Improved versions of the GLOBAL optimization algorithm and the GlobalJ modularized toolbox

Balázs Bánhelyi[1,a)], Tibor Csendes[1], Balázs Lévai[1], Dániel Zombori[1] and László Pál[2]

[1] *University of Szeged, Institute of Informatics*
[2] *Sapientia Hungarian University of Transylvania, Faculty of Economics, Socio-Human Sciences and Engineering*

[a)]Corresponding author: banhelyi@inf.u-szeged.hu

Abstract. Solving global optimization problems plays a key role in most branches of natural sciences whilst dealing with everyday problems. With the help of optimization algorithms and the exponentially fast growth of the capabilities of the underlying hardware, the scale of feasible optimization tasks is reaching new levels. We help this goal with revisiting *GLOBAL*, a stochastic optimization method aiming to solve non-linear constrained optimization problems by the penalty function approach. It is a versatile tool for a broad range of problems, proven to be competitive in multiple comparisons. With the similarly rapid evolution of programming habits and tools, time naturally passed by *GLOBAL's* latest implementation so that it became a drawback. Now, we present *GlobalJ*, a fully modularized Java framework refurbishing and extending the potential of this algorithm.

INTRODUCTION

In this presentation, we are focusing primarily on non-constrained, non-linear optimization. Formally, we consider the following global minimization problems:

$$\min_{a \le x \le b} f(x).$$

The violation of upper and lower bounds can easily be avoided if candidate optimum points are generated within the allowed interval during the optimization. Everyday tasks include a wide range of such optimization problems.

Generally, global optimization, unless the problem cannot be solved by substituting into a formula, is a continuous, iterative production of optimum point candidates until a stopping criterion is met. GLOBAL generates evaluation points in two different ways; first, random sample points are chosen from the problem space to be used as seed points. Secondly, local searches are started from promising seed points that may lead to new local optima.

In the scenario of multiple seed points converging to the same local minimum, the evaluations are redundant. This phenomena occurs when the samples are chosen from the same region of attraction. The region of attraction is the set of points from where the local search converges to the given x^* local optimum. With the help of low cost computations this inefficiency is significantly reducible. *GLOBAL* utilizes clustering in order to prevent redundant computations, by finding the regions of attraction for local optima and lessening the number of local searches leading to the same optimum. GLOBAL has proved its pertinence earlier in such diversified problems [6], it coped with hard mathematical problems as well, from the field of qualitative analysis of dynamical systems [2, 3].

GlobalJ

Having discussed the theoretical background, we are going to talk about how we extended the latest implementation of the algorithm written in MATLAB into a more general framework [5, 7, 8]. From now on, we will refer to the MATLAB implementation of GLOBAL as GlobalM, and the new framework as GlobalJ in order to tell one from the other.

While MATLAB is a great software for many purposes like rapid creation of proof of concepts and has high-end tool boxes for many fields of science, programs usually do not support integration with it. MATLAB is shipped with an array of tools to generate code in other languages, to package MATLAB code as a shared library, for example a .NET assembly. This will not change the fact that we have to handle a lot of auxiliary tools while our real intent is to solve an optimization problem. The other major problem of the platform is performance. MATLABs own language is interpreted. It is based on the theoretically best algorithms, and its engineers have made a tremendous effort to optimize MATLAB for vector operations, but this cannot replace fastness of native code.

Beyond the platform, the real weakness of GlobalM is its fragile and monolithic implementation. Logically and programmatically independent functions are tangled together. Understanding the operation is difficult, modify it to fit better a given problem is even more of an obstacle. Although, the paper, which introduced GlobalM, claims that only minor efforts are required for simple modifications, today we consider the code tangled together, with not clearly separated functionality. The user has to be familiar with large part of the code base to be able to confidently modify it.

It was time to create a new, easy-to-use implementation, to create GlobalJ. We decided to use JAVA for this purpose, hence the name, because numerous optimization libraries in the scientific community are published in JAVA. Industrial software often provides JAVA APIs for integration, and nonetheless, it is a mature language with a lot of approved libraries. Besides the aim of a recent implementation, an easy to use framework has its own benefits therefore it also became a goal.

