Efficiency Improvement of the GLOBAL Optimization Method by Local Search Changes

Abigél Mester[1], Dániel Zombori[1], László Pál[2], Balázs Bánhelyi[1*]

[1] University of Szeged, Institute of Informatics
Dugonics tér 13, H-6720 Szeged, Hungary
mester@inf.u-szeged.hu; zomborid@inf.u-szeged.hu; banhelyi@inf.u-szeged.hu

[2] Sapientia Hungarian University of Transylvania, Faculty of Economics, Socio-Human Sciences and Engineering, Piaţa Libertăţii nr. 1, 530104 Miercurea Ciuc, Romania; pallaszlo@uni.sapientia.ro

[*] Corresponding author

Abstract: There are many suitable global optimization approaches to find the minimum value of an objective function. In this paper, the improvement of the GLOBAL Optimization Method is studied, which is based on stochastic clustering. Through its three main components, which are sampling, clustering, and local search the algorithm aims to find the global minimum of the objective function. Local search methods significantly influence the efficiency of the GLOBAL method. The efficiency of our proposal may be improved by dividing the system into modules and by creating new variants of both the local and line search methods. The main contribution of this work is to show the achievements of modularization and the efficiency of the new variants of both local and line search methods.

Keywords: GLOBAL; Optimization; Local search; Line search; Modular software

1 Introduction

There is a wide range of optimization problems ranging from everyday tasks [1, 2] to economic issues and theoretical chemical problems [3]. Various stochastic, deterministic, and hybrid global optimization methods have been used to solve these problems. For example, in these papers, the authors explore numerical solution techniques for economic and chemical problems, using differential evolution (DE) or genetic algorithm (GA). In recent years, our research group has solved the same problems using GLOBAL. It has already proved its relevance in such diverse problems [4, 5, 6, 7]. Moreover, it even mastered difficult mathematical problems from the field of qualitative analysis of dynamical systems [6, 8, 9,10]. In these works GLOBAL was used to find a feature. For example, we find the parameters of regions that exhibit chaotic behavior.

These problems consist of global and local optimization challenges. During the execution of the global search method, we sweep the entire search space, and local searches are only executed in near areas where potential optimum values can be found. Since global search usually cannot precisely find the global optimum, we have to run it until a stopping criteria is reached. Afterwards, we run local searches, which are based on function evaluations that are mostly very expensive. Therefore, the number of local search runs should be considered. At this point, the clustering is responsible for reducing the number of sample points.

As we start running the global search algorithm, it generates sample points in the search space, which are then clustered around the local optimum's basin of attraction. If a point cannot be added to an existing cluster, we have to start a local search from that point in order to decide which cluster it should be classified to.

As we follow the operation of the search method, we can easily recognize the three main components, which are sampling, clustering, and local search. So we separated, our system into three different modules. This way, any of the algorithms can easily be implemented, improved, or even replaced. By running and testing these modules, we have realized that due to the high number of function evaluations the algorithm spends most of the time executing local searches [11]. Therefore, our aim was to reduce the time spent in local searches. We found two possible methods to achieve our goal. One is to rewrite the local search method, the other is to revise the line search method. Both methods have been implemented and tested.

First of all, we restructured the original Unirandi local search algorithm, so the line search method became an independent module. Afterwards, we created more local search processes, and we were able to develop three different line search approaches based on polynomial interpolation techniques as a result.

2 Environment

2.1 Modularized GLOBAL

Former versions of GLOBAL were available in Fortran, C, and Matlab [5, 12, 13]. To make GLOBAL more efficient to work with an easily extendable optimizer, we needed an object oriented implementation. To achieve that, we implemented GLOBAL in Java and separated the system into the three individual modules described earlier [15]. The two major modules are the clustering and the local search one, and it's the close cooperation of these modules which makes the algorithm efficient.

In the original version, clustering was an integral part of GLOBAL, and the line search was also integrated into the local search method. Due to the modularization,

these are significantly easier to call and customize. During the implementation, we had to ensure problem-free communication between the individual modules. We have solved this by using the Builder pattern.

