Performance Study of Bat Algorithm Running on Embedded Hardware

Iztok Jr. Fister [1], Grega Vrbancic[1], Tomaz Hozjan[2], Iztok Fister[1], Vili Podgorelec[1]
[1]*Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroska cesta 46, SI-2000 Maribor, Slovenia*
[2]*Faculty of Civil and Geodetic Engineering, University of Ljubljana, Jamova 2, SI-1115 Ljubljana, Slovenia*
iztok.fister1@um.si

***Abstract*—Bat algorithm belongs to a class of swarm intelligence algorithms. Comparing to the other stochastic nature-inspired population-based algorithms, this has always been considered as computationally inexpensive. Due to its simplicity and effectiveness, it is very popular in scientific community by solving various optimization problems. However, not enough devotion has been performed on the behaviour of this algorithm running on embedded hardware. In this paper, we look for an answer about the performance of bat algorithm running on three different Raspberry Pi devices of limited performances. In line with this, we conduct a series of experiments by solving the benchmark suite consisting of 10 functions. Because of achieving the similar quality of solutions on all observed hardware, thus, the execution times have been measured.**

***Index Terms*—Computational intelligence; Embedded software; Evolutionary computation.**

I. Introduction

Almost every day, the majority of human beings are faced with solving of many optimization problems unconsciously [1]. For instance, when we deal with money, resources or when we travel around, we always search for those options bringing us the highest benefit. In the real world, we can easily make decision in order to find the best solution, especially when we have a lot of information about the particular problem. Simultaneously to this analogy, scientists have been developing various algorithms for solving optimization problems that are based on mathematics, physics, chemistry, or biological principles. Especially, stochastic population-based nature-inspired algorithms are one of the more interesting algorithms that solve optimization problems by mimicking natural (biological) systems. Recently, there exists a lot of nature-inspired algorithms that are brought together under the umbrella of computational intelligence, CI [2]. For instance, Firefly algorithm, Bat algorithm, Particle Swarm Optimization, etc. are all the algorithms consisting of a set of individuals (population) and governing by variation operators (e.g. mutation, crossover, selection).

Time complexity of the nature-inspired optimization algorithms is usually very high [3]. For that reason, we usually have to run implementations of nature-inspired algorithms on very special hardware. This special hardware consists of many CPU cores that is equipped with big amount of RAM. In line with this, some scientists use rather other platforms for running their algorithms, e.g. grids or clouds. Nowadays, it is very popular to run algorithms on graphical processing units (GPUs) [4] or using FPGA [5], [6]. On the other hand, some authors prefer also multi-threaded techniques [7].

In contrary, some applications limit us from using computationally powerful devices [8]. In current era, we start to embed many Swarm Intelligence (SI) based algorithms to the hardware (e.g., swarm robotics [9], [10]), where very small single-board computers serve as agents in a swarm [11]. The agents in robotic swarm communicate and solve problems together. Here, many additional challenges have been encountered, i.e.:

– How to implement CI algorithms for embedded architectures?
– What are the main pitfalls in implementing CI algorithms for such devices?
– Are there any specific programming languages intended for these applications?
– Which CI algorithm is the most appropriate?
– How to communicate with other agents in environment?

The Raspberry Pi is considered as a very small single-board computer developed by Raspberry Pi Foundation. Initially, it was developed for the promotion of teaching the computer science in the developing countries. However, this community quickly found its efficiency and uses it for various tasks, e.g. web servers, smart home applications, etc.

Inspired by our previous paper [12], where we tested the implementation of bat algorithm on cheaper smartphones, this paper goes a step further. Study in [12] revealed that implementation of bat algorithm on smartphones achieves almost near real-time performance when optimizing small-scale problems. As a matter of fact, the modern even cheaper smartphones are equipped with powerful processor as well as abundance of RAM. This study showed that the execution time of the program on smartphone is approximately 10 times longer than on the PC. Due to their weak robustness, we cannot apply some special swarm robotics applications for running on smartphones [10]. For that reason, we examine the Raspberry Pi devices and

Manuscript received 18 December, 2018; accepted 30 March, 2019.
The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

measure their performance when running bat algorithm (BA) [13]. The results were then compared with the implementations on smartphone and personal computer. Indeed, the purpose of the paper was to obtain as much as possible information about performances of the SI-based algorithms on the Raspberry Pi hardware in order to establish its adequacy for application in swarm robotics [14].

