BULGARIAN ACADEMY OF SCIENCES

PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS • 2021 • Vol. **76**, pp. 39-46
p-ISSN: 2738-7356; e-ISSN: 2738-7364
https://doi.org/10.7546/PECR.76.21.03

Differential Evolution and Particle Swarm Optimization Efficiency According to Pseudo-Random Number Generator Quality

Gergana Mateeva, Dimitar Parvanov, Todor Balabanov

Institute of Information and Communication Technologies at
Bulgarian Academy of Sciences
Acad. Georgi Bonchev Str., block 2, office 514, 1113 Sofia, Bulgaria
E-mails: gergana.mateeva@iict.bas.bg, dimitar.parvanov@iict.bas.bg,
todor.balabanov@iict.bas.bg

Abstract: There are many evolutionary metaheuristics developed during the last decades. Most of them are highly dependent on the quality of the random numbers involved in calculations. When random numbers are generated with a true random number generator, the problem with the quality of the numbers does not exist. In most practical cases, random numbers are generated with pseudo-random number generators. In such cases, the statistical quality of the generated numbers can influence the optimization process a lot. In this research, the efficiency of Differential Evolution and Particle Swarm Optimization is observed. Research is done according to the quality of the pseudo-random number generated.

Keywords: Pseudo-random numbers, Differential evolution, Particle swarm optimization

1. Introduction

The information technology (IT) nowadays are the base for every company to run their businesses and to be competitive (Borissova et al., 2020). It should be mention also the area of Internet-of-Things (IoT) where complex systems are to be managed and huge amount of data are to be processed (Garvanov et al., 2021). In this regard, the recent developments in ICT strongly recommend usage of cryptography in order to improve the security of such information systems (Trifonov et al., 2018). Many cryptographic algorithms need random numbers used as keys and unique identifiers. Such problems related to the generation of random numerical values is a relatively difficult task. Therefore, pseudo-random number generators are often used (Koeune, 2005). One of the widely used algorithms based on swarm intelligence is the Particle Swarm Optimization (PSO), which is inspirited by nature, and its standard form and variants are presented in (Yang, 2021). Differential Evolution (DE) is one of the powerful and universal evolution optimizers for continuous parametric spaces, and an updated survey is proposed (Das et al., 2016). Due the complexity of real problems, many authors attempt to propose different techniques to cope with such complexities. For example, it is proposed an algorithm that use the advantages of DE and particle swarm optimization (PSO) that ensures the simple operation, easy implementation, and fast convergence of DE (Huimin, et al., 2021). Different characteristics of DE and PSO are used to determine a good point set to make the initial data more uniform. Other authors rely on deep-zoom to improve the pseudo-randomness properties (Machica & Bruno, 2017).

Metaheuristics for global optimization often depend heavily on random numbers to make each run unique (Dillen et al., 2021). In many of these heuristics, initial solutions are randomly generated (the initial population in Genetic Algorithms, the initial population in Ant Colony Optimization, the initial values in Simulated Annealing for example). Random numbers are involved not only during initialization but also it is involved during the optimization process. For example, in Genetic Algorithms, the crossover rate and mutation rate have probabilistic nature (Guliashki et al., 2009). In Ant Colony Optimization, the possible paths are investigated on a probabilistic principle. This study focuses on a hybrid implementation of Differential Evolution and Particle Swarm Optimization. The joint implementation of both heuristics is tested with two different, as a statistical quality of the numbers, pseudo-random number generators.

Differential Evolution is a floating point encoded evolutionary algorithm for global optimization owing to a special kind of differential operator, which invoked to create new offspring from parent chromosomes instead of classical crossover or mutation. Easy methods of implementation and negligible parameter tuning. Differential Evolution starts with a population of search variable vectors. Vectors are referred in literature as genomes or chromosomes. For each search-variable, there may be a certain range within which value of the parameter should lie for better search results. At the very beginning, problem parameters or independent variables are initialized somewhere in their feasible numerical range.

