Running Genetic Algorithms in the Edge: A First Analysis

José Á. Morell and Enrique Alba

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Andalucía Tech, Spain *
{jamorell, eat}@lcc.uma.es

Abstract. Nowadays, the volume of data produced by different kinds of devices is continuously growing, making even more difficult to solve the many optimization problems that impact directly on our living quality. For instance, Cisco projected that by 2019 the volume of data will reach 507.5 zettabytes per year, and the cloud traffic will quadruple. This is not sustainable in the long term, so it is a need to move part of the intelligence from the cloud to a highly decentralized computing model. Considering this, we propose a ubiquitous intelligent system which is composed by different kinds of endpoint devices such as smartphones, tablets, routers, wearables, and any other CPU powered device. We want to use this to solve tasks useful for smart cities. In this paper, we analyze if these devices are suitable for this purpose and how we have to adapt the optimization algorithms to be efficient using heterogeneous hardware. To do this, we perform a set of experiments in which we measure the speed, memory usage, and battery consumption of these devices for a set of binary and combinatorial problems. Our conclusions reveal the strong and weak features of each device to run future algorihms in the border of the cyber-physical system.

Keywords: Edge Computing, Fog Computing, Evolutionary Algorithms, Genetic Algorithms, Metaheuristics, Smartphone, Tablet, Ubiquitous AI

1 Introduction

At present, cloud computing looks like a mature and stable domain. Processing all the data in the Cloud have been effective until now. However, when we analyze the near future in terms of volume of data produced, we find some issues for which we are not yet ready. However, we can say that cloud computing and even big data as we knew them until now are not sustainable in the long term. To understand the problem, we must realize the large amount of data that is created by new devices, not only laptops and smartphones, but also tablets, wearable devices, home and city sensors, and so on. According to Cisco, "The Internet of Everything", which means all of the people and things connected to Internet, will generate 507.5 zettabytes of data by 2019 [1]. That is a large amount of data that needs to be analyzed. Big Data analysis is currently performed in cloud or

* This research has been partially funded by the Spanish MINECO and FEDER projects TIN2014-57341-R (http://moveon.lcc.uma.es), TIN2016-81766-REDT (http://cirti.es), TIN2017-88213-R (http://6city.lcc.uma.es), the Ministry of Education of Spain (FPU16/02595), and Universidad de Málaga.

enterprise data centers. Consequently, it is necessary to find new approaches to solve problems at the edge without sending all the information to the cloud.

Several new approaches are now emerging to solve these problems. One of them is called **edge computing** (aka fog computing) [2], a paradigm where computing tasks move closer to the source of data. Currently, sensors are collecting so much data that it is necessary to perform some data analysis at the edge instead of sending all data to the cloud. To achieve this, edge computing uses endpoint processing capacity in a distributed way. Peter Levine, a partner at the Silicon Valley venture capital firm Andreessen Horowitz, has recently talked about edge computing as the next multibillion-dollar tech market [3]. He said that edge computing would overtake the cloud and that the cloud would still store important data, but as devices become more sophisticated, all the data curation and learning would take place at the edge. Furthermore, both research and industry are taking steps in this direction. For instance, some companies such as HTC and Samsung are developing apps to allow people to contribute with the unused computational power of their smartphones in mobile grids (HTC Power to Give[1] and Samsung Power Sleep[2]).

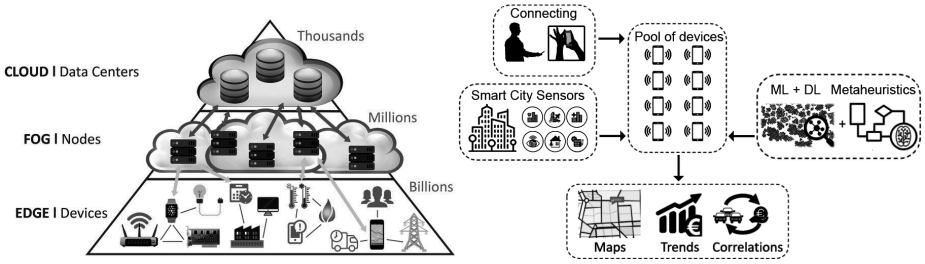

Fig. 1. Ubiquitous Artificial Intelligence at the Edge.

