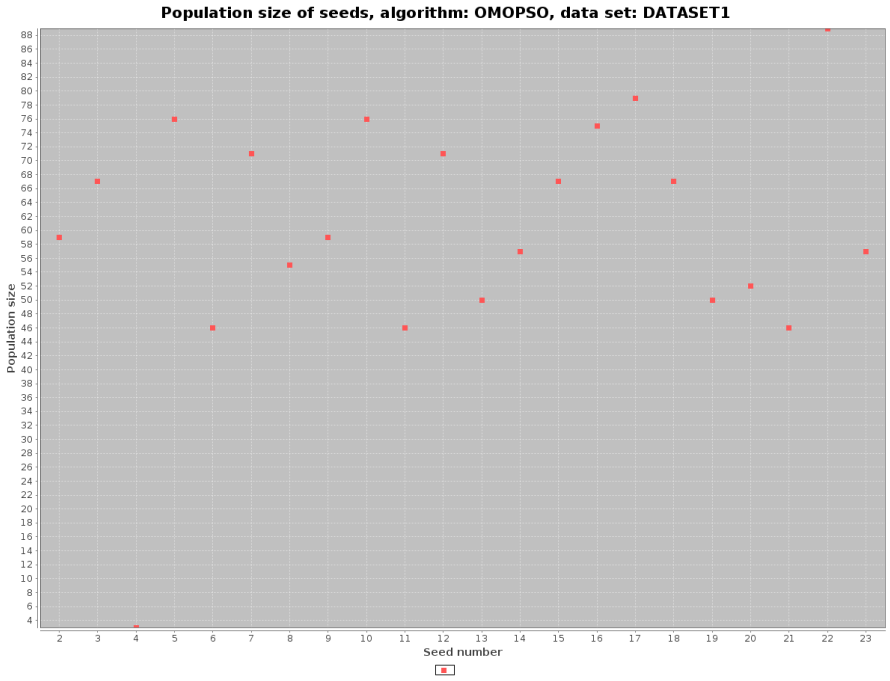


Figure 27: OMOPSO: the first data set population size by seed.

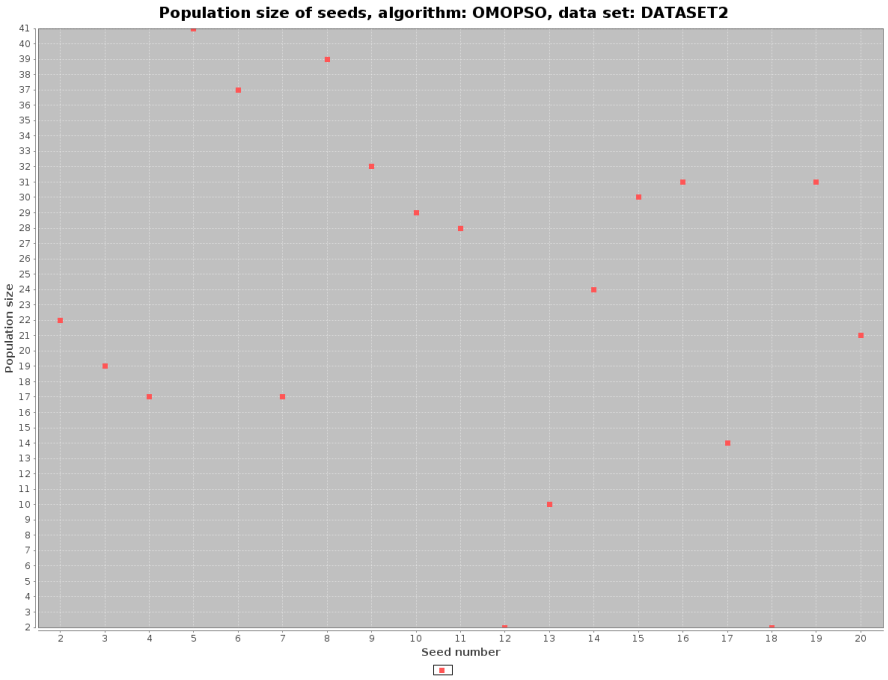


Figure 28: OMOPSO: the second data set population size by seed.

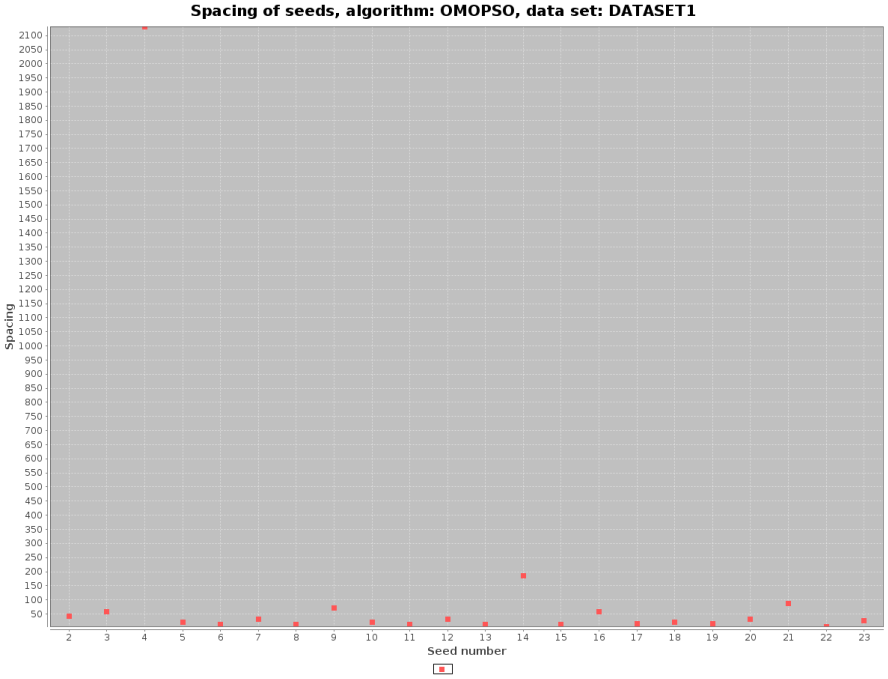


Figure 29: OMOPSO: the first data set spacing value by seed.

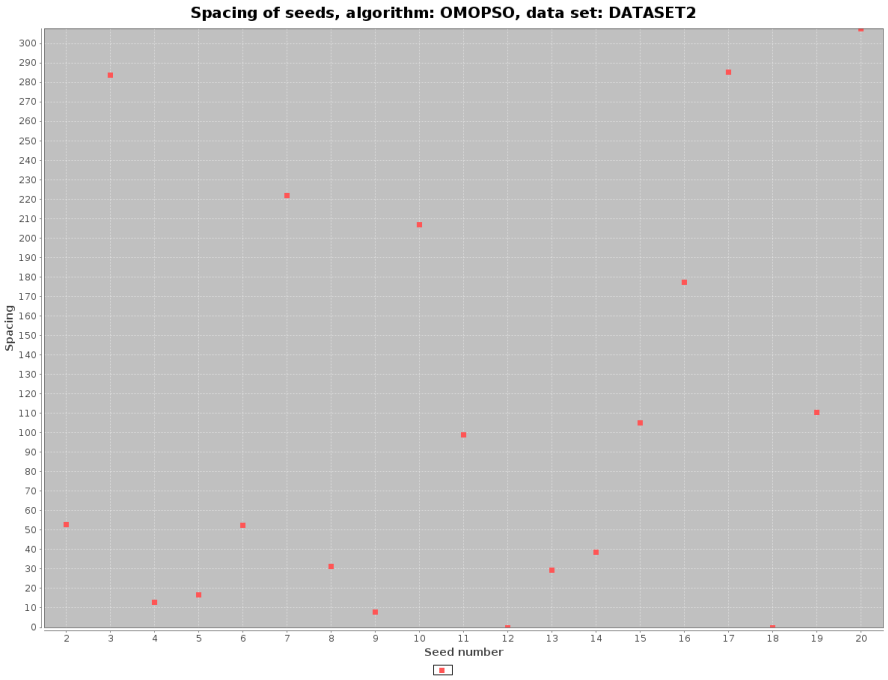


Figure 30: OMOPSO: the second data set spacing value by seed.

From the previously given plot that compare DOSE characteristics of different algorithms it could seem that OMOPSO does not perform specially well. However, the knowledge of population size and the fact that solutions are well-distributed offers the new point of view. Let we analyze the comportment of different seeds. At first, we observe that the fourth seed is completely different to the others. It provides only 3 solutions of poor quality. We do not know the reason this solution appeared. It could potentially indicate some weak point of OMOPSO algorithm, but we can not assume (or disprove) it without performing more tests and trying to re-simulate this comportment.

Apart this particular result, all the seeds expose similar characteristics. As in the case of MOEA/D, there is only one of them that does not give any solution with a DOSE value under 6000. 16 populations goes under 5500 and 6 under 5000. In 2 cases, a value under 4500 was reached. As the population is big with a very large scale of reached DOSE values, the average and mean information is worse than in the case of RVEA. However, detailed overview shows that, thanks to big populations, this algorithm outperforms RVEA at least in DOSE values.