An advantage of the Builder pattern is that building and parameterization can be automated. An interface is provided where parameters are passed through an XML file and are processed by the Builder methods. A proper Optimizer object holding the specified configuration is created, which can be called by the user through its interface.

2.1.1 Builder Pattern

The Builder design pattern ensures the proper module parameterization and helps easy reporting of misconfigurations. For every module ,we need a *Builder* nested class that implements setter methods for the module parameters and the required sub-modules. The builder also implements the *build()* function to produce a correctly parameterized module instance. In this function, we can check for the setting of required parameters, we can set default values, we can check if incompatible settings are present and we can log the configuration or report problems in a principled way.

2.1.2 XML Configuration

In a former version of the Java implementation, a lot of function calls were required to set the parameters one after the other. Not just the code was hard to read this way, but there were no efficient ways of testing larger sets of functions with different parameters. To solve this, we have developed a simple configuration building system that takes an XML configuration file and automatically generates the necessary function calls. This system automatically exposes all modules and their parameters to the XML file, relying only on the module structure. In this way, plenty of different parameterizations can coexist in XML files. Finally, it is much more readable when we are reviewing our code. The development of a graphical configurator also became much easier.

In the XML sample code below, the construction of the system is a lot more comprehensible. GLOBAL has search parameters such as sample size, and it needs clustering and local search modules. The XML tags correspond to parameters in the configuration tree. The root is the GLOBAL optimizer module, the implementing Java class is defined in the *class* attribute. The tags under Global hold the parameters identified by the tag names. The literal parameters can be one of the *double*, *long* and *string* types. They will result in a function call on their parent module, for example, the *NewSampleSize* parameter will become *moduleBuilder.setNewSampleSize(longValue)*. Sub-modules are similarly easy to define, Global's *LocalOptimizer* parameter will be set to a local optimizer object. The sub-modules Java class is specified in the *class* attribute, the sub-module is

built like the root module, in a recursive way. When a module's parameters are set, the Builder will output a configured object.

```
<? xml version ="1.0" ?>
<Global package =" org.uszeged.inf.optimization.algorithm "
        class =" optimizer.global.serialized.SerializedGlobal ">
        < NewSampleSize type =" long "> 50 </ NewSampleSize >
        < SampleReducingFactor type =" double "> 0.04 </SampleReducingFactor >
        < LocalOptimizer class =" optimizer.local.parallel.NUnirandiCLS ">
                < MaxFunctionEvaluations  type =" long "> 100000
                </ MaxFunctionEvaluations >
                < RelativeConvergence type =" double "> 1e-8
                </ RelativeConvergence >
                < LineSearchFunction class =" optimizer.line.parallel.Fminbnd5 ">
                </ LineSearchFunction >
        </ LocalOptimizer >
        <Clusterizer
        class =" clustering.serialized.SerializedGlobalSingleLinkageClusterizer ">
                <Alpha type =" double "> 0.01 </ Alpha >
        </ Clusterizer >
</ Global >
```

The class attribute of the node points to the Global class in the Optimizer package. The first subnode is NewSampleSize, the type is long and has a value of 50, which means that the optimizer will randomize at most 50 starting points for local search in the search space. The second argument is the SampleReducingFactor with type double and a value of 0.04. This means that 4% of the NewSampleSize (the default value is 100%, or 1) will be selected from the generated sample points. In this case, it is the best 2 samples. The LocalOptimizer node contains a class that must be instantiated. The algorithm is specified in the class argument, which must implement the interface specified by Global's *setLocalOptimizer(LocalOptimizer)* function. MaxFunctionEvaluations is a parameter for the local search module, not the whole optimization process. RelativeConvergence is a termination criterion, smaller values represent a longer and more accurate run. The LineSearchFunction submodule has no parameters itself, but it is a parameter of the LocalOptimizer. The Clusterizer is a parameter of the Global class. The Alpha parameter controls the rate at which the clusterizer shrinks its region of influence. The larger the value (in the interval [0;1] inclusive interval), the harder it is to cluster a sample. At 1 clustering is effectively disabled.