Considering this, our main aim is to design a ubiquitous intelligent system (Fig. 1) for solving complex problems of smart cities using a highly decentralized computing model. However, this system will be composed of a wide variety of heterogeneous hardware. We have to deal with asynchronism, fault tolerance and the devices that join and leave the system. Therefore, we must analyze this hardware and test it to solve problems locally, before doing it in a distributed way. For this purpose, we perform a set of experiments in which we measure the speed, memory usage, and battery consumption of these devices solving a set of well-known optimization problems using **Genetic Algorithms** (GAs). This research is an extension of a previous work [4].

[1] https://www.htc.com/es/go/power-to-give/

[2] https://news.samsung.com/global/power-sleep-app-lets-you-be-a-part-of-an-advanced-scientific-research

In summary, we will answer the following set of worth research questions:

- RQ1: Are these devices suitable to solve complex problems using smart algorithms?
- RQ2: What are the differences in resources usage between the different portable devices?
- RQ3: How can we develop efficient algorithms for so limited memory devices?

This article is organized as follows: Section II describes the problems chosen for experimentation. Section III shows details about the design of our algorithm. Section IV analyzes and discusses the experiments performed. Finally, Section V draws the main conclusions and the future work research.

2 Description of the Problems

In this section, we present the set of problems chosen for the experimentation, including different sizes, different types of underlying NP-Hard problems, different constraints, varied fitness functions, and considered the areas of research they represent. As a result, the benchmark contains binary and combinatorial problems with many different interesting features in optimization, such as epistasis, multimodality, and deceptiveness. The binary problems chosen are the Massively Multimodal Deceptive Problem (MMDP) [5], the Error Correcting Code Design (ECC) [6], the Minimum Tardy Task Problem (MTTP) [7], and finally, the One-Max problem [8] to ease comparisons to other domains and algorithms. Also, we have chosen a well-known combinatorial problem, the Capacitated Vehicle Routing Problem (CVRP) [9] that can scale and represent problems in smart mobility in cities. Let us now proceed to explain each one briefly.

MMDP is a problem in which bipolar deceptive functions with two global optima and with a number of deceptive optima are designed. In MMDP each subproblem s_i contributes to the total fitness value according to the number of ones it has. The number of local optma is quite large (n^k), while there are only 2^k global solutions. The degree of multimodality is regulated by the k parameter.

$$f_{MMDP(s)} = \sum_{i=1}^{k} fitness_{s_i} \tag{1}$$

ECC consists of assigning codewords to an alphabet that minimizes the length of transmitted messages and that provides maximal correction of single uncorrelated bit errors, when the messages are transmitted over noisy channels. A code can be formally represented by a three-tuple (n, M, d), where n is the length of each codeword, M is the number of codewords and d is the minimum Hamming distance between any pair of codewords. An optimal code consists in constructing M binary codewords, each of length n, such that d, the minimum Hamming distance between each codeword and all other codewords, is maximized.

$$f_{ECC} = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} \frac{1}{d_{ij}^2}} \tag{2}$$

MTTP is a NP-hard task-scheduling problem wherein each task i from the set of tasks $T = 1, 2, ..., n$ has a length l_i (the time it takes for its execution), a deadline d_i (before which a task must be scheduled), and a weight w_i. The weight is a penalty that has to be added to the objective function in the event that the task remains unscheduled. The lengths, weights and deadlines of tasks are all positive integers. Scheduling the tasks of a subset S of T is to find the starting time of each task in S, such as at most one task at time is performed and such that each task finishes before its deadline. The objective function for this problem is to minimize the sum of the weights of the unscheduled tasks.

$$f_{MTTP}(\boldsymbol{x}) = \sum_{i \in T-S} w_i \tag{3}$$

OneMax (or *BitCounting*) is a simple problem consisting in maximizing the number of ones of a bitstring. Formally, this problem can be described as finding an string $\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$, with $x_i \in \{0, 1\}$, that maximizes the following equation:

$$f_{OneMax}(\boldsymbol{x}) = \sum_{i=1}^{N} x_i \tag{4}$$

CVRP is a problem in which a fixed fleet of delivery vehicles of the same capacity must service known customer demands for a single commodity from a common depot at minimum total transit costs. CVRP is defined as an undirected graph $G = (V, E)$ where $V = \{v_0, v_1, ..., v_n\}$ is a vertex set and $V = \{(v_i, v_j) | v_i, v_j \in V, i < j\}$ is an edge set. The depot is represented by v_0 and it is from where m identical vehicles of capacity Q must serve all the customers, represented by the set of n vertices $\{v_1, ..., v_n\}$. A solution for the CVRP is a partition $R_1, R_2, ..., R_m$ of V representing the routes of the vehicles. The cost of the problem solution is the sum of the costs of its routes $R_i = \{v_{i0}, v_{i1}, ..., v_{ik+1}\}$ where $v_{ij} \in V$ and $v_{i0} = v_{ik+1} = 0$, where 0 is the depot, satisfying $\sum v_{ij \in R_i}, q_j \leq Q$, which can be presented as:

$$Cost = \sum_{i=1}^{m} Cost(R_i) = \sum_{j=0}^{k} c_{ij+1} \tag{5}$$

where matrix $C = (c_{ij})$ is a non-negative cost between customers v_i and v_j defined on E.

3 Design of our Algorithm

The algorithm used to test the performance of the portable devices is a Genetic Algorithm (GA). A GA is a population-based metaheuristic inspired by the process of natural selection. GAs are used to generate high-quality solutions to optimization and search problems. They are especially useful to solve NP-hard and even NP-complete problems. Exact algorithms are designed in such a way that it is guaranteed that they will find the optimal solution in a finite amount of time. However, when the problem is big, exact methods become useless while metaheuristic algorithms can still find good solutions in a reasonable amount of time.

The canonical GA uses three main types of rules at each step to create the next generation from the current population. First, selection rules select the individuals, called parents, that will be recombined. Second, recombination rules combine the parents to form a child for the next generation. Third, mutation rules apply random changes to the recombined individual to form the final child. Next, we replace the worst individual in the population with the new child. It is like natural selection. We repeat the process again and again until we achieve a good enough solution.

In this research we use a canonical steady state algorithm to solve our problems, adding an additional step in the case of the combinatorial problem (CVRP). In this case, we add a local search operator to the canonical GA due to its efficiency solving CVRP. We have implemented our algorithm similarly to that used by Dorronsoro and Alba [10].

4 Experimentation

For the experimentation we used five different portable devices of a wide variety of performances (Tables 1 and 2). These devices are a laptop, two smartphones, a tablet and a raspberry pi 3 (RP3). Their technical specifications are showed in Table 1. In experimentation we compare their performances solving known hard problems in literature in terms of speed, memory usage and battery consumption.

Table 1. Hardware in the edge: features

	laptop	**RP3**	**mobileA**	**tablet**	**mobileB**
OS	Ubuntu 16.04 LTS (64-bit)	Raspbian 9 (64-bit)	Android 5.1.1 ARMv7 (32-bit)	Android 5.0.1 x86 (32-bit)	Android 6.0 AArch64 (64-bit)
Java VM	1.8.0_161	1.8.0_65	-	-	-
Model	Toshiba Satellite L50-B	Raspberry Pi 3	Zuk Z1	Lenovo TAB S8-50F	LeMobile LEX653
Memory	7893 MB	745 MB	2871 MB	1870 MB	3759 MB
	1600 MHz	900 MHz	933 MHz	778 MHz	800 MHz
Factory Battery	14.8 V 2800 mAh	3.7 V 3800 mAh	3.7 V 4000 mAh	3.8 V 4290 mAh	4.2 V 4000 mAh

Table 2. Processor Specifications

	laptop	**RP3**	**mobileA**	**tablet**	**mobileB**
Name	Intel Core i7-4510U (64-bit)	ARM v7 BCM2709 (64-bit)	Qualcomm Krait 400 (32-bit)	Intel Atom Z3745 (32-bit)	ARM Cortex-A72 ARM Cortex-A53 MT6797D (64-bit)
Topology	1 Processor, 2 Cores, 4 Threads	1 Processor, 4 Cores	1 Processor, 4 Cores	1 Processor, 4 Cores	1 Processor, 10 Cores
Frequency	2.0 - 3.1 GHz	1.2 GHz	2.5 GHz	1.33 - 1.86 GHz	2.3 GHz × 2 1.85 GHz × 4 1.4 GHz × 4
L1 Inst. Cache	32.0 KB × 2	16.0 KB × 1	16.0 KB × 2	32.0 KB × 4	-
L1 Data Cache	32.0 KB × 2	16.0 KB × 1	16.0 KB × 2	24.0 KB × 4	-
L2 Cache	256 KB × 2	512 KB × 1	2.00 MB × 1	1.00 MB × 1	-
L3 Cache	4.00 MB × 1	-	-	-	-

In this research, we focus on the behavior of these devices without using multithreading. In future work, we will check what happen when we use multithreading and compare it with these results.