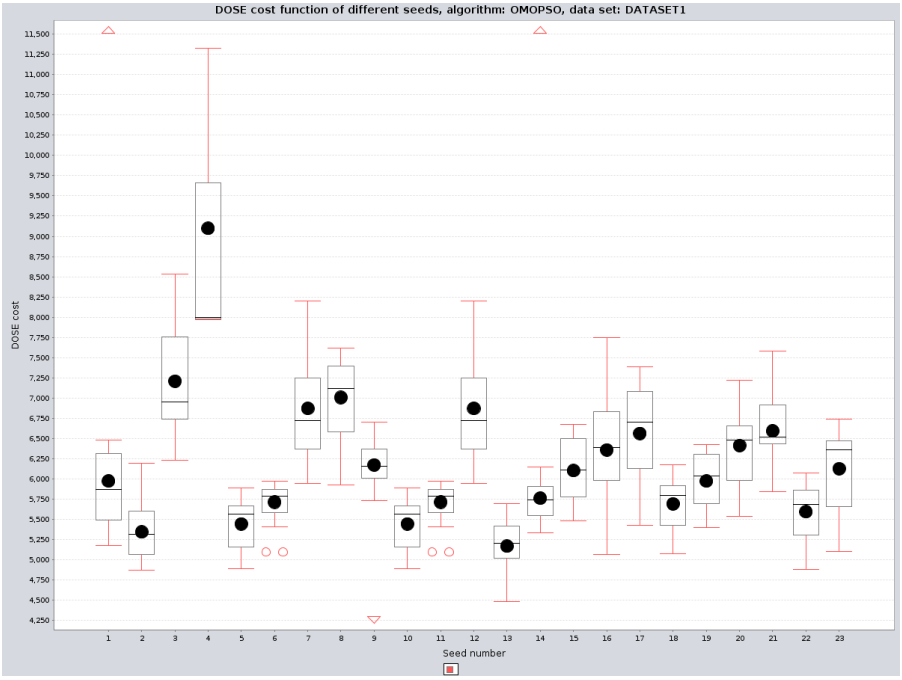


Figure 31: OMOPSO: the first data set DOSE cost characteristics by seed.

Regarding the second data set results, also here we observe two seeds with that particular comportment. These are seeds 12 and 18 with the populations of only 2 individuals. Contrarily to the previous case, the obtained DOSE values are not poor.

Every seed provides at least one solution with the DOSE value smaller than 6000. In 13 cases, the value smaller than 5500 was reached, in 3 cases, the algorithm approached (but did not reach) the value of 5000. In addition, the fact that the solutions are well-distributed promise the existence or more than only one very good solution in each seed. Also on this data set, RVEA seems to be outperformed by OMOPSO.

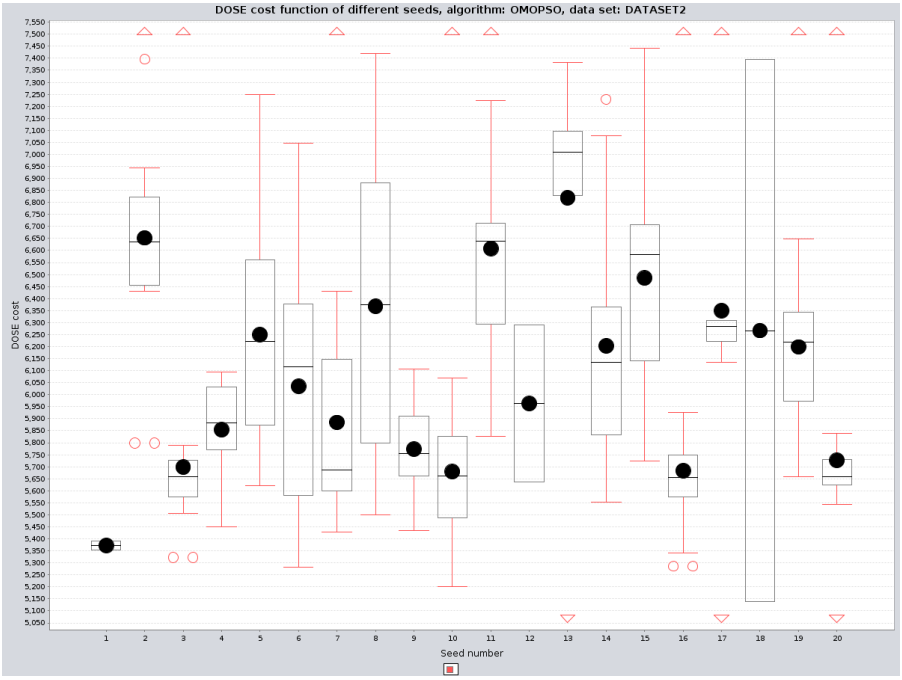


Figure 32: OMOPSO: the second data set DOSE cost characteristics by seed.

In conclusion, the strong point of OMOPSO is certainly the fact that it performs quite indifferently on both data sets. Provided population is stably big, except those particular seeds whose appearance we can not explain for instance. Solutions are well-distributed and cover a large scale of values. Another strong point is its reliability - some very good solutions can be found in almost every seed. To mention some weak point, the algorithm is among the worst ones regarding the LET. It does not seem to be a big problem in the case of the second data set where the LET practically does not vary, but it can be more important in the case of the first data set in applications where the LET optimization has a big priority. Also, we should mention that apart good solutions, we are generating many poor solutions that we will probably never use. That is quite redundant, especially if we take into account the fact that bigger the population is, slower the algorithm perform. On the other hand, regarding the relative generational distance, we observe that the population evolves a lot at the beginning for about 30 iterations. After this short period, the relative distance between successive generations is mostly zero, sometimes some negligible change appears between two iterations. So probably, we could stop the algorithm after less than 200 iterations without an impact on obtained results. That would considerably reduce the execution time.

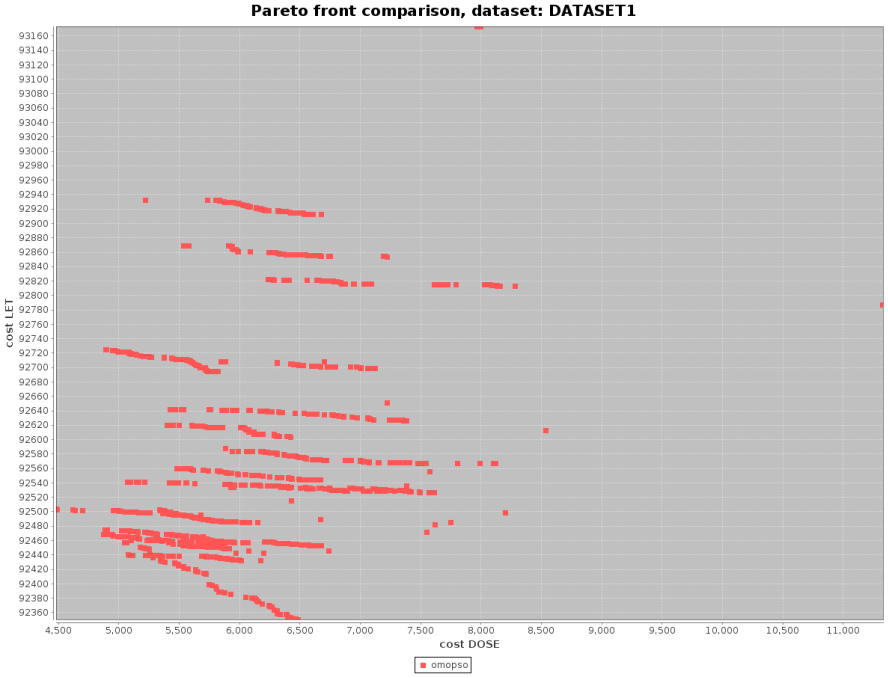


Figure 33: OMOPSO: the first data set solutions of all seeds plot together.

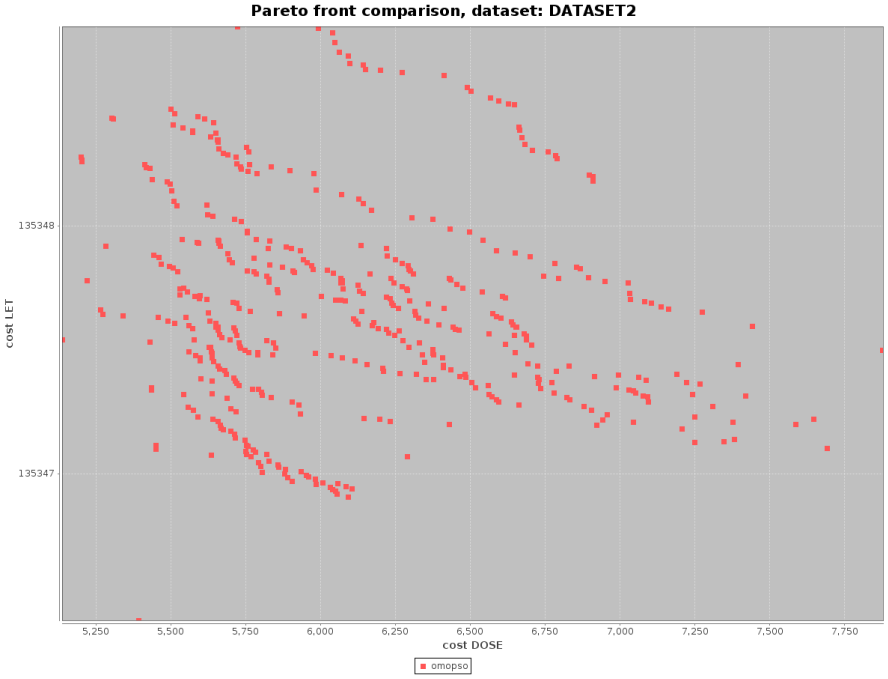


Figure 34: OMOPSO: the second data set solutions of all seeds plot together.

SMPSO

As SMPSO has been developed with motivation to improve some weak points of OMOPSO, one would expect that it would perform better than OMOPSO. That seems to be true on the first data set, but not on the second one. In both cases, population size vary from small to larger numbers. There are an important number of small populations with the size size about 10 individuals, there are big populations with 50 (or even more on the first data set) individuals and there are also lots of populations of the medium size. In general, we can say that it is an algorithm providing mostly medium-populated results with about 20 or 30 solutions which also often provides larger or smaller population. Regarding a spacing values, SMPSO is among the worst algorithms in both cases. However, solutions are in general well-distributed. In fact, as in the case of OMOPSO, we observe the appearance of an isolated solution which has the lowest value in LET and the highest value in DOSE. Population being in general slightly smaller than in the OMOPSO case, this isolated solution has an important impact on the final spacing values.