Looking over the GLOBAL algorithm [4], the operation can be easily decomposed into several key tasks: generating samples, clustering samples, selecting samples possibly leading to new local optima, and executing local searches started from these points. Beside sample generation, these subproblems are clustering and local searching issues that are independent from each other from the execution point of view. The connection between them is the shared data. The GLOBAL algorithm produces samples and pushes them into a pipeline of clustering and searching operations. The relationship of these functionality suggests to separate the tasks into three distinct modules at the highest level of abstraction. These are the clustering module, the local searching module, and the Global module. The Global module is responsible for providing structure to the operations of the other two and keeps the data flowing in the pipeline (Figure 1).

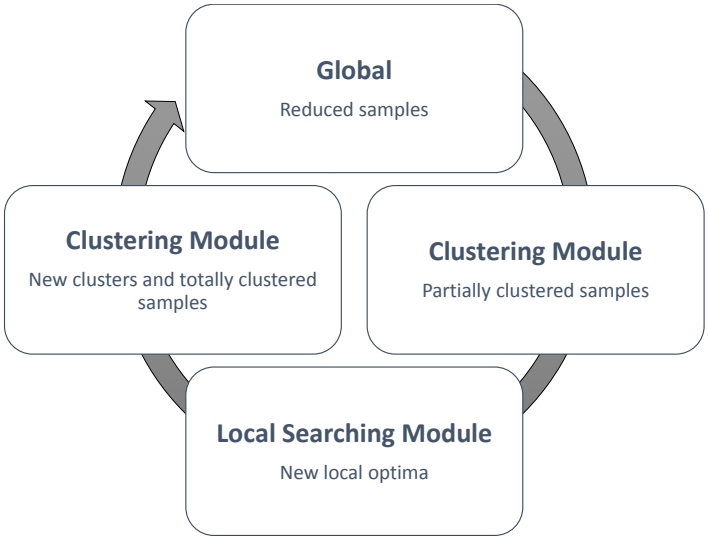

FIGURE 1. Relationship of GLOBAL modules, the pipeline of an iteration.

To achieve great usability, the framework supports any hierarchical modularization in which the modules are freely interchangeable along the interfaces. For GLOBAL the submodules are already given but other optimizer implementations can use different modularization approaches.

Parallel GLOBAL

In case of large scale problems, when the optimization process on a single computer would take years, the application of computational clusters is inevitable. After finishing GlobalJ, our next objective was to add a parallel version of GLOBAL to the framework that can utilize multiple CPU cores simultaneously.

Getting out the most of parallel computing requires the partitioning of computation into as many non-interleaving tasks as possible. In case of the sample generation and the local search phases this is straightforward. Briefly, sample creation is a single function evaluation, while local search is a series of function evaluations. Race condition may only occur when the result of a parallel task is inserted into the shared data structures. These read and write operations must be serialized, the actual computations are thread-safe by their definition.

Clustering is special because it is data intensive. Fortunately, the bulk of these data accesses are read operations, which are not interfering with each other.

Our parallel implementation of GLOBAL combines the advantages of functional decomposition, data decomposition, and the pipeline architecture. The implementation was based on a priority queue that distributes the tasks between the workers. The functional and data decomposition are implemented as multiple logical units and multiple instances of the same logical unit, which are all running simultaneously. From the data point of view, the algorithm is sequential, working like a pipeline, as each data packet goes through a fixed sequence of operations, and multiple packets can coexist.

Since the thread scheduling is performed by the host operating system, there is no guarantee for the order of execution, thus the optimization process is not deterministic even with the same randomization seed. We discussed more details about the parallel version of GLOBAL in our latest publication [1].

Results

We selected some of the famous and widely used test functions to compare GlobalM, GlobalJ, and PGlobal. The implementations operated under the same parametrization. Only robust results are shown where runs found the global optimum every time. The Java versions have a significant reduction in NFEV (Table 1). This is caused by the updated local search scheduling, which causes big differences at low count of local searches. For lower number of seed points per iteration the difference decreases. The function implementations were compared to the equations at *https://www.sfu.ca/ ssurjano/optimization.html* and they can be found in Appendix B of [1].