The problems chosen for experimentation are detailed in section 2. They are 4 binary problems (MTTP, ECC, MMDP, OneMax) and 1 combinatorial problem (CVRP). The algorithm used to solve these problems is a genetic algorithm and it is explained in section 3.

Table 3. Problem Instances

Problem	Chrom. Size	Pop. Size	Probabilities			Type
			Recomb.	Mutation	Local Search	
OneMax	5000	100	0.8	$\frac{1}{chroSize}$	-	Binary
MTTP	200	100	0.8	$\frac{1}{chroSize}$	-	Binary
ECC	144	5000	0.8	$\frac{1}{chroSize}$	-	Binary
MMDP	240	5000	0.8	$\frac{1}{chroSize}$	-	Binary
CVRP	54	500	0.4	$\frac{1.2}{chroSize}$	1	Combinatorial

We have chosen one instance of each problem (Table 3). The four binary instances have been chosen from JCell Library[3]. The *CVRP* instance is the CMT1 which is one of the set of instances proposed by Christofides [11] for this problem. The chosen instances are not large because they must be solved by devices with low processing capacity. For now, what interests us is to observe the performance in each platform and not solve very large problems. This is why we have chosen a varied set of problems that can be solved in an acceptable time by all the evaluated devices.

The parameters chosen to solve the binary problems are the most common in literature and the parameters chosen to solve CVRP can be found in [10] , both are shown in Table 3. Regarding to population size, we tested with different sizes among 100, 200, 500, 1000, 2000, 5000 until we got 100% of success with 30 different seeds in each problem. For each experimental instance, we have carried out 30 independent runs with a different random seed in each repetition. Same seeds were used for each problem.

Our first experiment is to run some of the most well-known benchmarks on each platform to have an idea of their real performances. Most of these benchmarks are not multiplatform, however, we can obtain certain information using them in the platforms that we can.

Next, we evaluate the performance of each problem in each platform. Since all the devices use the same seeds, and since we are not using multithreading, all the devices solve the instances in the same number of evaluations. Therefore, what we compare is the time each device devotes to solve each instance.

Then, we analyze the use of the CPU and the memory of the different platforms when solving these instances. And finally, we compare the consumption of the battery.

4.1 Knowing Edge HW by Running Standard Benchmarks on it

First of all, we perform some benchmarks on these devices. These benchmarks are composed by the classical Whetstone [12], Dhrystone 2 [13], Linpack [14] and Livermore [15]. Also, we use Antutu benchmark[4] which is specific for Android OS, and we focus on their CPU (1 single core) and memory score. And finally we use one of the benchmarks most used today for multiplatforms devices,

[3] http://neo.lcc.uma.es/software/jcell/
[4] http://www.antutu.com/en/

Geekbench 4 (1 single core)[5]. The results of the different benchmarks have been normalized getting a score. The score 1 represents the best result and the rest of scores are proportionals to this one.

Table 4. Benchmarks

	laptop	**RP3**	**mobileA**	**tablet**	**mobileB**
GeekBench 4	**1**	-	2.15	2.43	1.38
Antutu CPU	-	-	1.12	1.26	**1**
Antutu Maths	-	-	1.06	1.90	**1**
Antutu Mem.	-	-	**1**	1.39	1.13
Whetstone	**1**	1.38	1.44	1.56	3.55
Dhrystone 2	**1**	4.42	2.65	4.13	2.18
Linpack	**1**	9.71	3.95	12.16	40.89
Livermore	**1**	13.53	6.89	17.90	5.91

The implementation of the classical benchmarks have been taken from Roy Longbottom's Benchmark Collection[6] which have implementations of many benchmark for a wide variety of platforms. The results obtained in these benchmarks are not very accurate, apparently. This is because these benchmarks are very dependents of the compilers and the hardware architectures.

The results obtained in Antutu and GeekBench 4 are closer to what we expected. The problem is that Antutu is just for Android OS and GeekBench 4 does not have an implementation for Raspbian OS.

Based on the benchmark results and the features of the devices (Table 1 and 2) we can expect that the order from the faster to the slower device is as follow: *laptop*, *mobileB*, *mobileA*, *tablet*, *RP3*.