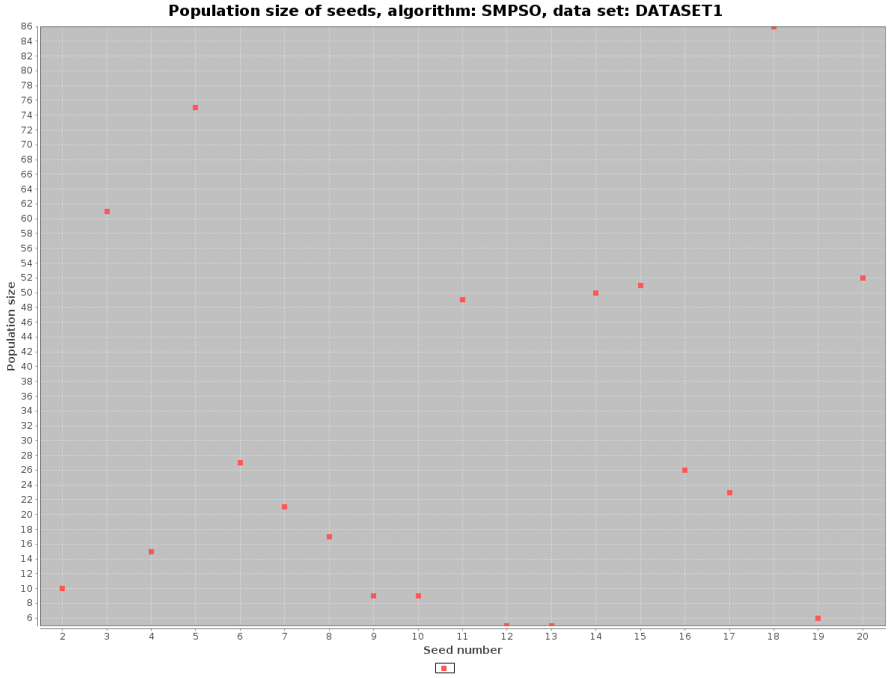


Figure 35: SMPSO: the first data set population size by seed.

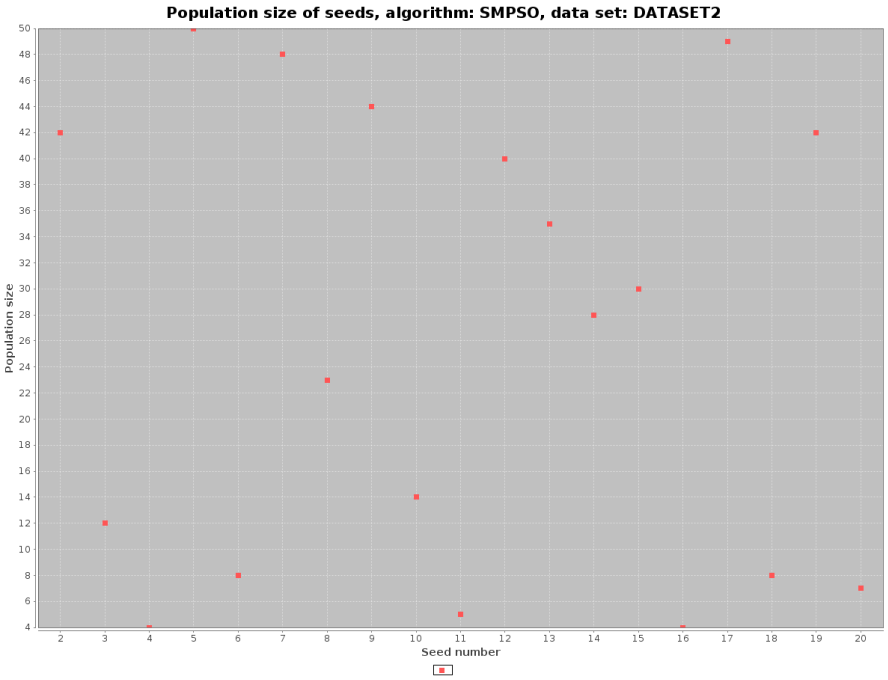


Figure 36: SMPSO: the second data set population size by seed.

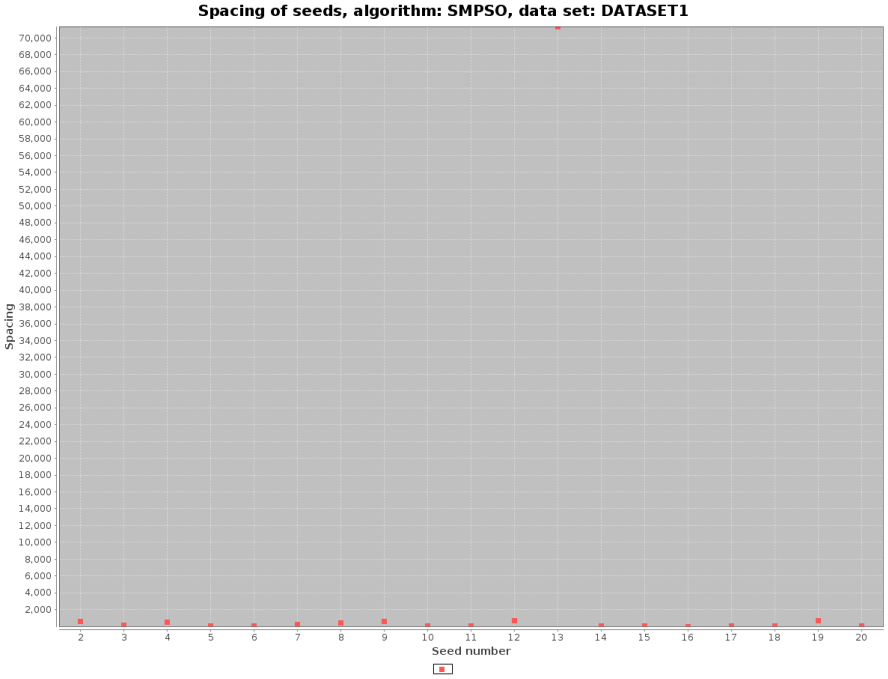


Figure 37: SMPSO: the first data set spacing value by seed.

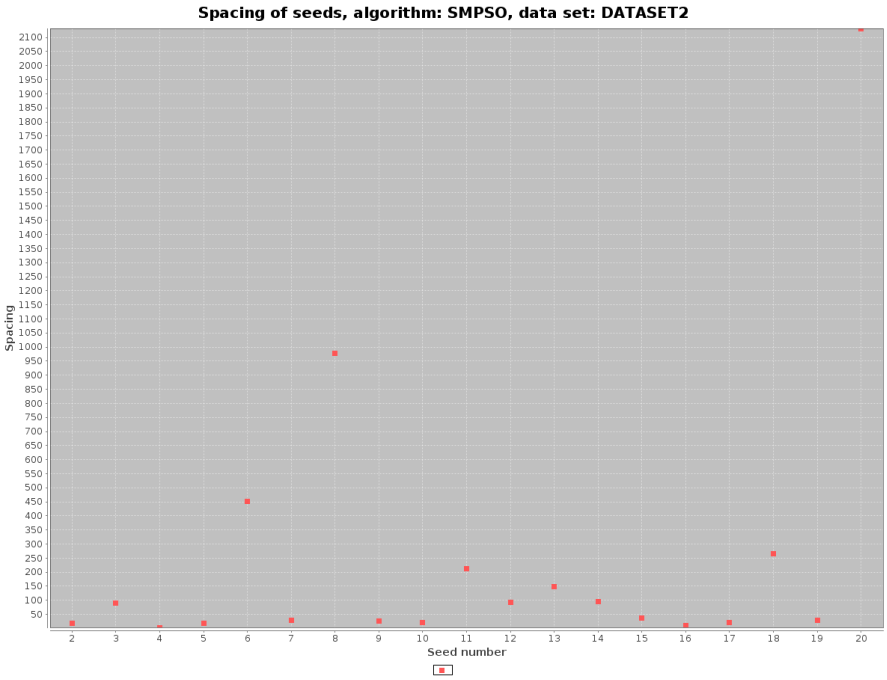


Figure 38: SMPSO: the second data set spacing value by seed.

Regarding the first data set results, we have to note that they are very variable. About a half of seeds offer a large scale of solutions from very good to very poor ones, contrary to the rest of seeds providing a very close scale of more or less good solutions in which the mentioned isolated (poor in DOSE) solution is present. We should also mentioned the seed number 13 which is very particular. In fact, it provides 4 very good solution with a DOSE value under 5000, but its isolated solution costs 164480 in DOSE. That is the only case we observe such a enormous isolation. In all the others seeds, the isolated solution costs about 8000 in DOSE. As in the case of OMOPSO, we can not conclude without more tests and re-simulations of this phenomenon if it points to some weak point of SMPSO or for which reason it appeared.

There are 2 seeds of 20 that does not provide any solution with a DOSE cost function value under 6000. 17 tests provides some solution with a DOSE value under 5500, 12 of them goes under 5000, 9 under 4500. Finally, 5 of them approaches the value of 4000. We note that in this point of view, SMPSO outperform even MOEA/D.

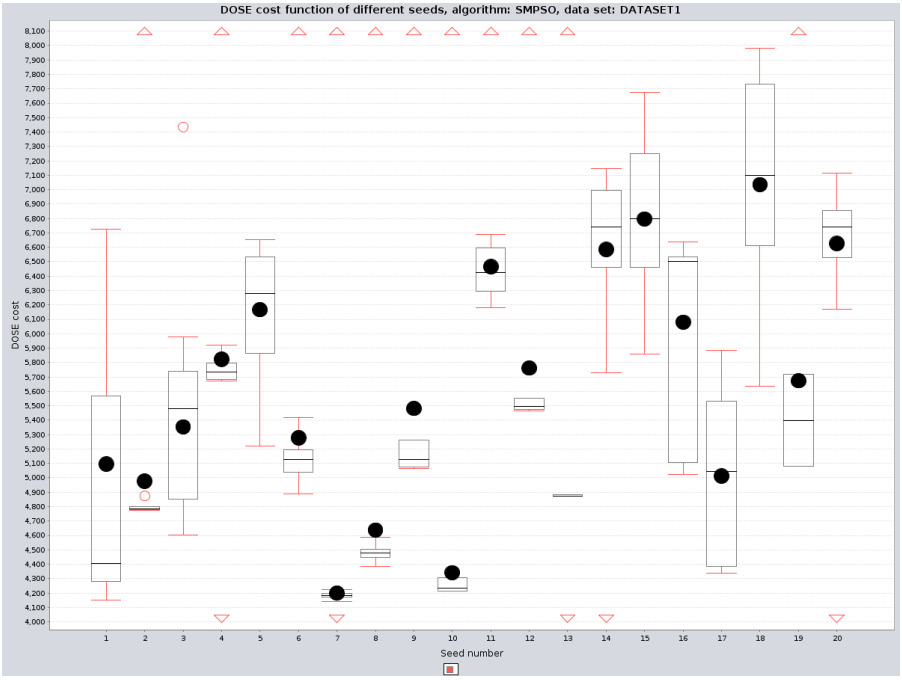


Figure 39: SMPSO: the first data set DOSE cost characteristics by seed.