2.2 Local Search

Local search is a crucial part of the GLOBAL algorithm, hence the performance depends a lot on the attached local search method. GLOBAL randomly chooses points in the search space, then, by evaluating them we can conclude some information about the location of the global optimum. We attempt to create clusters

around the local optima from the sample points, which fall into the local optima's basin of attraction. To assign the unclustered points to a cluster, we have to start a local search from the points which could not be assigned to clusters that were already found based on the applied clustering criteria. However, local search is an expensive technique because it takes many function evaluations, so we would like to limit its use.

The easy change of the GLOBAL algorithm parts was the main profit of the modular implementation. The Local search algorithm is one module of the GLOBAL Java implementation. All described local search methods contain a line search technique which can be considered as a separate module. Three kinds of local search and three variants of line search methods were implemented in the new Java version. Therefore, nine combined local and line search versions could be chosen by the users. Now the algorithms and the efficiency of these new variants will be discussed as follows.

2.2.1 Unirandi

The Unirandi [16] local search method was originally part of GLOBAL. We have updated it so that arbitrary line search technique can be attached. In this way, we created UnirandiCLS, Unirandi with Custom Line Search. The pesudocode can be followed in Algorithm 1.

Unirandi performs line search along randomly generated directions. A trial point is computed based on the current point, on the actual generated direction, and on a step length parameter. If the current point fails to reduce the best function value the negative direction will be tried. The algorithm decreases the step length parameter after two consecutive failures along two generated directions. In a successful case (the actual function value is smaller than the best one), an arbitrary line search technique can be attached which performs further function reductions.

Algorithm 1 Unirandi local search method with custom line search

Step 1. While the maximal number of function evaluations is not reached or the change in function values or vectors are larger than a threshold value do:
Step 2. Generate a random direction.
Step 3. Run a line search algorithm.
Step 4. If line search succeeded go to Step 1.
Step 5. Turn towards the opposite direction.
Step 6. Run a line search algorithm.
Step 7. If line search succeeded go to Step 1.
Step 8. Increase the step length and check the number of iterations, if it is less than 2 go to Step 1.
Step 9. Decrease the step length and set the number of iterations to zero.
Step 10. End while.
Step 11. Save new optimum point.

2.2.2 Rosenbrock

The Rosenbrock [17] method achieves nonlimited search directions by rotating the axes, but still searching along them. The essence of the method is that it rotates one of the axes towards the most favorable direction and it continues searching along the rotated coordinate system's axes.

After a successful step, it increases the step length. After an unsuccessful step, it decreases the step length and turns in the opposite direction. This process is continued until at least one function evaluation becomes successful in each coordinate direction. After this, the axes are rotated again. The stopping criteria are examined after each transformation.

The Rosenbrock method is rotating the axes towards the best vector using the Gram-Schmidt orthogonalization process. The best vector is determined by the sum of the starting vector and the best vector.

Algorithm 2 Rosenbrock local search method with custom line search

Step 1. While the maximal number of function evaluations is not reached or the change in function values or vectors are larger than a threshold value do:
Step 2. For: iterate through every dimension.
Step 3. Do:
Step 4. Run a line search algorithm.
Step 5. If the line search is not successful: turn towards the opposite direction.
Step 6. While line search is not successful and maximum one direction is tested at the same time.
Step 7. If we don't have an unsuccessful step in any coordinate direction: rotate coordinates.
Step 8. Else: halve step length.
Step 9. End for.
Step 10. End while.
Step 11. Save the new optimum point.

2.2.3 NUnirandi

A disadvantage of the Unirandi local search method is that it is not effective on ill-conditioned problems. These problems can be characterized by long, and almost parallel contour lines, hence function reduction can only be achieved along a few directions. By searching along in random directions Unirandi has difficulty finding good directions, especially on high dimensional problems. Many problems, especially real-life ones have an ill-conditioned nature, so it is beneficial to implement methods that can cope with this kind of problem.

Both Rosenbrock and NUnirandi [18] (New Unirandi) use the advantages of random directions, which idea comes from Unirandi. But NUnirandi, just like

Rosenbrock, follows the direction in which we will hopefully find the optimum. Once we have found a good direction, we do a few more function evaluations there [12].

The algorithm is mostly the same as Unirandi, described earlier, but with an effective modification in the algorithm. Namely, the method tries to make further line searches along the last two saved pattern directions.