These benchmarks help us to have a first approach to these platforms as a first step before our experiments begin. Although, we should know that these benchmarks are just a number and does not have to be similar to the final results. For instance, Antutu is so common that hardware manufacturers have taken to cheating on the benchmark which makes the benchmark unreliable.

4.2 Time Results

In Table 5 we can see the experiment results per problem and per device. The most important thing here is the time performance because the number of evaluations is the same for all devices in each problem. We have normalized the times getting a score. The device with the fastest time in each problem gets a score of 1, and the rest of them get a score proportional to this in a incremental way.

We can see that the performance in Linux devices (laptop and RP3) is much better than in Android devices. It is interesting to see that RP3, a device of €30 cost, can obtain a result very similar than a laptop even when it has a much worse CPU. RP3 needs 10 times the time of the laptop to solve the same problem, that is not too much because of the cost of the RP3 compared to the laptop.

[5] https://www.geekbench.com/
[6] http://www.roylongbottom.org.uk/

Table 5. Problem Results

		Fitness		Evaluations			Time (s)			
Device	**Prob.**	**avg**	**hit**	**avg ± sd**	**max**	**min**	**avg ± sd**	**max**	**min**	**score**
laptop	OneMax	5×10^{3}	100 %	$(18 \pm 1) \times 10^{4}$	23×10^{4}	16×10^{4}	**1.95 ± 0.14**	**2.38**	**1.76**	**1.00**
RP3	OneMax	5×10^{3}	100 %	$(18 \pm 1) \times 10^{4}$	23×10^{4}	16×10^{4}	21.47 ± 1.46	26.28	19.43	11.01
mobileB	OneMax	5×10^{3}	100 %	$(18 \pm 1) \times 10^{4}$	23×10^{4}	16×10^{4}	98.79 ± 7.65	117.13	87.46	50.66
mobileA	OneMax	5×10^{3}	100 %	$(18 \pm 1) \times 10^{4}$	23×10^{4}	16×10^{4}	99.91 ± 6.85	119.85	90.33	51.23
tablet	OneMax	5×10^{3}	100 %	$(18 \pm 1) \times 10^{4}$	23×10^{4}	16×10^{4}	176.32 ± 12.34	215.39	158.40	90.41
laptop	MTTP	25×10^{-4}	100 %	$(48 \pm 48) \times 10^{4}$	24×10^{5}	39×10^{3}	**0.52 ± 0.53**	**2.80**	**0.05**	**1.00**
RP3	MTTP	25×10^{-4}	100 %	$(48 \pm 48) \times 10^{4}$	24×10^{5}	39×10^{3}	5.86 ± 5.82	29.65	0.50	11.23
mobileB	MTTP	25×10^{-4}	100 %	$(48 \pm 48) \times 10^{4}$	24×10^{5}	39×10^{3}	55.67 ± 54.97	278.69	4.42	106.67
tablet	MTTP	25×10^{-4}	100 %	$(48 \pm 48) \times 10^{4}$	24×10^{5}	39×10^{3}	66.01 ± 65.44	334.04	5.63	126.49
mobileA	MTTP	25×10^{-4}	100 %	$(48 \pm 48) \times 10^{4}$	24×10^{5}	39×10^{3}	71.75 ± 71.93	369.39	6.48	137.49
laptop	MMDP	40	100 %	$(37 \pm 46) \times 10^{4}$	22×10^{5}	19×10^{4}	**2.12 ± 0.66**	**4.68**	**1.69**	**1.00**
RP3	MMDP	40	100 %	$(37 \pm 46) \times 10^{4}$	22×10^{5}	19×10^{4}	17.94 ± 5.66	40.09	14.29	8.47
mobileB	MMDP	40	100 %	$(37 \pm 46) \times 10^{4}$	22×10^{5}	19×10^{4}	331.35 ± 195.25	1180.94	199.50	156.46
tablet	MMDP	40	100 %	$(37 \pm 46) \times 10^{4}$	22×10^{5}	19×10^{4}	416.22 ± 74.83	683.50	349.13	196.53
mobileA	MMDP	40	100 %	$(37 \pm 46) \times 10^{4}$	22×10^{5}	19×10^{4}	476.01 ± 122.47	983.44	379.61	224.76
laptop	ECC	674×10^{-4}	100 %	$(33 \pm 6) \times 10^{4}$	45×10^{4}	23×10^{4}	**2.68 ± 0.34**	**3.41**	**1.99**	**1.00**
RP3	ECC	674×10^{-4}	100 %	$(33 \pm 6) \times 10^{4}$	45×10^{4}	23×10^{4}	25.42 ± 3.45	32.85	18.94	9.48
mobileB	ECC	674×10^{-4}	100 %	$(33 \pm 6) \times 10^{4}$	45×10^{4}	23×10^{4}	470.22 ± 75.97	621.57	310.18	175.30
tablet	ECC	674×10^{-4}	100 %	$(33 \pm 6) \times 10^{4}$	45×10^{4}	23×10^{4}	533.14 ± 66.72	672.33	403.31	198.76
mobileA	ECC	674×10^{-4}	100 %	$(33 \pm 6) \times 10^{4}$	45×10^{4}	23×10^{4}	577.81 ± 71.82	725.69	438.77	215.41
laptop	CVRP	524.61	100 %	$(45 \pm 11) \times 10^{6}$	80×10^{6}	24×10^{6}	**9.85 ± 2.45**	**17.29**	**5.09**	**1.00**
RP3	CVRP	524.61	100 %	$(45 \pm 11) \times 10^{6}$	80×10^{6}	24×10^{6}	76.39 ± 19.54	136.23	39.89	7.76
mobileB	CVRP	524.61	100 %	$(45 \pm 11) \times 10^{6}$	80×10^{6}	24×10^{6}	1637.20 ± 553.04	3389.71	771.45	166.28
tablet	CVRP	524.61	100 %	$(45 \pm 11) \times 10^{6}$	80×10^{6}	24×10^{6}	2376.80 ± 601.78	4218.04	1247.25	241.40
mobileA	CVRP	524.61	100 %	$(45 \pm 11) \times 10^{6}$	80×10^{6}	24×10^{6}	2440.79 ± 589.31	4150.62	1332.99	247.90