In each generation (or one iteration of the algorithm) to change each population member, a donor vector is created. In DE/rand/1 scheme, to create donor vector three other parameter vectors are chosen in a random fashion from the current population. A scalar number scales the difference of any two of the three vectors and the scaled difference is added to the third one whence the donor vector is obtained. To increase the potential diversity of the population a crossover scheme comes to play (exponential or binomial) (Tanabe & Fukunaga, 2014). The donor vector exchanges its body parts with the target vector. To keep the population size constant over subsequent generations, selection is done to determine which one of the target vector and the trial vector will survive in the next generation. If the new trial vector yields a better value of the fitness function, it replaces its target in the next generation. Otherwise the target vector is retained in the population. Hence the population either gets better remains constant but never deteriorates (Das et al., 2008).

Particle Swarm Optimization is a multi-agent parallel search technique. Particles are conceptual entities, which fly through the multi-dimensional search space (Mirjalili et al., 2014). At any particular instant, each particle has a position and a velocity. The position vector of a particle with respect to the origin of the search space represents a trial solution of the search problem. At the beginning, a population of particles is initialized with random positions marked by vectors and random velocities. The population of such particles is called a swarm. Each particle has two state variables, its current position and its current velocity (Mohais et al., 2005). It is also equipped with a small memory comprising its previous best position (one yielding the highest value of the fitness function found so far), personal best experience and the best all particles, the best position found so far in the neighborhood of the particle. Once the particles are all initialized, an iterative optimization process begins, where the positions and velocities of all the particles are altered by recursive equations. After having calculated the velocities and position for the next step, the iteration of the algorithm is completed. Typically, this process is iterated for a certain number of time steps, or until some acceptable solution has been found (Das et al., 2008).

When differential evolution and particle swarm optimization are used with a true random number generator, the statistical quality of the random number is guaranteed. In real practice, a true random number generator is rarely available. In such a situation, different pseudo-random number generators are used. Different pseudo-random number generators have different statistical qualities of the numbers generated. In this research, the influence of the two different quality pseudo-random number generators is investigated in a hybrid implementation of differential evolution and particle swarm optimization.

The rest of this paper is organized as follows: Section 1 introduces the problem with the statistical quality in pseudo-random numbers, which are used in population-based optimization; Section 2 describes the way in which pseudo-random number generators are compared; Section 3 reveals some practical experiments and related results; and Section 4 concludes with some suggestions for further work.

2. Practical Uses Pseudo-Random Number Generators

Particular pseudo-random number generators are widely used in practice. The most popular of them is the Linear Congruent Generator (LCG) (Class Random. Java SE Documentation, 2021). The experimental part in this study is done with Java source code, part of LibreOffice Calc NLP Solver (NLP Solver. LibreOffice Calc, 2021). The standard distribution of Java comes with the LCG (Class Random. Java SE Documentation, 2021) and SecureRandom generators (Class SecureRandom. Java SE Documentation, 2021). For this reason, a comparison is made between the two.

The mathematical formula of LCG is pretty simple:

$$X(m+1) = (aX(m) + b) \bmod c \qquad (1)$$

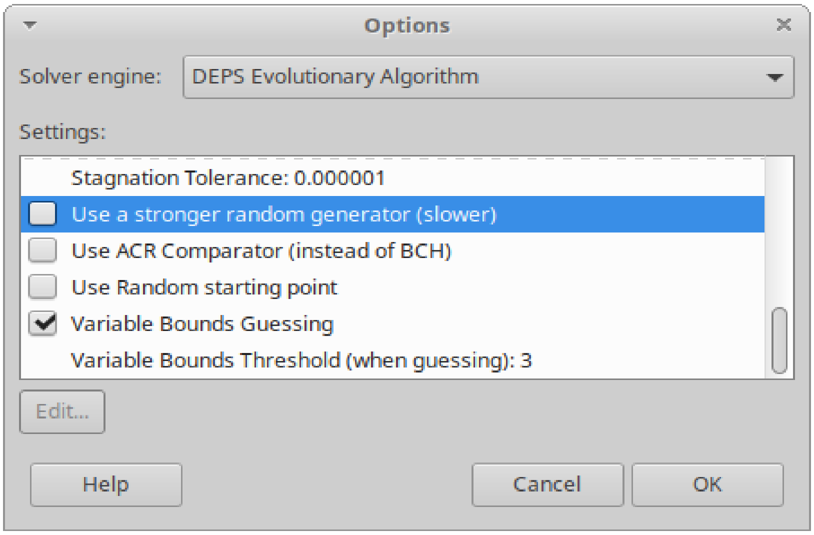
Each subsequent value is functionally dependent on the previous one. It is extremely important that the initial value $X(0)$ is random and as difficult to predict as possible. The different implementations of LCG differ only in the choice of the constants a, b, and c. Due to its simplicity, LCG is very fast. In contrast to the performance is the compromise with the statistical quality of the generated numbers. It is proven that LCG should not be used for anything important.

On the other side, the secure random generator in Java varies in different Java Virtual Machine implementations. It minimally should comply with the statistical random number generator tests specified in FIPS 140-2, Security Requirements for Cryptographic Modules (Class SecureRandom. Java SE Documentation, 2021). It must produce non-deterministic output. When it is possible, entropy is being gathered from sources like /dev/random on various Unix-like operating systems.

The main difference between the two generators is their performance. Tests done on Intel Core i5 2,3 GHz, Single CPU with 2 cores, 8GB RAM, macOS High Sierra 10.13.6, Java SE 11.0.2 machine shows that the SecureRandom is 25 times slower than Random. Performance is the price of getting random numbers with better statistical qualities. The main question that arises from this difference in performance is whether statistically better numbers can compensate for speed and lead to faster converging optimization.

3. Training ANN with LibreOffice Calc NLP Solver

The actual stable release version of LibreOffice at the moment of current writing is 7.0.6.2. All conducted experiments are performed with a development version of the product (7.3.0.0.alpha0+). The development version of the product is used, as the functionality for random number generators is not yet part of the release version of the product. A separate flag has been added to the Calc's NLP Solver settings dialog so that the LCG and SecureRandom generators can be switched (Fig. 1).


Options

Solver engine: DEPS Evolutionary Algorithm

Settings:

Stagnation Tolerance: 0.000001

☐ Use a stronger random generator (slower)

☐ Use ACR Comparator (instead of BCH)

☐ Use Random starting point

✓ Variable Bounds Guessing

Variable Bounds Threshold (when guessing): 3

Fig. 1. The user interface for switching random number generators

As a benchmark model, training of feed-forward artificial neural network is used. The artificial neural network is trained to do forecasting of financial time series (Tomov, 2021). Dataset used in publicly available at (Balabanov, 2021). The artificial neural network has three layers (input, hidden, and output) with a total number of weights of 555 (Experiment-26-Mar-2021-2.ods).

Experiments show that LCG gives only slightly weaker results than the *SecureRandom* generator. LCG achieves a target value of 0.021592 in 5340 seconds (Fig. 2), while *SecureRandom* achieves a target value of 0.0212213 in 5280 seconds (Fig.3). This difference in the third decimal place shows that both generators give practically good results.