Organization of this paper is as follows: Section II describes basics of the BA. Implementation of the algorithm is a subject of Section III. In Section IV, experiments and results are discussed. The paper concludes with Section V, where directions for the further work are also outlined.

II. Bat Algorithm

Bat algorithm (BA) is a stochastic population-based nature-inspired algorithm. The roots of BA goes back to the 2010 when Yang initially presented this algorithm in [13]. As the majority of the nature-inspired algorithms, BA is also inspired by biological/physical phenomenon, more precisely by echolocation of micro-bats. In BA, each bat is represented with a velocity $\upsilon_i^{(t)}$ and a location $x_i^{(t)}$, in *D*-dimensional search at iteration *t*. Based on the original paper by Yang [13], the mathematical equations for updating the locations $x_i^{(t)}$ and velocities $\upsilon_i^{(t)}$ can be written as:

$$f_i = f_{\min} + \left(f_{\max} - f_{\min}\right)\beta, \tag{1}$$

$$\upsilon_i^t = \upsilon_i^{t-1} + \left(x_i^{t-1} - x_*\right) f_i, \tag{2}$$

$$x_i^t = x_i^{t-1} + \upsilon_i^t, \tag{3}$$

where $\beta \in [0,1]$ is a random number drawn from a uniform distribution. In addition, the loudness $A_i^{(t)}$ and pulse emission rates $r_i^{(t)}$ can be varied during the iterations. For the simplicity, we can use the following equations for varying the loudness and pulse emission rates:

$$A_i^{(t+1)} = \alpha A_i^t, \tag{4}$$

$$r_i^{t+1} = r_i^0 \left[1 - \exp(-\gamma t)\right], \tag{5}$$

where $0 < \alpha < 1$ and $\gamma > 0$ are constants. Algorithm 1 depicts basic variant of BA, while Table I outlines the main components of BA.

Algorithm 1: Canonical Bat Algorithm

```
Input: Bat population
Output: The best solution x_best and its corresponding value.
initbat();
eval = evaluate_the_new_population;
fmin = find_the_best_solution(xbest);{initialization}
while termination_condition_not_meet do
for I = 1 to Np do
     y= generate_new_solution(xi);
     if rand(0,1) > ri then
     y= improve_the_best_solution(xbest)
   end if {local search step}
   fnew = evaluate_the_new_solution(y);
   eval = eval + 1;
   if fnew ≤ fi and N(0,1) < Ai then
     xi = y; fi = fnew;
   end if {save the best solution conditionally}
   fmin = find_the_best_solution(xbest);
end for
end while
```

III. Implementation on Raspberry Pi Device

The Raspberry Pi is a fully featured credit-card sized single-board computer, which is capable of performing similar tasks like a standard desktop PC. Due to a great success of the first revisions of the Raspberry Pi model A and model B, some new models and new revisions of existing models have been emerged recently (e.g., model A+, model B+, etc.). Regardless of the improvements and hardware upgrades of those models, the general design remains the same across all models. In general, the Raspberry Pi board contains a central and graphics processing units, Random-Access Memory (RAM) chip, and various interfaces and connectors for external devices. Some of the interfaces and connectors are necessary for each implementation, while the others are optional. Actually, all of the Raspberry Pi models base on some version of Broadcom system on a chip (SoC) and implement the ARM architecture.

TABLE I. MAIN COMPONENTS OF THE BAT ALGORITHM.

Component	Bat algorithm
initialization	Initialization of the parameters of the algorithm and initial population is conducted, while evaluation determines the best solution xbest in the population.
generate_new_solution	Virtual bats are moved in the search space according to updating rules of the Bat Algorithm.
local_search_step	The best solution is being improved using random walk.
evaluate_the_new_solution	The evaluation of the new solution is achieved.
save_the_best_solution_conditionaly	Conditional archiving of the best solution takes place.
find_the_best_solution	The current best solution is updated.

In our experiment, we used three different Raspberry Pi models. The Raspberry Pi 3 Model B+ is the most powerful one with Broadcom BCM2837B0 quad-core processor running at 1.4 GHz, while the least powerful is the first revision Raspberry Pi Model B with Broadcom BCM2835 single-core processor running at 700 Mhz. The Raspberry Pi model A+ is sharing the same SoC as the model B+ while the amount of RAM is halved to 512 MB. A more detailed specification comparison is provided in Table II.

(a) (b) (c)

Fig. 1. Presented as: a) is Raspberry Pi Model B revision 1; as b) Raspberry Pi 3 Model B+; and as c) Raspberry Pi 3 Model A+.