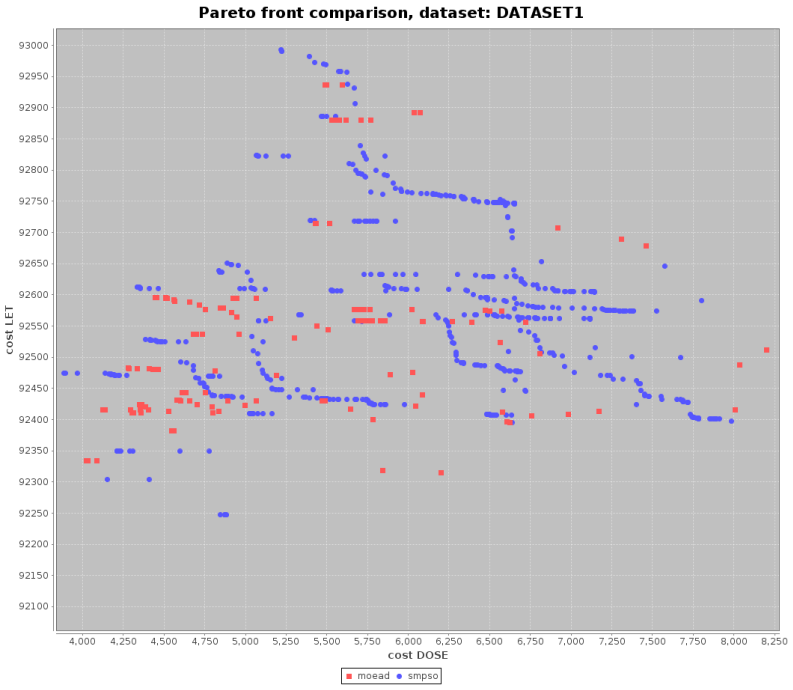


Figure 40: The comparison of SMPSO and MOEA/D Pareto fronts, data set 1

On the second data set, SMPSO did not perform so well. 17 tests provided any solution with a DOSE value below 6000, but only 6 of them reached the value below 5500. From this point of view, SMPSO did not outperform MOEA/D, neither OMOPSO or RVEA. Also here we can observe some large-scaled seeds and the others providing a close set of solutions.

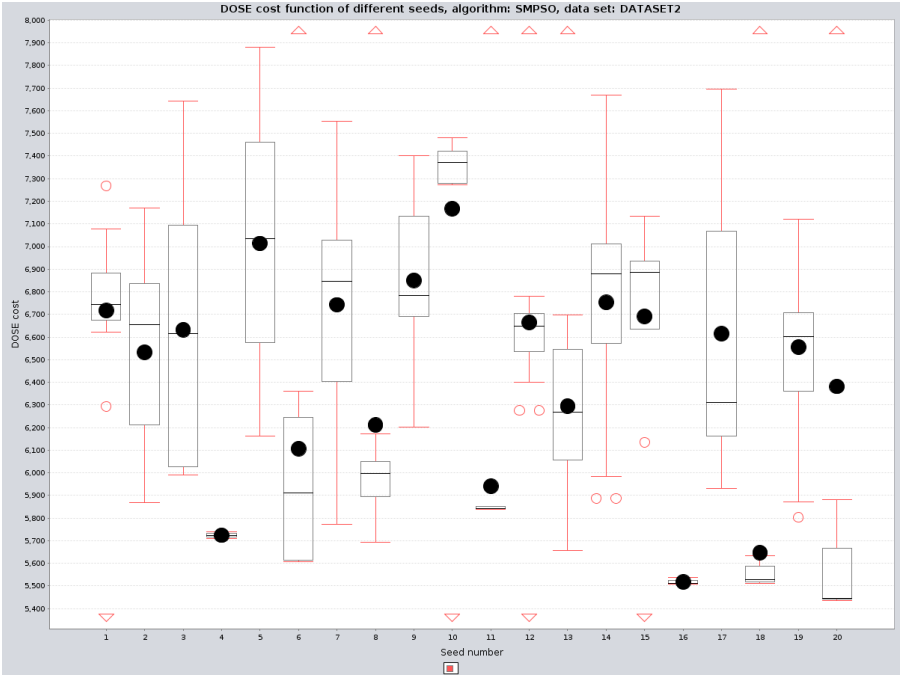


Figure 41: SMPSO: the second data set DOSE cost characteristics by seed.

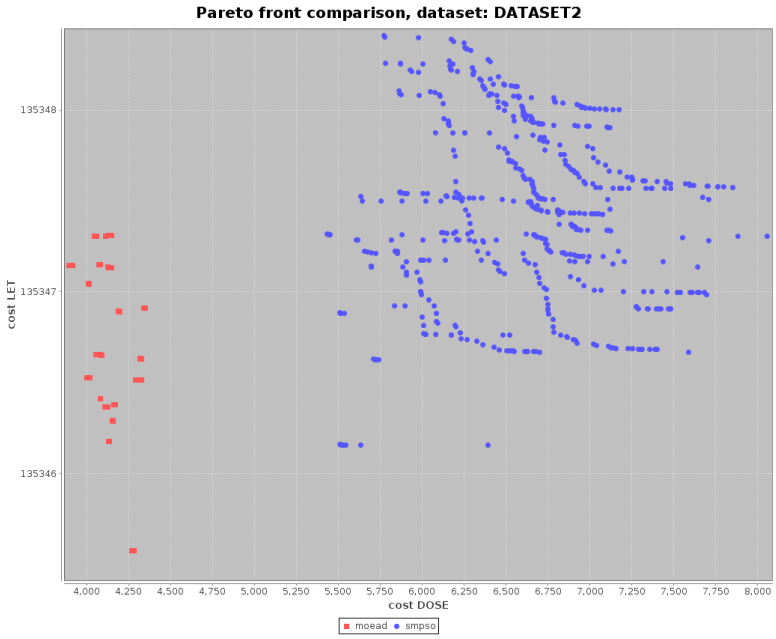


Figure 42: The comparison of SMPSO and MOEA/D Pareto fronts, data set 2

In conclusion, the weakest point of SMPSO seems to be the fact that its performance depends on a given data set. The differences between data sets are so significant that the algorithm can not be considered as a reliable one - while it seems to be the best one on one data set, it provides quite poor results on another one. What is more, neither the comportment of different seeds is very variable. Running this algorithm, we can obtain excellent results, however, the probability of obtaining the poor (or at best average) solution is quite high. The strong point could be a relatively stable population size and good distribution of solutions. Regarding the relative generational distance, in the most of cases, the generations move for a certain number (in general between 50 and 100) of iterations. This movement period on appear at the beginning, at the end but also in the middle of run. For the rest of iteration, population do not change at all, or its changes are negligible.

PESA2

This algorithm provides stably big populations having usually between 40 and 80 very well distributed solutions. Observing the relative generational distance, the algorithm generally converge after at most 50 iterations, the RGD is the mostly zero even though some rare negligible changes can appear between two iterations. It seems to be quite independent on a given data set, providing medium or large-scaled populations in DOSE cost function value.

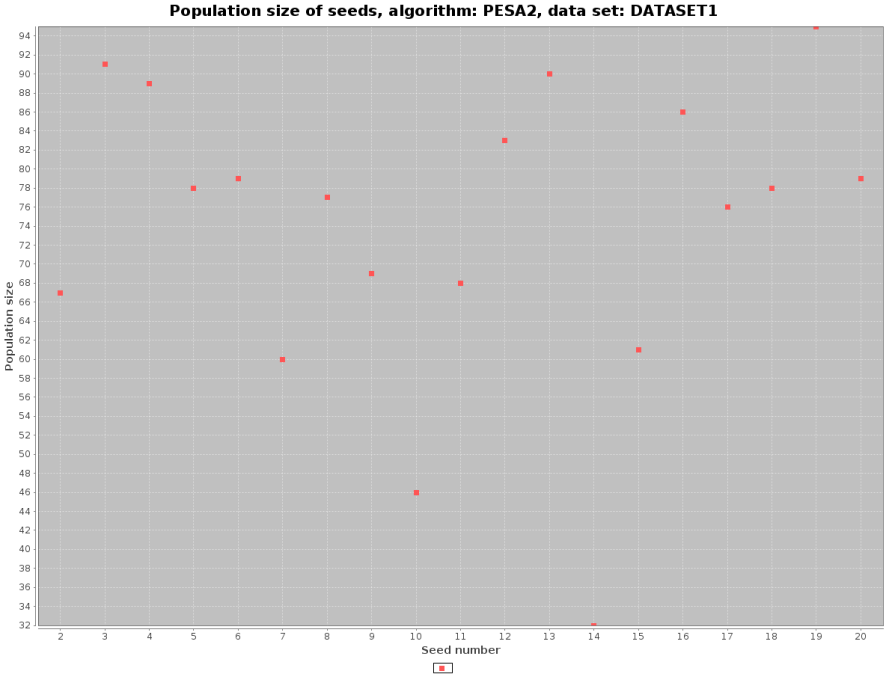


Figure 43: PESA2: the first data set population size by seed.