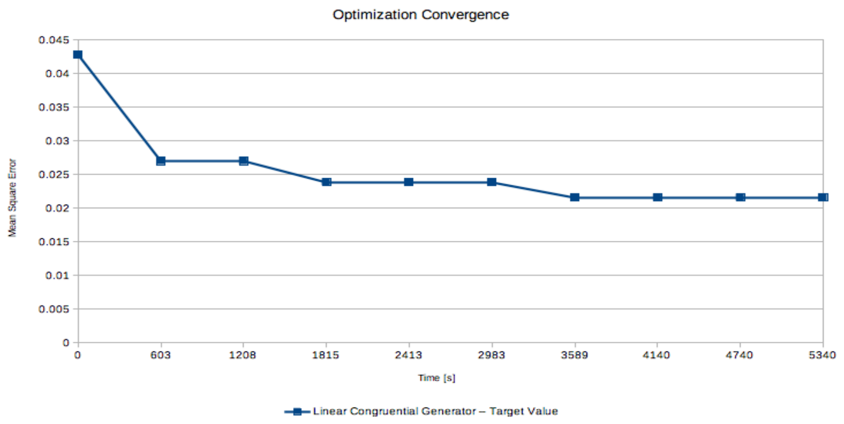


Fig. 2. Optimization convergence with Linear Congruential Generator

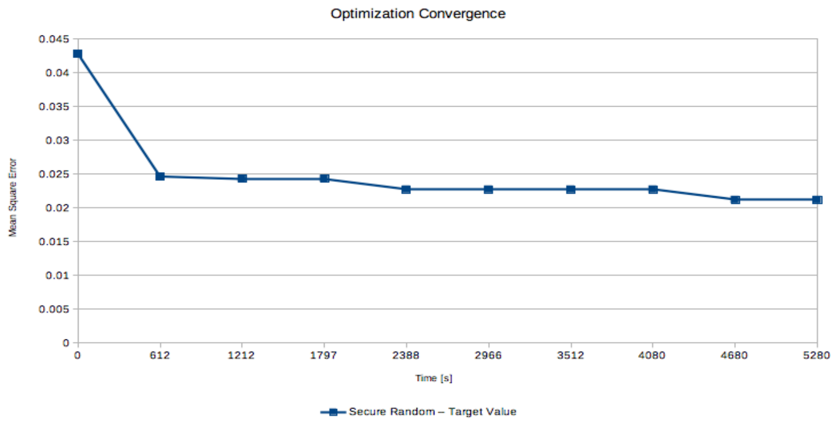


Fig. 3. Optimization convergence with SecureRandom generator

These results clearly show that the use of a generator with statistically higher quality numbers is fully justified in LibreOffice Calc for the NLP Solver module. Such use does not affect the performance. In some cases, even improves the convergence of optimization.

4. Conclusion

This research examines the influence of the quality that pseudo-random numbers have on heuristic optimization algorithms. A hybrid implementation of Differential Evolution and Particle Swarm Optimization is chosen for the experiments. The source code is written in Java and is part of the NLP Solver module of LibreOffice Calc. Newly added random number generator selection functionality allows experiments to be performed. The conclusion from the performed tests is that better quality generators do not slow down the performance, but in some models, they lead to faster convergence of the optimization process.

Possibilities for more thorough testing of the various generators can be mentioned as guidelines for future research. It is essential to monitor the behavior of the SecureRandom generator in different operating systems and different hardware. It is also important to study different optimization models with different levels of complexity and nonlinearity.

Acknowledgment

This research is funded by Velbazhd Software LLC and it is partially supported by the Bulgarian Ministry of Education and Science (contract D01–205/23.11.2018) under the National Scientific Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)", approved by DCM # 577/17.08.2018.