The bad news is the time Android devices need to solve these problems. It is clear that something is not working as well as expected in the optimization of the Java VM on Android. Android does not use Oracle Java VM, but uses Android Runtime (ART) (which replaced Dalvik Virtual Machine after Android KitKat). DVM was created to avoid JVM copyright and to better use memory and power in more limited devices. There are many studies [16] that shown that DVM was much slower than JVM, we have to say that some years later ART does not improve that results.

The tablet, and the smartphones have CPU much better than the RP3, however, they are getting a very bad performance solving these problems. Android OS is an obstacle to obtain our long-term goal and we will have to deal with it in future work. If benchmarks can obtain good results on these devices it means that it is possible and maybe we have to use native code to solve the restrictions of the Android OS.

4.3 Resources Usage

To measure memory we use the unique set size (USS) which is the portion of main memory (RAM) occupied by a process which is guaranteed to be private to that process. It was proposed by Matt Mackall because of the complications that arose when trying to count the "real memory" used by a process.

It is interesting to see in Fig. 2 how smartphones, the tablet and the RP3 manage the memory better than the laptop. It seems like the VM is optimized to use the memory better on these devices with memory restrictions than on the laptop. We have to highlight the use of memory in the tablet and in the RP3 that need much less memory than the other devices to solve the same problems.

Regarding to CPU, we have to say that to run the same java code on Android we needed a simple interface with a button to start the algorithm on background. That is why Android devices are using two cores when laptop and RP3 are using only one. However, we can see how the tablet uses less CPU than smartphones. It seems that Intel x86 is using the CPU in a smarter way than ARM/AArch on Android OS.

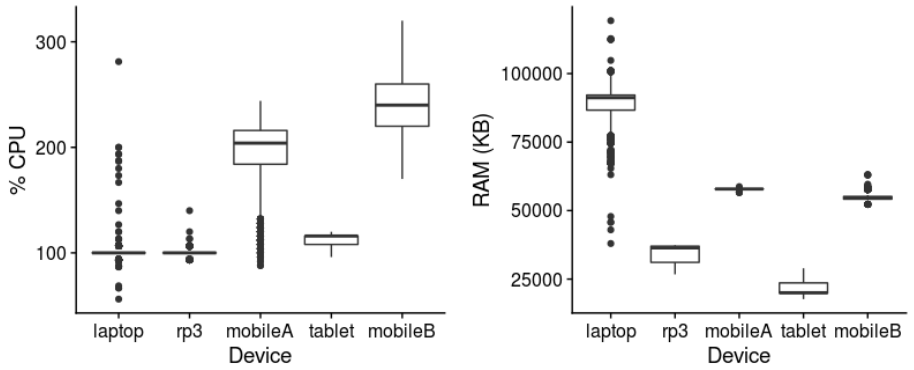

Fig. 2. Resources Usage.