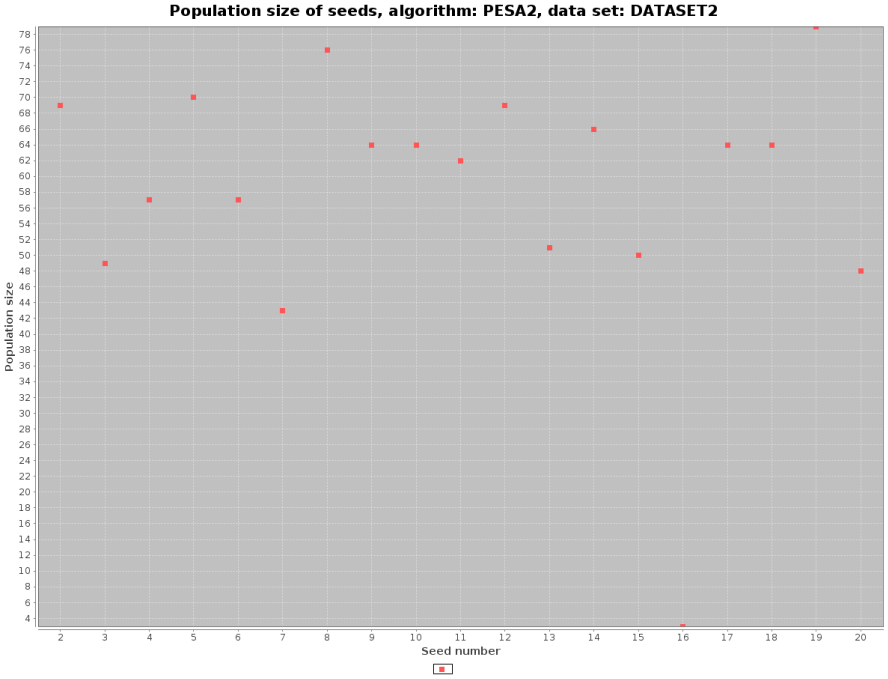


Figure 44: PESA2: the second data set population size by seed.

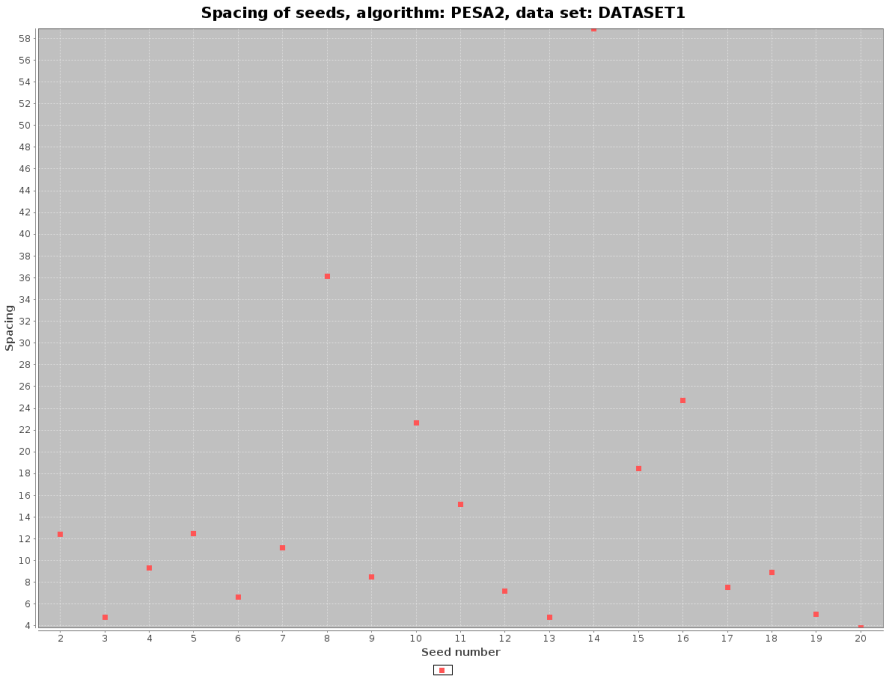


Figure 45: PESA2: the first data set spacing value by seed.

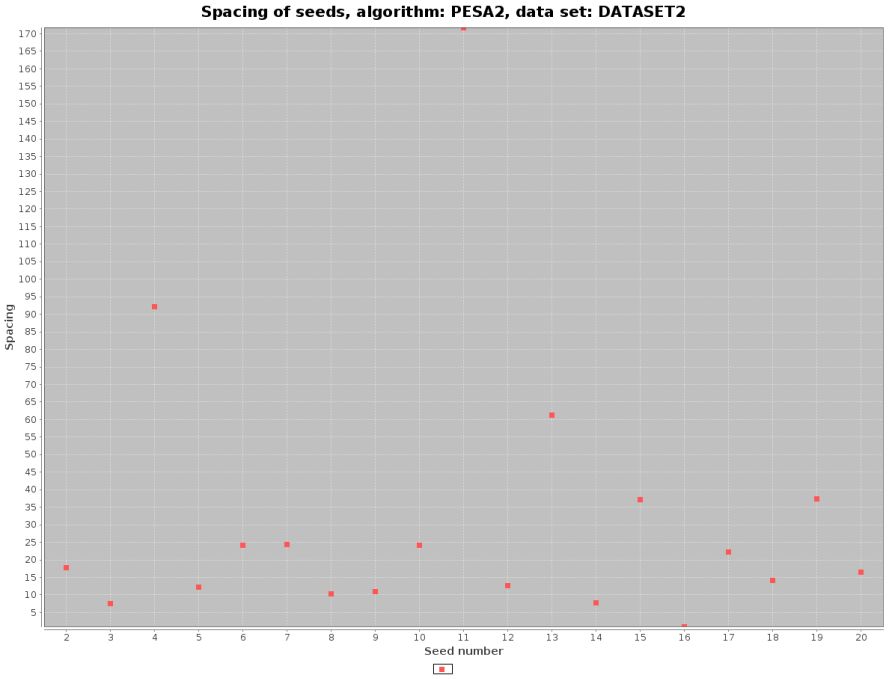


Figure 46: PESA2: the second data set spacing value by seed.

However, this algorithm can not be classified as a performing one. On the first

data set, only 7 tests provided any solution with a DOSE cost function value below 6000. The situation was better on the second data set where all the tests reached the value of 6000 and 9 of them provided any solution with a DOSE value between 5100 and 5500.

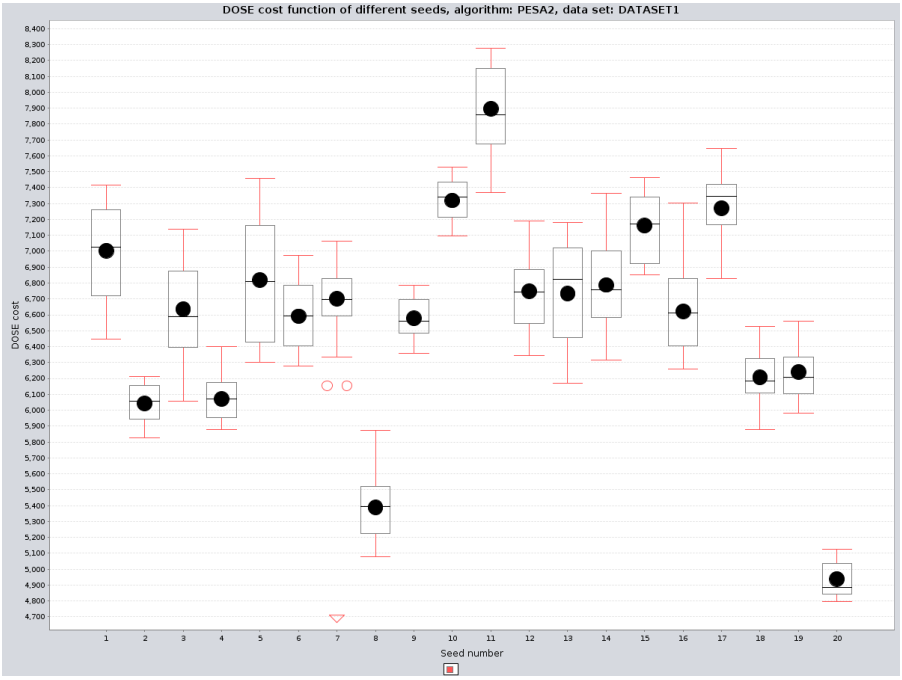


Figure 47: PESA2: the first data set DOSE cost characteristics by seed.

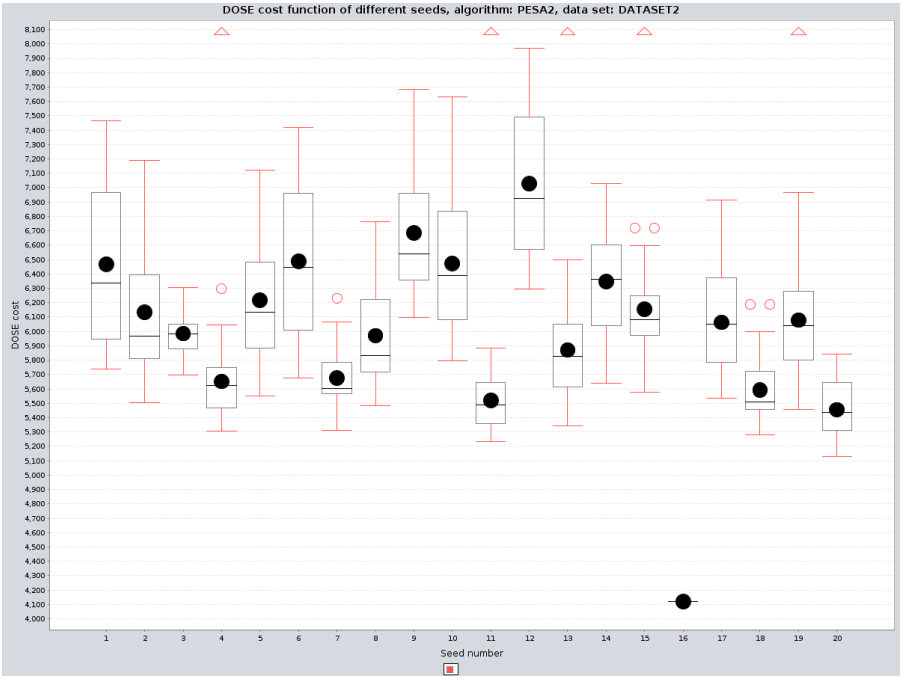


Figure 48: PESA2: the second data set DOSE cost characteristics by seed.