Algorithm 3 NUnirandi local search method with custom line search, supplement to the original Unirandi

Step 10 1/2-a. For: iterate through the last two saved pattern directions:
Step 10 1/2-b. Evaluate the new point based on the best point, step length and direction.
Step 10 1/2-c. Run a line search algorithm.
Step 10 1/2-d. If the line search succeeded go to Step 10 1/2-a.
Step 10 1/2-e. Turn towards the opposite direction.
Step 10 1/2-f. Run a line search algorithm.
Step 10 1/2-g. If the line search succeeded go to Step 10 1/2-a.
Step 10 1/2-h. End for.

2.3 Line Search

An important component of most local search methods is the line search technique. Implementation of the local search method is crucial in terms of function evaluations. Hence, carefully designed line search algorithms are welcomed.

As the line search is a separate module in the new Java implementation, it can be attached easily to the local search method. A line search algorithm should receive a decent direction and a starting point, and in the end, it returns a point with a corresponding function value.

2.3.1 Doubling Stepper

The algorithm you can see below was originally a part of Unirandi, thus we isolated and developed it further. The method moves as far as possible in the search direction until the function stops decreasing. Fast progress is ensured by the duplication of the step length after each successful step -- this is where the method got its name from.

Algorithm 4 Doubling stepper line search method

Step 1. Move by step length towards the search direction and evaluate the new point.
Step 2. While the new function value is smaller than the previous one do:
Step 3. Double step length.
Step 4. Move by step length towards the search direction and evaluate the new point.
Step 5. End while.

2.3.2 Function Fit

We can use the results we obtained from the doubling stepper's unsuccessful steps as base points and fit a curve on them. By fitting a curve near a possible optimum, the minimum point of the curve and the optimum we are searching for can be close. In this way in the case of some functions-which have sections where we can fit a quadratic or biquadratic curve-we can find the optimum way faster.

Algorithm 5 Function fit line search method that fits a curve on three/five points, extension after doubling stepper
Step 5 1/2-a. If: we have enough (three/five) control points:
Step 5 1/2-b. For: iterate through the dimensions:
Step 5 1/2-c. Fit a curve and evaluate its minimum point.
Step 5 1/2-d. End for.
Step 5 1/2-e. Save the last three/five best points.
Step 5 1/2-f. End if.

Fitting on three starting points. If the doubling stepper made at least three unsuccessful steps, it means that we have at least three starting points, where we know the function values. Then, we can fit a quadratic curve, and we can obtain the minimum point of this curve by using the formula:

$$x = b - \frac{\frac{1}{2}\left((b-a)^2(f_b-f_c)-(b-c)^2(f_b-f_a)\right)}{(b-a)(f_b-f_c)-(b-c)(f_b-f_a)} \quad (1)$$

If the minimum value we get with the formula is better than the previously-stored optimum, then we use the new one for further calculations.

Fitting on five starting points. When we are fitting on five starting points, we need some preprocessing before calculating the minimum value of the function's minimum point. First, we remove the duplicated points, then we create the matrix A for elimination. We use Gaussian elimination with partial pivoting. Afterward, we can get the minimum value depending on the number of coefficients using the cubic equation solution (Cardano-formula), quadratic formula, or just simply divide variable c with variable a.

3 Results

Previously, GLOBAL has been compared with common optimization procedures on standard test functions. In these comparisons, the standard Unirandi and NUnirandi were used in GLOBAL with a simple doubling stepper line search [19, 20]. In this work as well the standard test functions were used, e.g. Branin, Goldstein Price, Six-Hump Camel, Zakharov and Hartmann in several dimensions [15]. The directional choice of the local search method was combined with the three-line search methods so that a total of nine different algorithms were tested on

more than 60 standard test functions. Using the stopping criterion used by László Pál et al [11], we were able to compare our results and illustrate the improvement. Therefore, we specified that the number of function evaluations in the global search should not exceed 100.000, and the sample size was set to fifty and the alpha parameter to 0.1. The algorithms were run until they reached a value correct to six decimal places, or 100.000 function evaluations, and we analyzed how many function evaluations (FE) were needed. During the computation, each test function was executed one hundred times and its average was calculated. The success rate (SR) shows how many times the global optimum was reached. Finally, the line search we developed was compared to the original one and shows the improvement in function evaluation (%) between the original line search method and the new one.