4.4 Battery Consumption

We have measured the battery consumption during 2 hours with and without the algorithm running. As a result, we have obtained that all devices increment their battery consumption in a similar way from 1.5 to 1.97 depending on the device. The laptop increment their battery consumption 1.72 with respect to their normal consumption, RP3 did 1.8, mobileA did 1.97, tablet did 1.5 and mobileB did 1.76.

These are good results comparing it with making a call, web browsing or playing a video. These activities can increment battery drain in 5 or 6 times depending on the device.

5 Conclusions

In this article, we have analyzed the performance of five different devices while running smart algorithms in the edge. We have measured their performance, CPU and memory usage, and battery consumption.

First, we chose a representative set of problems with different interesting features and solved them using a GA. Second, we showed that benchmarks are not so useful to known the real performance of the devices although they can help to have a first impression about it. Also, we showed that OS could be more important than HW to get good results running these algorithms. Another remarkable result is that battery consumption is lower than many applications we commonly use, and much more lower than a call for instance.

Finally, we have got many unexpected results. On the one hand, RP3 arises as a perfectly suitable platform to run algorithms in the edge. RP3 is very cheap, and it uses less memory than laptop and an acceptable time solving the problems. RP3 is commonly used in smart city sensors, so it is very suitable device to perform edge computing. On the other hand, Android devices got very poor

results in terms of performance. It seems like Android OS has many restrictions to run high-performance applications, Android OS seems more interested in maintaining battery consumption low and in using less memory (cleaning garbage collector more often). Anyway, this only encourages us to look for new approaches to solve the problem of performance in Android. This platform is crucial for our long-term goals, so in future work we will use native code (JNI).

We have shown how devices with low processor capacity as RP3 are perfectly useful solving optimization problems. In the case of smartphones and tablets, we still have to deal with the limitations of Android OS. In future work, we will check what happen when we use multithreading and we will research to new ways to improve the performance in Android devices like using native code.

References

1. Index, C.G.C.: Forecast and methodology, 2015-2020 white paper. Retrieved 1st June (2016)
2. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal **3**(5) (2016) 637–646
3. Levine, P.: Return to the edge and the end of cloud computing. https://a16z.com/2016/12/16/the-end-of-cloud-computing/ (Dec. 2016) Online; accessed 17 February 2018.
4. Morell, J., Alba, E.: Distributed genetic algorithms on portable devices for smart cities. In: International Conference on Smart Cities, Springer (2017) 51–62
5. Goldberg, D.E., Deb, K., Horn, J.: Massive multimodality, deception, and genetic algorithms. Urbana **51** (1992) 61801
6. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes. Elsevier (1977)
7. Stinson, D.: An introduction to the design and analysis of algorithms. The Charles Babbage Research Centre, St. Pierre (1985)
8. Eshelman, L.: On crossover as an evolutionarily viable strategy. In: Proceedings of the Fourth International Conference on Genetic Algorithms. (1991) 61–68
9. Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Management science **6**(1) (1959) 80–91
10. Alba, E., Dorronsoro, B.: Cellular genetic algorithms. Volume 42. Springer Science & Business Media (2009)
11. Christofides, N., Mingozzi, A., Toth, P.: Loading problems. N. Christofides and al., editors, Combinatorial Optimization (1979) 339–369
12. Curnow, H.J., Wichmann, B.A.: A synthetic benchmark. The Computer Journal **19**(1) (1976) 43–49
13. Weicker, R.P.: Dhrystone: a synthetic systems programming benchmark. Communications of the ACM **27**(10) (1984) 1013–1030
14. Dongarra, J.J., Bunch, J.R., Moler, C.B., Stewart, G.W.: LINPACK users' guide. Siam (1979)
15. McMahon, F.H.: The livermore fortran kernels: A computer test of the numerical performance range. Technical report, Lawrence Livermore National Lab., CA (USA) (1986)
16. Batyuk, L., Schmidt, A.D., Schmidt, H.G., Camtepe, A., Albayrak, S.: Developing and benchmarking native linux applications on android. In: International Conference on Mobile Wireless Middleware, Operating Systems, and Applications, Springer (2009) 381–392