Nevertheless, we should mention that regarding the LET values, this is probably the best algorithm in both cases. However, we can almost always find an algorithm that performs almost as well as PESA2 in LET, but considerably better than PESA2 in DOSE.

SPEA2

SPEA2 does not differ a lot from PESA2. Its population size and spacing characteristics are practically identical, the same can be said on the relative generational distance and population scale in reached DOSE values.

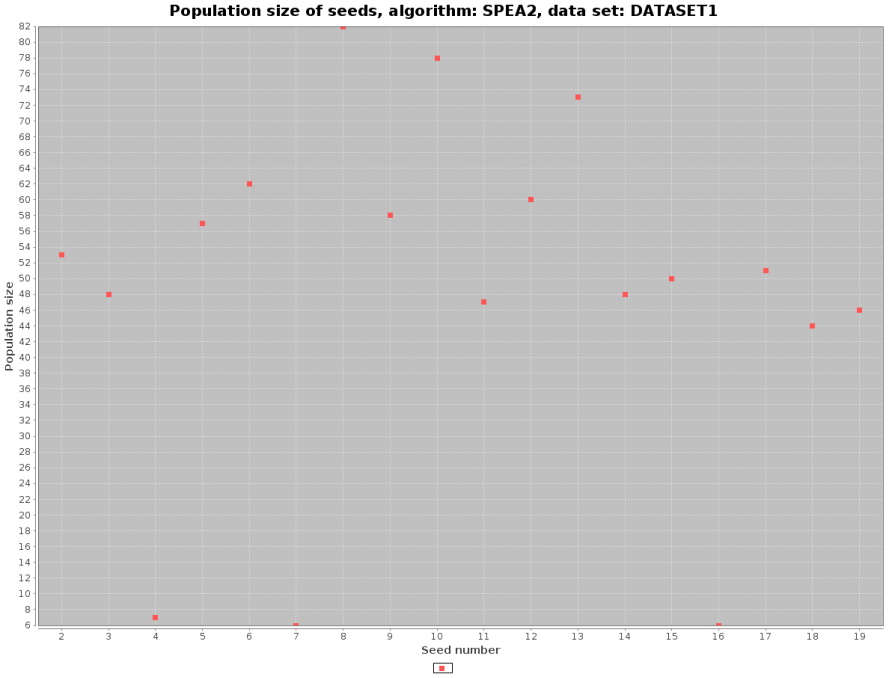


Figure 49: SPEA2: the first data set population size by seed.

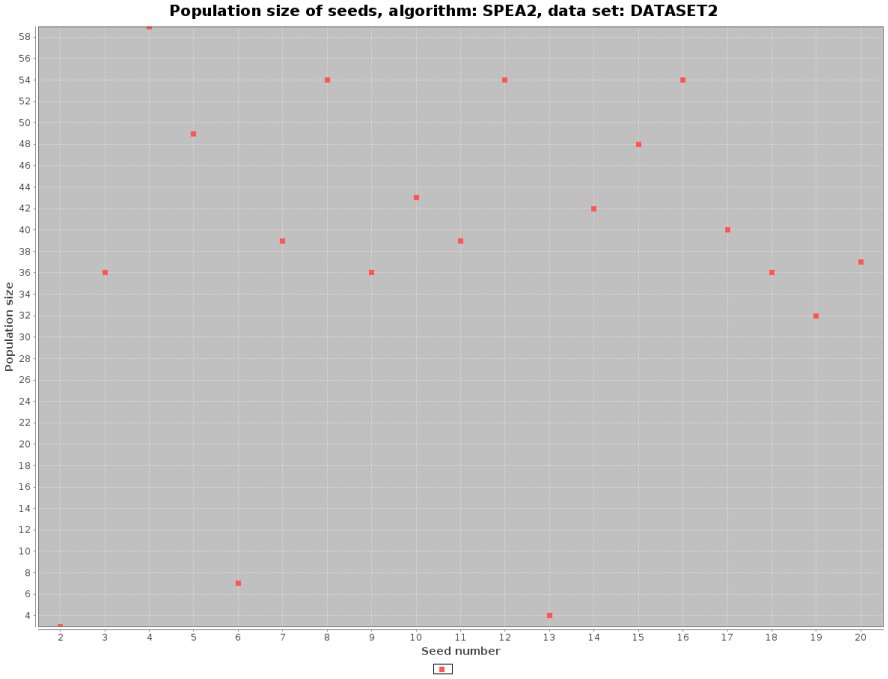


Figure 50: SPEA2: the second data set population size by seed.

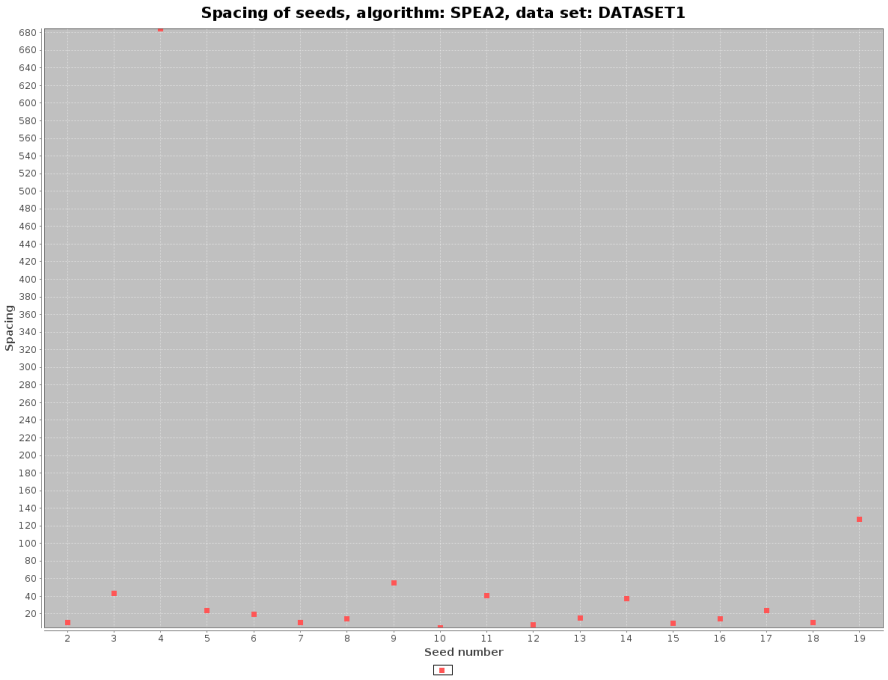


Figure 51: SPEA2: the first data set spacing value by seed.

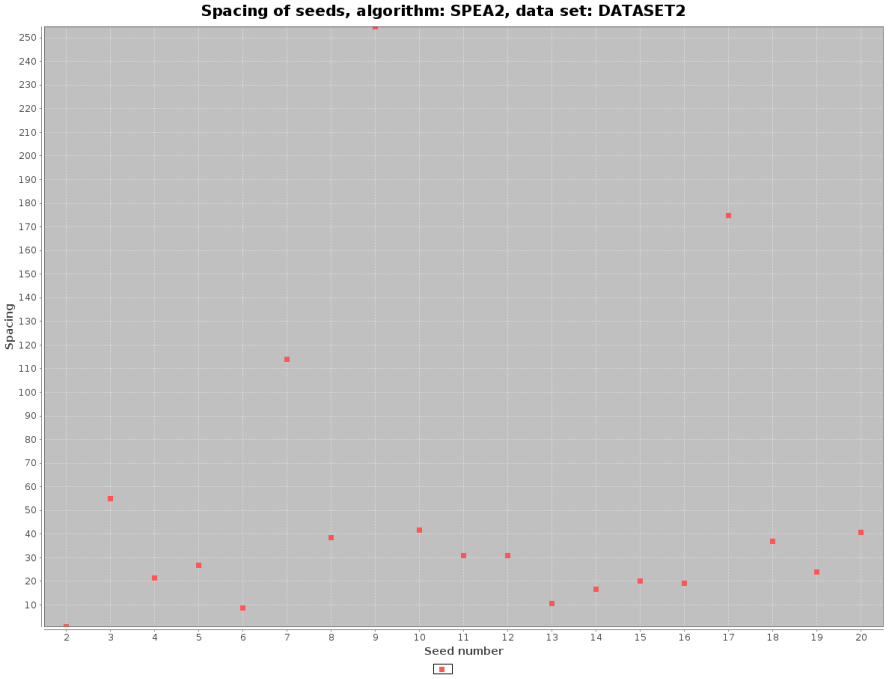


Figure 52: SPEA2: the second data set spacing value by seed.

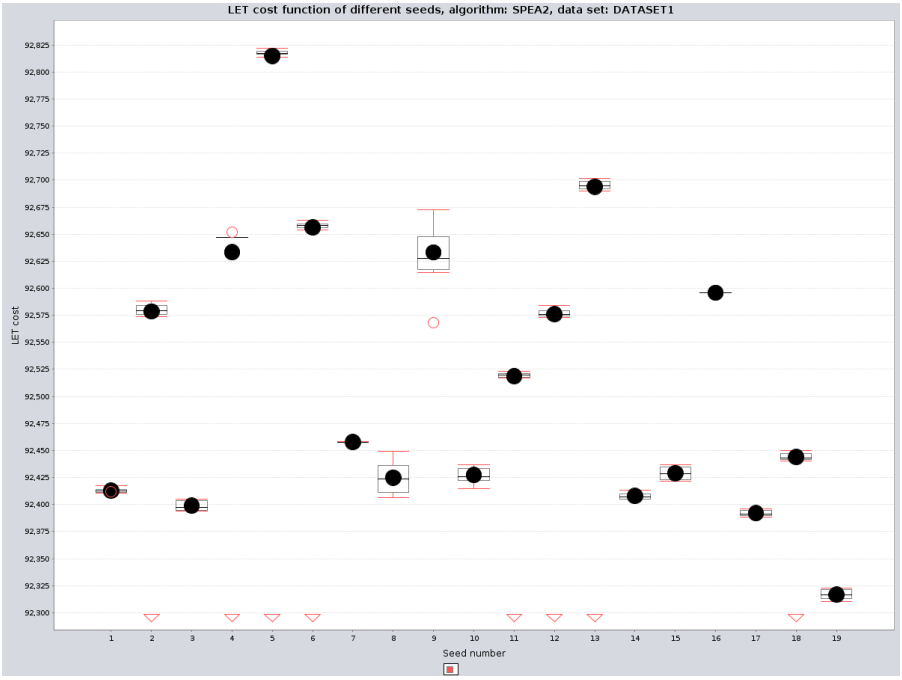


Figure 53: SPEA2: the first data set DOSE cost characteristics by seed.