TABLE II. DEVICES USED IN EXPERIMENT.

Device	CPU	Cores	Freq	RAM	OS
Raspberry Pi Model B rev. 1	Broadcom BCM2835	Single-core	700 MHz	256 MB	Raspbian OS
Raspberry Pi 3 Model B+	Broadcom BCM2837 B0	Quad-core	1.4 GHz	1 GB	Raspbian OS
Raspberry Pi 3 Model A+	Broadcom BCM2837 B0	Quad-core	1.4 GHz	512 MB	Raspbian OS
Smart-phone	Qualcomm MSM8212	Quad-core	1.2 GHz	1 GB	Android
Personal computer	Intel(R) Xeon(R) E3-1240	Octa-core	3.5 GHz	16.344 GB	Ubuntu 16.04.5 LTS

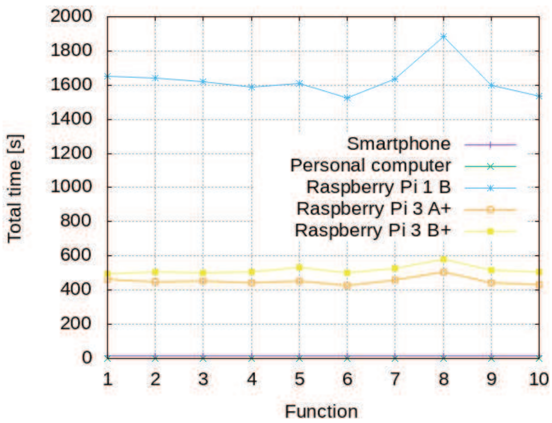

Fig. 2. Comparison of the total time between various platforms.

IV. Experiments and Results

The purpose of our experimental work was to show that the used Raspberry Pi platforms are appropriate not only for running optimization algorithms, but also for embedding the evolutionary and swarm intelligence algorithms into hardware. In the experiments, we implemented the BA algorithm in C++ programming language that was rewritten from the main implementation of the BA in Matlab programming language [13]. The same algorithm was run on all five different platforms: Smartphone, PC, and three different Raspberry Pi's.

Parameter settings of the BA on all platforms were the same, as follows: D = 10, NP = 10, MAX_RUNS = 25, MAX_GEN = 1,000, A = 0.5, r = 0.5, $Q_{\min}$ = 0.0, $Q_{\min}$ = 1.0. Let us mention that 10 well known benchmark functions [15] are used in our study, i.e., Griewank, Rastrigin, Rosenbrock, Ackley, Schwefel, De-Jong, Easom, Michalewicz, Xin-She, and Zakharov. The quality of results was measured according to the time needed for execution of all the number of runs. The lower the time complexity, the better the observed algorithm. Thus, the minimum, maximum, mean, median, and standard deviation of time values were taken into consideration.

As can be seen from the tables, the BA running on the PC exposes the best results in the sense of the minimum time complexity. The same algorithm is 10 times slower, when it run on the smartphone. Interestingly, the time complexities of the BA algorithm running the Raspberry Pi devices are much higher comparing with the both already mentioned implementations. Actually, the time complexity of the BA algorithm running the Raspberry Pi Model A+ and B+ are around 3.10^2 times higher than the time complexity on the PC and around 10^3 times higher than the time complexity on the Raspberry Pi Model B rev. 1. Interestingly, the time complexity of this algorithm running on the Raspberry Pi Model B+ is lower than by its counterpart the Raspberry Pi Model A+.

The results of experiments are illustrated in Table III–Table V, where five statistical measures obtained by the BA running on various hardware devices are presented according to the particular function. Additionally, the line displaying the average values of each statistical measure are included into the tables.

The same finding can be derived also from Fig. 2 that represents the comparison of total times needed for one run of the BA on the particular device.

As can be seen from the figure, the time complexity of the BA running on the PC and smartphone is much lower than those running on the Raspberry Pi devices. On the other hand, the running the BA on these devices showed that they are capable of running the stochastic nature-inspired population-based algorithms as well.

V. Conclusions

Nowadays, stochastic nature-inspired population-based algorithms running on disembodied computers (e.g., PCs) reached their matured phase. The next challenge for these algorithms is to embed them into the hardware, where these act as agents devoted to solve particular problems.