TABLE 1. Number of function evaluations on various test functions

Function	**Number of function evaluations**			**Dim**
	PGlobal	**GlobalJ**	**GlobalM**	
Beale	762.3	1,060.1	14,709.24	2
Booth	576.6	600.8	595.31	2
Branin	516.1	540.7	829.74	2
Discus-5	18,605.9	19,343.3	59,879.67	5
Goldstein Price	502.4	584.2	577,05	2
Griewank-20	9,847.8	10,185.8	25,198.83	20
Matyas	615.2	646.1	679.84	2
Shubert	517.0	895.1	1169.83	2
Sphere-40	4,083.1	4,118.7	45,417.97	40
Sphere-5	781.9	794.3	976.41	5
Sum Squares-40	24,478.5	24,272.1	303,991.87	40
Sum Squares-5	856.2	867.9	1,500.15	5
Zakharov-40	20,431.5	20,811.9	288,607.88	40
Zakharov-5	953.4	983.9	3,741.72	5
Average	**7,787.9**	**8,045.6**	**53,419.7**	**-**

We also tested the parallelization capabilities of PGlobal up to 16 parallel cores. We achieved 10% to 25% less execution times with easy to compute functions and even 85% drop with slower ones. These numbers can be

further improved if the function is harder to compute (which is very common). The presented data was generated using the Shubert test function. We tested the effects of additional threads and different computational costs of the objective function. The computational cost was increased by computing the function not once but 10^x times for every evaluation request. Besides the clear advantages of multithreading we noticed that additional threads increase the function evaluation count in a linear fashion. This can cause some limitations but we were not able to test the algorithm on much larger scales. For the range of 2 to 100 threads the main efficiency loss is caused by race conditions between threads which (as we shown in Figure 2.) decreases with the increasing computational cost of the objective function.

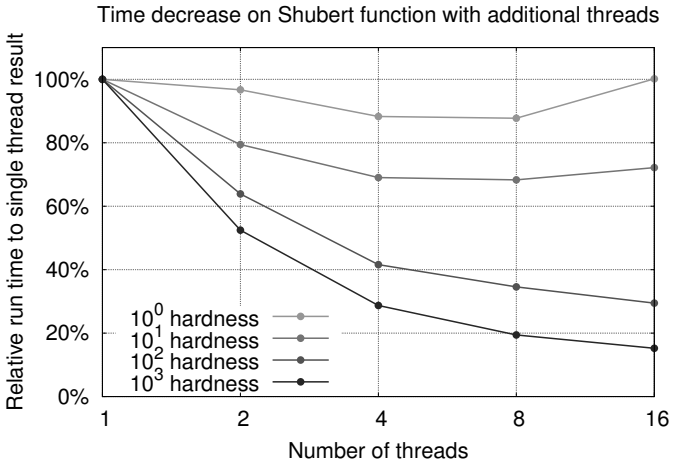

FIGURE 2. Effects of multithreading and computational cost on run time.

ACKNOWLEDGMENTS

This research was supported by the project "Integrated program for training new generation of scientists in the fields of computer science", EFOP-3.6.3-VEKOP-16-2017-0002. The project has been supported by the European Union and co-funded by the European Social Fund.

REFERENCES

[1] B. Bánhelyi, T. Csendes, B. Lévai, L. Pál, and D. Zombori, The updated GLOBAL optimization algorithm, Springer, Submitted for publication (2018).

[2] B. Bánhelyi, T. Csendes, and B.M. Garay, Optimization and the Miranda approach in detecting horseshoe-type chaos by computer, Int. J. Bifurcation Chaos **17**, 735–747 (2007).

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

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

[5] T. Csendes, L. PÃ¡l, J.O.H. Sendin, and J.R. Banga, The GLOBAL Optimization Method Revisited. Optimization Letters, **2**, 445–454, (2008).

[6] M. Csete, G. Szekeres, B. BÃ¡nhelyi, A. Szenes, T. Csendes, and G. Szabo, Optimization of plasmonic structure integrated single-photon detector designs to enhance absorptance. Advanced Photonics 2015, JM3A.30, (2015).

[7] L. Pál, An Improved Stochastic Local Search Method in a Multistart Framework, Proceedings of the 10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics, 117–120 (2015).

[8] J.O.H. Sendín, 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).