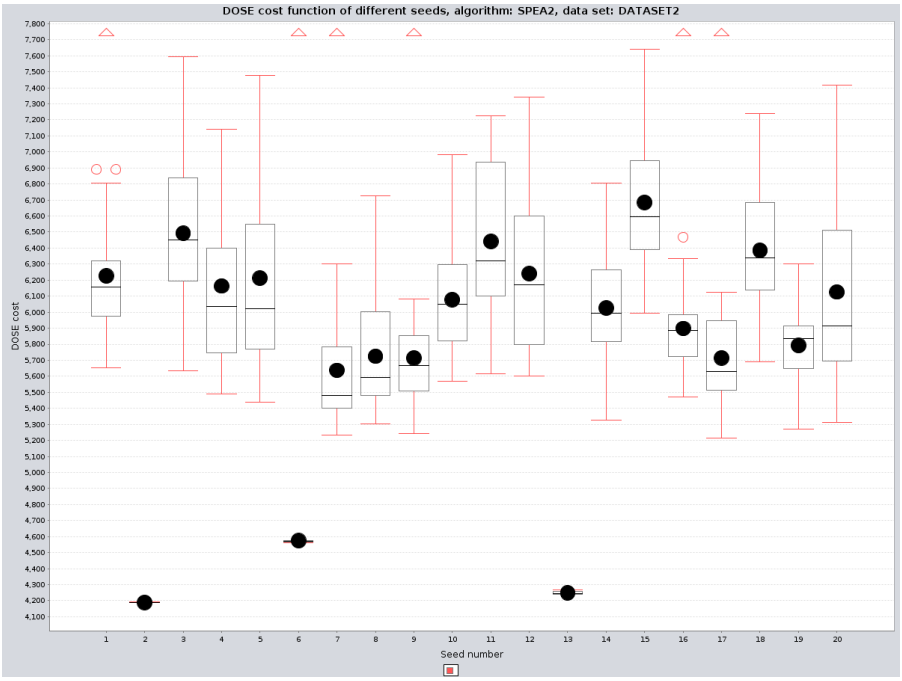


Figure 54: SPEA2: the second data set DOSE cost characteristics by seed.

On the first data set, only about a half of seeds provided any solution with a

DOSE value below 6000, on the second data set, the situation is very similar to those of PESA2. Although, there were a few number of solutions in both cases that reached the dose value below 4500 in the first case and near the 4200 in the second one. These solutions could compete to those provided by MOEA/D, however, they are so rare that they have no importance in practice.

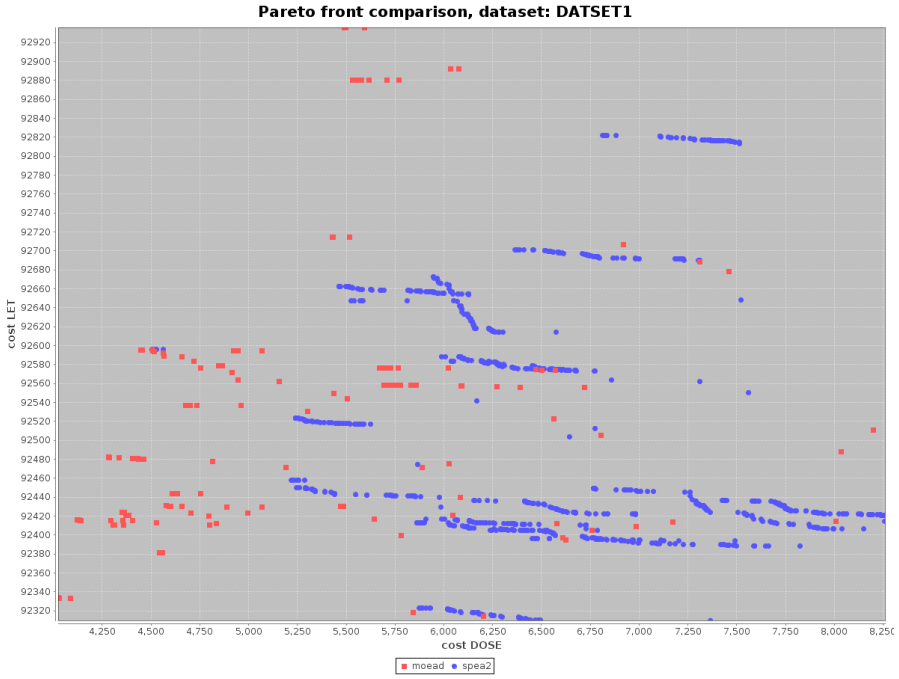


Figure 55: The comparison of SPEA2 and MOEA/D Pareto fronts, data set 1

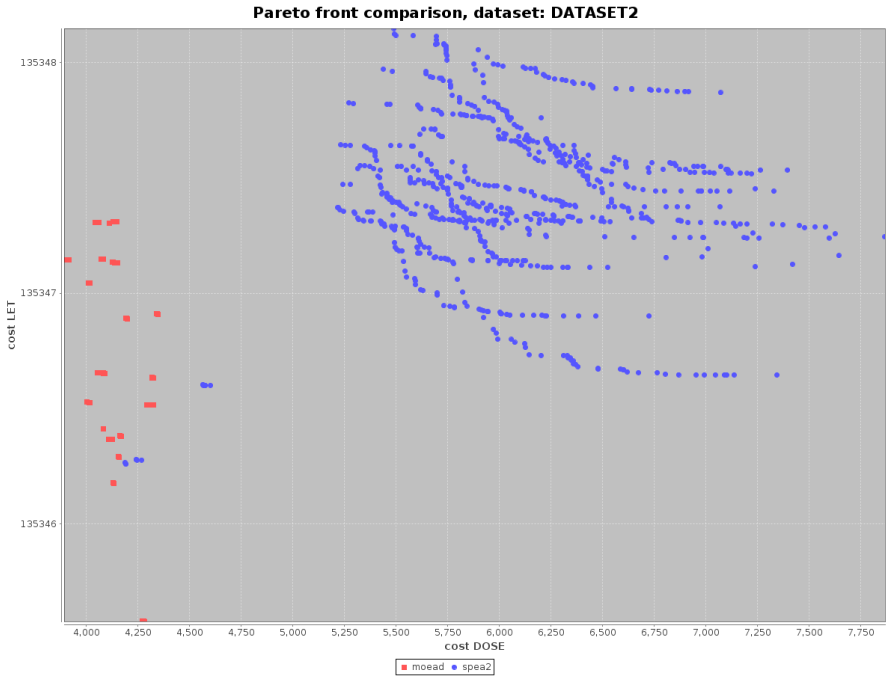


Figure 56: The comparison of SPEA2 and MOEA/D Pareto fronts, data set 2

PAES

PAES algorithm seems not to have any strong point. It provides small populations containing from 2 to 8 individuals in both cases. And, first of all, there are practically no solutions with a DOSE value below 6000, apart some rare isolated ones.

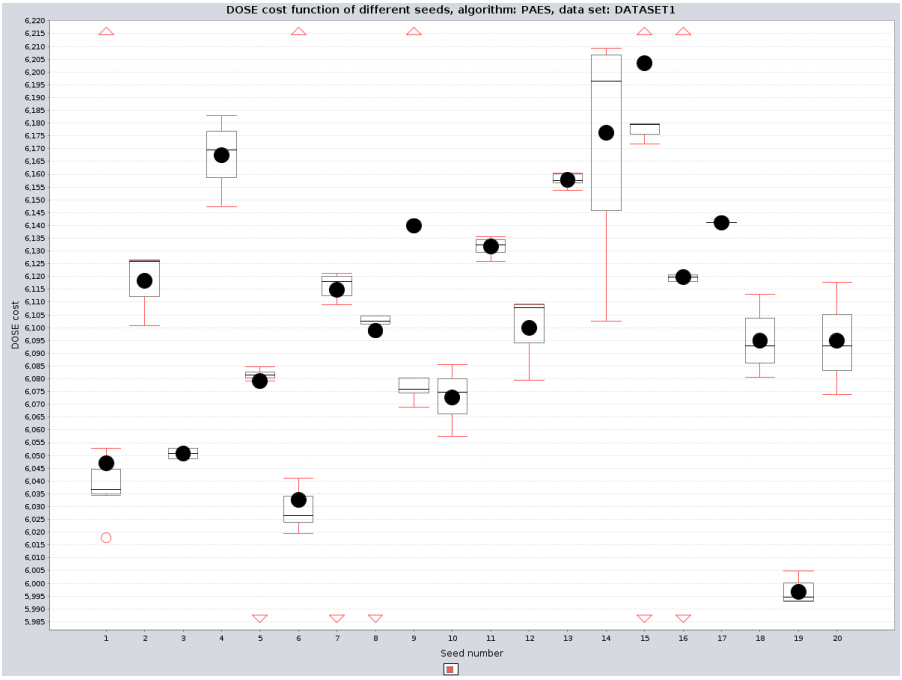


Figure 57: PAES: the first data set DOSE cost characteristics by seed.