In this study, we tried to solve the classical global optimization problems on specific Raspberry Pi devices that serve as hardware platforms for embedding the evolutionary and SI-based algorithms. In line with this, a benchmark consisting of 10 test functions were taken into consideration. The experiments showed that the results of the same quality as on the regular PC can be obtained also on these devices, but in slightly longer time. However, this finding courage us to speculate that the further development of these platforms would ensure the stable infrastructure for using in evolutionary and swarm robotics.

TABLE III. RESULTS OF BA IMPLEMENTATION ON SMARTPHONE AND PC.

Function	Smartphone						Personal computer				
	t_{best}	t_{worst}	t_{mean}	t_{median}	t_{stdev}		t_{best}	t_{worst}	t_{mean}	t_{median}	t_{stdev}
Griewank	0.4433	0.5028	0.4633	0.4611	0.0132		0.0472	0.0567	0.0478	0.0472	0.0017
Rastrigin	0.4288	0.4603	0.4393	0.4367	0.0072		0.0482	0.0614	0.0491	0.0485	0.0023
Rosenbrock	0.4670	0.5094	0.4796	0.4796	0.0094		0.0453	0.0578	0.0460	0.0456	0.0022
Ackley	0.4176	0.4593	0.4308	0.4283	0.0102		0.0479	0.0621	0.049	0.0488	0.0024
Schwefel	0.4336	0.4824	0.4461	0.4459	0.0094		0.0455	0.0601	0.0467	0.0463	0.0024
DeJong	0.4225	0.4616	0.432	0.4307	0.0074		0.0438	0.0571	0.0445	0.0439	0.0023
Easom	0.5153	0.5639	0.525	0.5238	0.0086		0.0487	0.0616	0.0499	0.0498	0.0023
Michalewicz	0.467	0.4998	0.4832	0.4851	0.0079		0.0412	0.0563	0.0424	0.042	0.0026
Xin-She	0.4229	0.4577	0.4366	0.436	0.0094		0.0462	0.0586	0.0467	0.0462	0.0022
Zakharov	0.4321	0.4684	0.4452	0.4438	0.0088		0.0452	0.0587	0.046	0.0457	0.0023
Average	0.0459	0.059	0.0468	0.0464	0.0023		0.445	0.4866	0.4581	0.4571	0.0092

TABLE IV. RESULTS OF BA IMPLEMENTATION ON RASPBERRY PI B AND B+.

Function	**Raspberry Pi Model B rev. 1**						**Raspberry Pi Model B+ rev 3**				
	t_{best}	t_{worst}	t_{mean}	t_{median}	t_{stdev}		t_{best}	t_{worst}	t_{mean}	t_{median}	t_{stdev}
Griewank	52.7862	54.2845	53.3105	53.2109	0.2832		14.9474	17.0021	15.9906	16.0928	0.7456
Rastrigin	52.1841	53.8043	52.9292	52.8962	0.3061		14.6756	17.0661	16.381	16.6063	0.6498
Rosenbrock	51.7238	53.1704	52.2313	52.2185	0.2523		14.5413	17.1585	16.2178	16.5552	0.8656
Ackley	50.6798	52.0978	51.29	51.3175	0.2871		14.2913	16.8393	16.4064	16.6696	0.6057
Schwefel	51.291	52.9672	51.9528	51.9248	0.2993		16.4396	17.3308	17.1512	17.1645	0.1509
DeJong	48.5576	50.1524	49.2024	49.1363	0.2843		15.869	16.3194	16.1085	16.1053	0.0968
Easom	52.3973	53.5768	52.8189	52.7569	0.2813		14.9516	17.4873	17.0063	17.2707	0.6349
Michalewicz	60.0257	61.9097	60.7138	60.5764	0.4197		16.4578	19.2288	18.7262	18.99	0.6863
Xin-She	51.0384	52.1921	51.4968	51.5139	0.2668		14.8765	16.8788	16.6348	16.7006	0.3502
Zakharov	49.2344	49.9676	49.5778	49.5555	0.1873		14.2843	16.5936	16.2592	16.4686	0.5486
Average	51.9918	53.4123	52.5524	52.5107	0.2867		15.1334	17.1905	16.6882	16.8624	0.5334

TABLE V. RESULTS OF BA IMPLEMENTATION ON RASPBERRY PI A+.