Improvement is defined by dividing the results we get from the original method (the one obtained by the doubling stepper) with the results coming from the function fitting. For this reason, when reviewing *Table 1* we need to look for the rows where the percentages are below zero. In these cases, we needed less function evaluations with the new method. The local Rosenbrock search method with the five-point-based fitting line search works better in 78% of solved test cases. For the functions Cigar, Sphere, and Sum Squares, we were able to decimate the number of evaluations because these have sections where their curves are a quadratic function, so that the fitting can find the local optima almost perfectly when the line search direction is correct.

The results show that the algorithm is efficient with this type of function, and if the local optimal environment is different, its efficiency is worse. The Rosenbrock method with the three-point-based fitting allows us to achieve an improvement in 60% of the solved test cases.

Table 1

The Success Rate (SR) and the number of Function Evaluations (FE) using the Rosenbrock local search method on different functions in several dimensions with the percentage of improvement (%) using the various line search techniques

Function		Rosenbrock		Fit3			Fit5		
	dim	SR.	FE.	SR.	FE.	%	SR.	FE.	%
Booth	2	1.00	321	1.00	317	-1	1.00	274	-15
Cigar-40	40	0.23	77772	0.17	71733	-8	1.00	10286	-87
Cigar-5	5	1.00	1592	1.00	1529	-4	1.00	946	-41
Discuss-40	40	1.00	40753	1.00	40759	0	1.00	40597	0
Discuss-5	5	1.00	3801	1.00	3510	-8	1.00	3679	-3
Griewank-5	5	0.30	59450	0.23	58292	-2	0.16	57722	-3
Hartman-3	3	1.00	699	1.00	693	-1	1.00	588	-16
Hartman-6	6	1.00	2549	1.00	2457	-4	1.00	2090	-18
Levy	5	0.43	21142	0.40	17374	-18	0.42	17543	-17
Matyas	2	1.00	375	1.00	360	-4	1.00	289	-23
Perm-(4.1/2)	4	0.22	59960	0.26	54334	-9	0.34	48054	-20

Perm-(4.10)	4	0.30	10670	0.18	11983	12	0.29	10301	-3
Power sum	4	0.03	75427	0.06	54718	-27	0.12	53936	-28
Rastrigin	4	0.02	29074	0.04	22214	-24	0.04	25507	-12
Rosenbrock-5	5	1.00	4780	1.00	5687	19	0.99	4702	-2
Schaffer	2	0.47	45411	0.42	42124	-7	0.52	49749	10
Shekel-10	4	0.93	25034	0.93	18441	-26	0.96	17465	-30
Shekel-5	4	1.00	5346	1.00	6133	15	1.00	5627	5
Shekel-7	4	0.96	16917	0.95	17112	1	0.96	24148	43
Shubert	2	1.00	475	1.00	523	10	1.00	457	-4
Six hump	2	1.00	241	1.00	247	2	1.00	226	-6
Sphere-40	40	1.00	10002	1.00	9870	-1	1.00	3516	-65
Sphere-5	5	1.00	733	1.00	706	-4	1.00	380	-48
Sum squares-40	40	1.00	43011	1.00	42761	-1	1.00	24532	-43
Sum squares-5	5	1.00	816	1.00	836	2	1.00	860	5
Zakharov-5	5	1.00	1035	1.00	1045	1	1.00	986	-5
Zakharov-40	40	1.00	46244	1.00	46487	1	1.00	45293	-2

Since Unirandi does not take advantage of the calculation and use of the favorable direction, this method has not improved as much as the others (see Table 2). When using the five-point-based fitting method, the results are usually a few percent better. Although we have seen a significant improvement with the fitting, there are features where it performs noticeably worse. In 65% and 48% of the solved test cases, the two methods were an improvement.

In all cases, the success rate is mainly not changed. A line search substantially affects the number of function evaluations only in the right direction.