References

1. Balabanov, T.: LibreOffice Calc Three Layers Perceptron Builder. https://github.com/VelbazhdSoftwareLLC/LibreOffice-Calc-Three-Layers-Perceptron-Builder, last accessed 2021/08/04.
2. Borissova, D., Dimitrova, Z., Dimitrov, V.: How to support teams to be remote and productive: Group decision-making for distance collaboration software tools. Information and Security. Digital Transformation, Cyber Security and Resilience, vol. 46, pp. 36-52, 2020. https://doi.org/10.11610/isij.4603.
3. Class Random. Java SE Documentation, Oracle Corporation, https://docs.oracle.com/javase/8/docs/api/java/util/Random.html, last accessed 2021/08/02.
4. Class SecureRandom. Java SE Documentation, Oracle Corporation, https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html, last accessed 2021/08/02.
5. Das, S., Abraham, A., Konar, A.: Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. Advances of Computational Intelligence in Industrial Systems, Studies in Computational Intelligence, vol. 116, 1-38, (2008). https://doi.org/10.1007/978-3-540-78297-1_1.
6. Das, S., Mullick, S., Suganthan, P.: Recent advances in differential evolution – An updated survey. Swarm and Evolutionary Computation, vol. 27, 1-30 (2016). https://doi.org/10.1016/j.swevo.2016.01.004.
7. Dillen, W., Lombaert, G., Schevenels, M.: Performance assessment of metaheuristic algorithms for structural optimization taking into account the influence of algorithmic control parameters. Frontiers in Built Environment, (2021). https://doi.org/10.3389/fbuil.2021.618851.

8. Garvanov, I., Garvanova, M., Borissova, D., Vasovic, B., Kanev, D.: Towards IoT-Based Transport Development in Smart Cities: Safety and Security Aspects. Business Modeling and Software Design, Lecture Notes in Business Information Processing. Springer, Cham, vol. 422, 392-398 (2021) https://doi.org/10.1007/978-3-030-79976-2_27.
9. Guliashki, V., Toshev, H., Korsemov, C.: Survey of evolutionary algorithms used in multiobjective optimization. Problems of Engineering Cybernetics and Robotics, vol. 60, 42-54 (2009).
10. Huimin, L., Shuwen, X., Yanlong, Y., Chenwei, L.: Differential evolution particle swarm optimization algorithm based on good point set for computing Nash equilibrium of finite noncooperative game. AIMS Mathematics, vol. 6, no. 2, 1309-1323 (2021). https://doi.org/10.3934/math.2021081.
11. Koeune, F.: Pseudo-random number generator. Encyclopedia of Cryptography and Security, Springer, Boston, MA, (2005), https://doi.org/10.1007/0-387-23483-7_330.
12. Machica, J., Bruno, O.: Improving the pseudo-randomness properties of chaotic maps using deep-zoom. Chaos vol. 27, 053116 (2017). https://doi.org/10.1063/1.4983836.
13. Mirjalili, S., Lewis, A., Sadiq, A.: Autonomous Particles Groups for Particle Swarm Optimization. Arabian Journal for Science and Engineering, vol. 39, 4683-4697 (2014). https://doi.org/10.1007/s13369-014-1156-x.
14. Mohais, A., Mendes, R., Ward, C., Posthoff, C.: Neighborhood Re-structuring in Particle Swarm Optimization. Advances in Artificial Intelligence, LNCS, Springer, Berlin, Heidelberg, vol. 3809, 776-785 (2005). https://doi.org/10.1007/11589990_80.
15. NLP Solver. LibreOffice Calc, https://github.com/LibreOffice/core/tree/master/nlpsolver, last accessed 2021/08/02.
16. Tanabe, R., Fukunaga, A.: Reevaluating Exponential Crossover in Differential Evolution. Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Springer, Cham, vol. 8672, 201-210 (2014). https://doi.org/10.1007/978-3-319-10762-2_20.
17. Tomov, P.: Multilayer perceptron fast prototyping with differential evolution and particle swarm optimization in LibreOffice Calc. Problems of Engineering Cybernetics and Robotics 75, 5-14 (2021). https://doi.org/10.7546/PECR.75.21.02.
18. Trifonov, R., Manolov, S., Yoshinov, R., Tsochev, G., Nedev, S., Pavlova, G.: Operational cyber-threat intelligence supported by artificial intelligence methods. In: Proc. of International Conference on Information Technologies, 20-21 (2018).
19. Yang, X.: Chapter 8 - Particle Swarm Optimization. Nature-Inspired Optimization Algorithms (Second Edition), Academic Press, 111-121 (2021). https://doi.org/10.1016/B978-0-12-821986-7.00015-9.