Function	**Raspberry Pi Model A+ rev 3**				
	t_{best}	t_{worst}	t_{mean}	t_{median}	t_{stdev}
Griewank	14.7891	15.2181	14.9349	14.9441	0.0837
Rastrigin	14.3215	14.6564	14.4855	14.4945	0.0801
Rosenbrock	14.4288	14.6636	14.5575	14.5474	0.0658
Ackley	14.1392	14.4392	14.28	14.2724	0.0713
Schwefel	14.5226	14.7979	14.6312	14.626	0.0632
DeJong	13.6351	13.9708	13.784	13.7693	0.0715
Easom	14.7127	14.9043	14.7818	14.7663	0.056
Michalewicz	16.1075	16.4426	16.2655	16.2599	0.0653
Xin-She	14.1612	14.4274	14.3157	14.3249	0.0703
Zakharov	13.8854	14.1794	14.0431	14.0354	0.059
Average	14.4703	14.77	14.6079	14.604	0.0686

REFERENCES

[1] C. R. Reeves, *Modern heuristic techniques for combinatorial problems. Advanced topics in computer science*. Mc Graw-Hill, 1995.

[2] A. P. Engelbrecht, *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

[3] M. W. Krentel, "The complexity of optimization problems", *Journal of computer and system sciences*, vol. 36, no. 3, pp. 490–509, 1988. DOI: 10.1016/0022-0000(88)90039-6.

[4] A. V. Husselmann, K. A. Hawick, "Parallel parametric optimization with firefly algorithms on graphical processing units", in *Proc. Int. Conf. on Genetic and Evolutionary Methods (GEM 2012)*, 2012, pp. 77–83.

[5] M. El-Shafei, I. Ahmad, M. Gh Alfailakawi, "Implementation of harmony search on embedded platform", *Microprocessors and Microsystems*, vol. 45, pp. 187–197, 2016. DOI: 10.1016/j.micpro.2016.05.003.

[6] M. El-Shafei, I. Ahmad, M. Gh Alfailakawi, "Hardware accelerator for solving 0–1 knapsack problems using binaryharmony search", *International Journal of Parallel, Emergent and Distributed Systems*, vol. 33, no. 1, pp. 87–102, 2018. DOI: 10.1080/17445760.2017.1324025.

[7] D. Polap, K. Kesik, M. Wozniak, R. Damasevicius, "Parallel technique for the metaheuristic algorithms usingdevoted local search and manipulating the solutions space", *Applied Sciences*, vol. 8, no. 2, p. 25, 2018. DOI: 10.3390/app8020293.

[8] E. Mininno, F. Neri, F. Cupertino, D. Naso, "Compact differential evolution", *IEEE Trans. Evolutionary Computation*, vol. 15, no. 1, pp. 32–54, 2011. DOI: 10.1109/TEVC.2010.2058120.

[9] N. Moustafa, A. Galvez, A. Iglesias, "A general-purpose hardware robotic platform for swarm robotics", *In International Symposium on Intelligent and Distributed Computing*, pp. 259–271, 2018. DOI: 10.1007/978-3-319-99626-4_23.

[10] P. Suarez *et al.*, "Bat algorithm swarm robotics approach for dual non-cooperative search with self-centered mode", *Intelligent Data Engineering and Automated Learning (IDEAL 2018)*, 2018, pp. 201–209. DOI: 10.1007/978-3-030-03496-2_23.

[11] P. Suarez, A. Iglesias, A. Galvez, "Make robots bebats: specializing robotic swarms to the bat algorithm", *Swarm and Evolutionary Computation*, to be published. DOI: 10.1016/j.swevo.2018.01.005

[12] I. Jr. Fister, S. Deb, I. Fister, "Near real-time performance of population-based nature-inspired algorithms on cheaper and older smartphones", *Int. conf. soft computing and machine intelligence (ISCMI 2018)*, Nairobi, Kenya, Kenya, pp. 12-16, 2018. DOI: 10.1109/ISCMI.2018.8703236.

[13] X. S. Yang, "A new metaheuristic bat-inspired algorithm", *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pp. 65–74, 2010. DOI: 10.1007/978-3-642-12538-6_6.

[14] A. E. Eiben, J. Smith, "From evolutionary computation to the evolution of things", *Nature*, vol. 521, pp. 476–482, 2015. DOI: 10.1038/nature14544.

[15] M. Jamil, X.-S. Yang, "A literature survey of benchmark functions for global optimization problems", *International Journal of Mathematical Modelling and Numerical Optimisation (IJMMNO)*, vol. 4, no. 2, 2013. DOI: 10.1504/IJMMNO.2013.055204.