Table 2

The Success Rate (SR) and the number of Function Evaluations (FE) using the Unirandi local search method on different functions in several dimensions with the percentage of improvement (%) using the various line search techniques

Function		Unirandi		Fit3			Fit5		
	dim	SR.	FE.	SR.	FE.	%	SR.	FE.	%
Booth	2	1.00	299	1.00	301	1	1.00	269	-10
Cigar-40	40	0.00	0	0.00	0		0.00	0	
Cigar-5	5	0.24	90044	1.00	73963	-18	1.00	87779	-3
Discuss-40	40	1.00	42550	1.00	32731	-23	1.00	36797	-14
Discuss-5	5	1.00	9774	1.00	7614	-22	1.00	8415	-14
Griewank-5	5	0.38	57091	0.37	56398	-1	0.40	57067	0
Hartman-3	3	1.00	995	1.00	811	-19	1.00	945	-5
Hartman-6	6	1.00	4484	1.00	3979	-11	1.00	5361	20
Levy	5	0.46	18070	0.58	20163	12	0.42	21427	19
Matyas	2	1.00	335	1.00	343	3	1.00	318	-5
Perm-(4.1/2)	4	0.01	23111	0.04	22644	-2	0.03	44854	94
Perm-(4.10)	4	0.00	0	0.00	0		0.00	0	
Power sum	4	0.04	34460	0.05	27285	-21	0.04	16656	-52

Rastrigin	4	0.00	0	0.00	0		0.02	43683	
Rosenbrock-5	5	0.01	79881	0.02	98684	24	0.00	0	
Schaffer	2	0.67	35937	0.61	39623	10	0.54	45265	26
Shekel-10	4	0.98	19713	0.96	15303	-22	0.98	16026	-19
Shekel-5	4	1.00	5276	0.99	7180	36	1.00	5781	10
Shekel-7	4	0.98	21659	0.96	17313	-20	0.96	18538	-14
Shubert	2	1.00	393	1.00	384	-2	1.00	387	-2
Six hump	2	1.00	170	1.00	167	-2	1.00	171	1
Sphere-40	40	1.00	4813	1.00	4833	0	1.00	5471	14
Sphere-5	5	1.00	556	1.00	548	-1	1.00	573	3
Sum squares-40	40	1.00	35566	1.00	33610	-5	1.00	40857	15
Sum squares-5	5	1.00	689	1.00	650	-6	1.00	714	4
Zakharov-5	5	1.00	783	1.00	781	0	1.00	857	9
Zakharov-40	40	1.00	27081	1.00	27313	1	1.00	30281	12

As far as the efficiency increase of NUnirandi is concerned, NUnirandi executed at least 65% of the solved test functions with fewer function evaluations with both line search methods (See Table 3). All test functions achieved fewer evaluations than Rosenbrock. And just as with Rosenbrock, we get more stable results by adjusting to five points.

The performance graphs (see Figure 1) show the percentage of tasks that found the global optimum in a given number of function evaluations. As you can see, the function of fit3 and fit5 is generally above the traditional doubling step in all cases. That is, in addition to a similar number of evaluations, it has already found the global optimum in several cases. Since it is never below the traditional doubling step in terms of test cases, its use should not be harmful in all cases.

Table 3

The Success Rate (SR) and the number of Function Evaluations (FE) using the NUnirandi local search method on different functions in several dimensions with the percentage of improvement (%) using the various line search techniques

Function		NUnirandi		Fit3			Fit5		
	dim	SR.	FE.	SR.	FE.	%	SR.	FE.	%
Booth	2	1.00	332	1.00	334	0	1.00	291	-12
Cigar-40	40	1.00	32283	1.00	30780	-5	1.00	18673	-42
Cigar-5	5	1.00	1503	1.00	1557	4	1.00	1276	-15
Discuss-40	40	1.00	36503	1.00	28495	-22	1.00	32039	-12
Discuss-5	5	1.00	6117	1.00	5110	-16	1.00	5342	-13
Griewank-5	5	0.38	47461	0.37	50819	7	0.46	48210	2
Hartman-3	3	1.00	352	1.00	339	-4	1.00	334	-5
Hartman-6	6	1.00	1141	1.00	1437	26	1.00	1449	27
Levy	5	0.39	16631	0.50	23831	43	0.50	21598	30
Matyas	2	1.00	348	1.00	358	3	1.00	323	-7
Perm-(4.1/2)	4	0.71	44912	0.73	42684	-5	0.79	37474	-17
Perm-(4.10)	4	0.30	8157	0.25	7969	-2	0.26	7469	-8