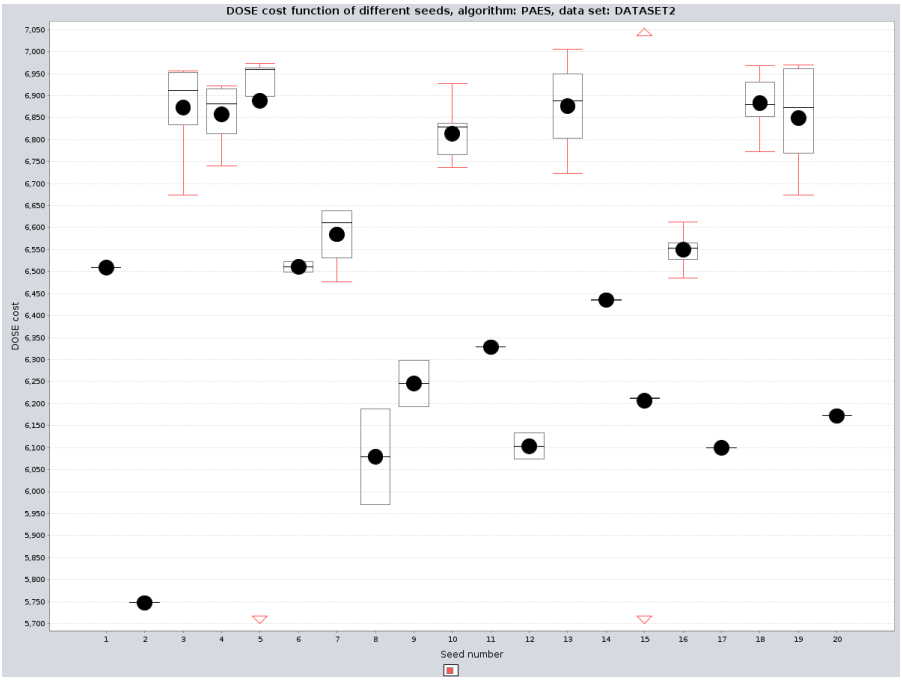


Figure 58: PAES: the second data set DOSE cost characteristics by seed.

Regarding a generational distance, we can observe in about a half of cases that

the population is varying all the time. However, in all the other cases, the relative generational distance is almost constantly zero, except several iterations. Thus, we can not hope that the algorithm performance would improve with an increasing number of iterations.

SMS-EMOA

This algorithm was the poorest one in both cases. As for the previously mentioned algorithm, there were almost no solutions with a DOSE value below 6000.

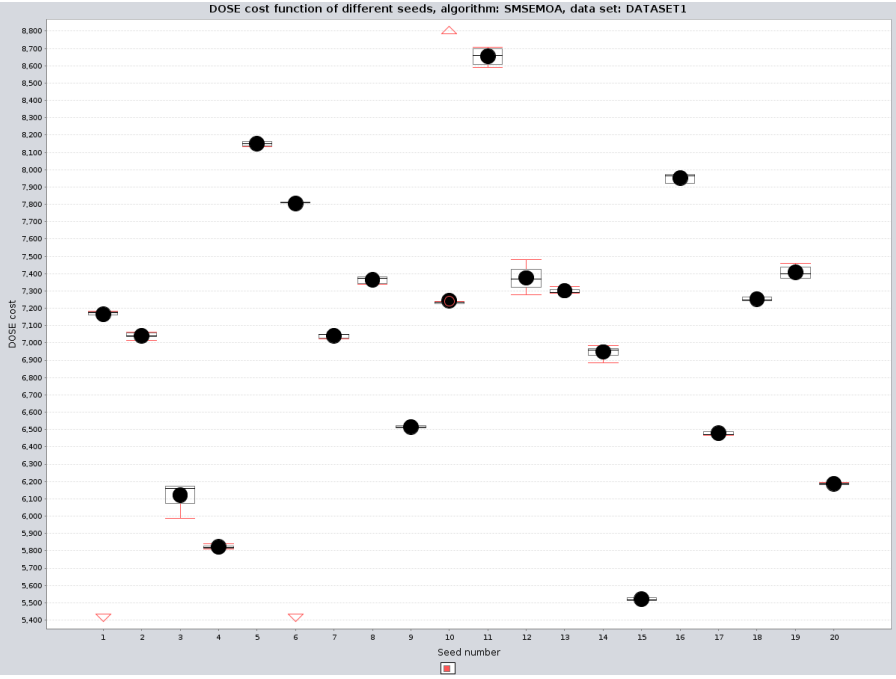


Figure 59: SMS-EMOA: the first data set DOSE cost characteristics by seed.

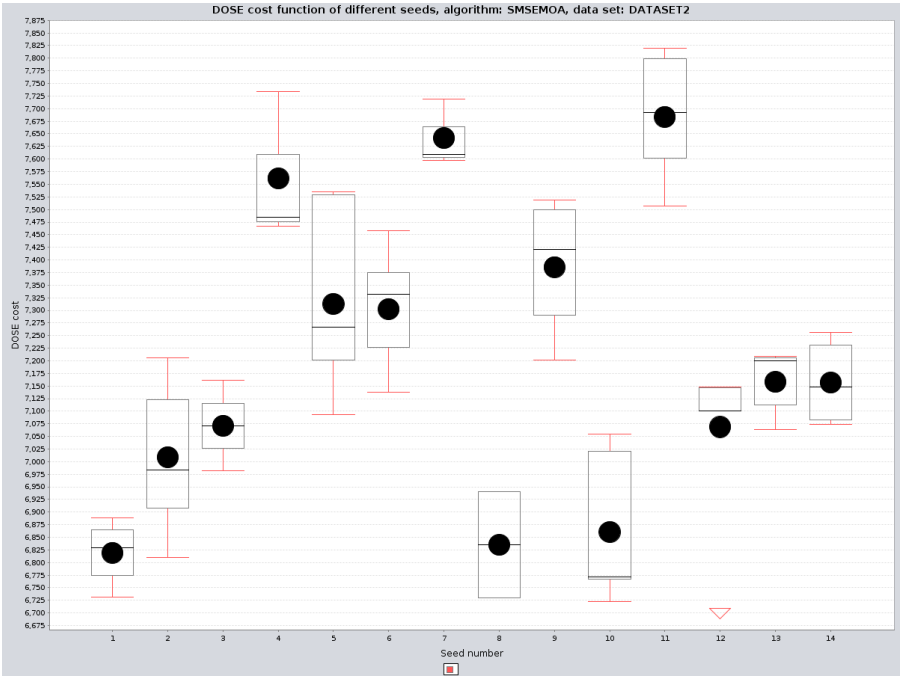


Figure 60: SMS-EMOA: the second data set DOSE cost characteristics by seed.

Conclusion

There were 8 multiobjective evolutionary algorithms tested on 2 different data sets of our problem. After analyzing the results, we note that classical decomposition-based algorithms, and especially MOEA/D, were the most performing ones. The weak point of these algorithms is small population size.

Particle swarm optimization algorithms have shown a promising performance. Their main qualities are the population size and a very good distribution of solutions. Another strong point is that they converge quite fast in comparison with decomposition-based algorithms. OMOPSO performs indifferently on both data sets, however, it can not compete with decomposition-based algorithms. SMPSO has provided excellent results fully comparable to those given by MOEA/D, but unfortunately only on the first data set. The difference between the results given by the both data sets are so significant that we can not consider the SMPSO algorithm as a stably performing one. Also, these algorithms sometimes provides some particular results completely different from the commonly obtained ones.

Classical Pareto dominance-based approaches are remarkable for a great stability of quality indicator and results obtained in different seeds, big population, very good distribution of solution and the independence of given data set. However, they can not compete with previously mentioned algorithms. As algorithms in this group performed practically identically, we don't suggest to test another algorithms of this type. However, it would be very valuable if we could improve their solution quality while keeping all mentioned strong points.

PAES did not show any strong point and seems to be completely unsuitable for our purpose. However, as we tested only one evolutionary algorithm, we can not definitely conclude that these algorithms are not suitable for our purposes.

Neither SMS-EMOA algorithm did not show any strong point, so it seems to be completely unsuitable for our purpose. As in the previous case, we can not definitely conclude that indicator based algorithms are not suitable for our purpose.

To strengthen (or disprove) these conclusions, tests should be performed on more data sets. For further work, there are several possible ways of continuation. Another variants of MOEA/D algorithm, such as MOEA/DD or MOEA/D-M2M could be implemented and tested. As these algorithms were designed principally to improve MOEA/D, there is a real chance to improve our results. To test another decomposition-based algorithms such as IBEA or NSGA-III-OSD seems to be a good idea. Especially the NSGA-III-OSD could be promising, as it is based on the classical Pareto dominance-based NSGA-II and NSGA-III, thus could conserve the qualities of this class of algorithms.

Another goal can be to focus on particle swarm optimization algorithms, try to divest them of their weak points and create algorithms able to compete MOEA/D. To give an concrete example, it could be very useful to find the reason of SMPSO performance instability. Eventually, we could test another similar algorithms, for example algorithms inspired by the comportment of a

bee swarm, fish schools or maybe ant colony. There is a real chance that algorithms of this type outperform decomposition-based ones.

As already mentioned, improvement of classical dominance-based algorithms, like for example in the case of NSGA-III-OSD, could lead to very good results.

Last but not least, we should mention that whatever algorithm is used, all the decisions are made with the help of diverse metrics and quality indicators. We should not forget that the performance of these indicators influence the performance of algorithms. Even though this question is mostly mathematical an is not directly related with the subject of this project, any improvement of existing or creation of new indicators, especially of those that do not require the knowledge of the real Pareto front, could have a positive impact on algorithm performance.

Bibliography

Simon, Dan. *Evolutionary Optimization Algorithms: Biologically-Inspired and Population-Based Approaches to Computer Intelligence.* John Wiley & Sons, Inc, 2013.

Carlos A Coello Coello, David A Van Veldhuizen, and Gary B Lamont. *Evolutionary algorithms for solving multi-objective problems, volume 242.* Springer, 2002.

Tan, K. C., et al. *Multiobjective Evolutionary Algorithms and Applications.* Springer Science Business Media, Inc, 2005.

J.-E. García-Ramos et al. (eds.). *Basic Concepts in Nuclear Physics: Theory, Experiments and Applications, Springer Proceedings in Physics*, 182,DOI 10.1007/978-3-319-21191-6_2. pages 55-84.

Antoniucci, Guido Arnau. (2016)*Analysis of the Distribution of Pareto Optimal Solutions on various Multi-Objective Evolutionary Algorithms* (Bachelor Thesis). University College of London.

Riquelme, Nery, et al. "Performance Metrics in Multi-Objective Optimization." *2015 Latin American Computing Conference (CLEI)*, 2015, doi:10.1109/clei.2015.7360024.