Power sum	4	0.64	51886	0.77	47936	-8	0.98	39377	-24
Rastrigin	4	0.01	60261	0.01	62353	3	0.05	35100	
Rosenbrock-5	5	0.99	5284	1.00	5093	-4	0.99	4739	-10
Schaffer	2	0.62	38196	0.54	34742	-9	0.61	35514	-7
Shekel-10	4	0.95	16823	0.96	21124	26	0.98	17190	2
Shekel-5	4	1.00	6200	1.00	4685	-24	1.00	6348	2
Shekel-7	4	0.96	17637	0.97	15246	-14	0.95	21256	21
Shubert	2	1.00	470	1.00	425	-10	1.00	456	-3
Six hump	2	1.00	198	1.00	197	0	1.00	178	-10
Sphere-40	40	1.00	4949	1.00	4907	-1	1.00	5576	13
Sphere-5	5	1.00	608	1.00	606	0	1.00	633	4
Sum squares-40	40	1.00	16198	1.00	16098	-1	1.00	14968	-8
Sum squares-5	5	1.00	714	1.00	682	-4	1.00	730	2
Zakharov-5	5	1.00	813	1.00	828	2	1.00	869	7
Zakharov-40	40	1.00	24075	1.00	24100	0	1.00	26702	11

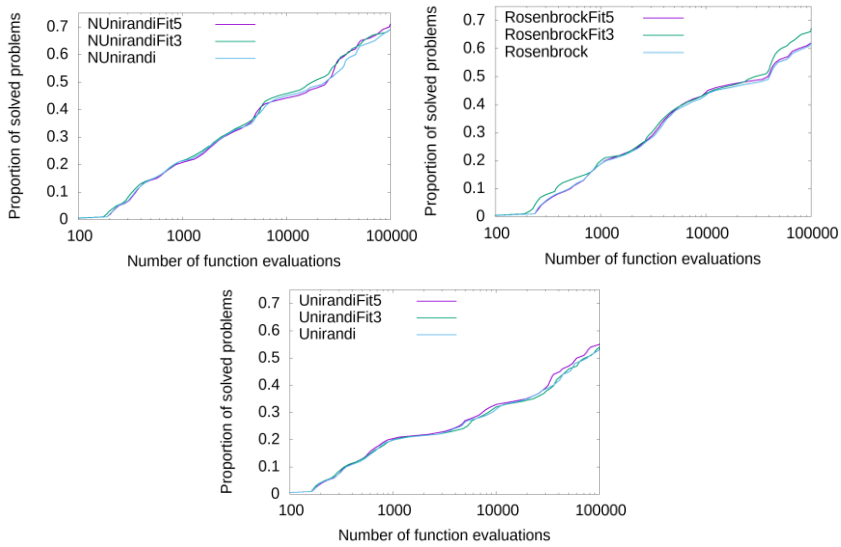

Figure 1

Proportion of the solved problems

Unfortunately, we could not achieve an improvement by adjusting more curves one after the other. So we won't make a further discussion about this task.

Conclusions

We can see that for those functions where the right direction can be found, a quite satisfactory improvement in local search can be achieved. We can therefore say that it is worth using our method and that we have achieved a certain improvement on more than 50-70% of our test functions. This result is not prominent on the whole

test set, as can be seen in the figures. But it is very effective for objective functions with a quadratic-like local optimum. Also, in the other test cases, the number of evaluations will not be much higher.

In summary, the modularization implementation allows us to quickly and effectively test huge amounts of test functions with different combinations. Respectively, with all the new line search versions, the user is not expected to perform worse than the previous version. Unfortunately, there is no clear choice between the new versions, but this modularzation allows us to find the best optimizer for each test function.

Acknowledgement

This research was supported by the projects "Extending the activities of the HU-MATHS-IN Hungarian Industrial and Innovation Mathematical Service Network" EFOP-3.6.2-16-2017-00015, the János Bolyai Research Scholarship of the Hungarian Academy of Sciences, and the Unkp-19-4-Bolyai+ New National Excellence Program of the Ministry of Human Capacities.

References

[1] Banga, J. R., C. G. Moles, and A. A. Alonso, Global Optimization of Bioprocesses using Stochastic and Hybrid Methods, Frontiers in Global Optimization, 45-70 (2003)

[2] Moles, C. G., J. R. Banga, and K. Keller, Solving nonconvex climate control problems: pitfalls and algorithm performances, Applied Soft Computing 5, 35-44 (2004)

[3] Balogh, J., T. Csendes, and R. P. Stateva, Application of a stochastic method to the solution of the phase stability problem: cubic equations of state, Fluid Phase Equilibria 212, 257-267 (2003)

[4] Balogh, J., T. Csendes, and T. Rapcsák, Some Global Optimization Problems on Stiefel Manifolds, J. of Global Optimization 30, 91-101 (2004)

[5] Balogh, J., T. Csendes, and R. P. Stateva, Application of a stochastic method to the solution of the phase stability problem: cubic equations of state, Optimization Letters 2, 445-454 (2008)

[5] Csendes, T., Nonlinear parameter estimation by global optimization efficiency and reliability, Acta Cybernetica 8, 361-370 (1988)

[6] Bánhelyi, B., T. Csendes, and B. M. Garay, A veried optimization technique to bound topological entropy rigorously, Proceedings of the SCAN-2006 Conference, IEEE (2007)

[7] Csete, M., G. Szekeres, B. Banhelyi, A. Szenes, T. Csendes, and G. Szabo, Optimization of Plasmonic structure integrated single-photon detector designs to enhance absorptance, Advanced Photonics 2015, JM3A.30 290 (2015)

[8] Bánhelyi, B., T. Csendes, B. M. Garay, Optimization and the Miranda approach in detecting horseshoe-type chaos by computer, Int. J. Bifurcation 270 Chaos 17, 735 (2007)

[9] Csendes, T., B. Bánhelyi, and L. Hatvani, Towards a computer-assisted proof for $\sum^3$ chaos in a forced damped pendulum equation, J. Computational and Applied Mathematics 199, 378-383 (2007)

[10] Csendes, T., B. M. Garay, and B. Bánhelyi, A veried optimization technique to locate chaotic regions of Hénon systems, J. of Global Optimization 35, 145-160 (2006)

[11] Csendes, T., L. Pál, J. O. H. Sendín, J. R. Banga, The GLOBAL Optimization Method Revisited, Optimization Letters 2, 445-454 (2008)

[12] Pál, L., An Improved Stochastic Local Search Method in a Multistart Framework, Proceedings of the 10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics, 1170 (2015)

[13] Sendín, J. O. H., J. R. Banga, and T. Csendes, Extensions of a Multistart Clustering Algorithm for Constrained Global Optimization Problems, Industrial & Engineering Chemistry Research 48, 3014-3023 (2009)

[15] Bánhelyi, B., T. Csendes, B. L. Lévai, L. Pál, and D. Zombori, The updated GLOBAL optimization algorith: Newly Updated with Java Implementation and Parallelization, book, SpringerBriefs in Optimization, Springer (2019)

[16] Järvi, T., A random search optimizer with an application to a max-min problem, Publications of the Institute for Applied Mathematics, University of Turku, No. 3 (1973)

[17] Rosenbrock, H. H., An Automatic Method for Finding the Greatest or Least Value of a Function, Computer J. 3, 175-184 (1960)

[18] Pál, L., Empirical study of the improved UNIRANDI local search method, CEJOR, Vol. 25, 4: 929{952 (2017)

[19] Pál, L., T. Csendes, MC. Markót, A Neumaier, Black box optimization benchmarking of the global method, Evolutionary computation 20 (4), 609-639, (2012)

[20] Pošík P., W. Huyer, L. Pál, A comparison of global search algorithms for continuous black box optimization, Evolutionary computation 20 (4), 509-541 (2012)