A time performance comparison of particle swarm optimization in mobile devices

Luis Antonio Beltrán Prieto[1,a], Zuzana Komínkova-Oplatková[1], Rubén Torres Frías[2] and Juan Luis Escoto Hernández[2]

[1]*Faculty of Applied Informatics, Department of Informatics and Artificial Intelligence, Tomas Bata University in Zlín, Nad Stráněmi 4511,76005 Zlín, Czech Republic*
[2]*Department of Systems and Computing, Technological Institute of Celaya, Av. Tecnológico y A. García Cubas S/N, 38010, Celaya, México*

Abstract. This paper deals with the comparison of three implementations of Particle Swarm Optimization (PSO), which is a powerful algorithm utilized for optimization purposes. Xamarin, a cross-platform development software, was used to build a single C# application capable of being executed on three different mobile operating systems (OS) devices, namely Android, iOS, and Windows Mobile 10, with native level performance. Seven thousand tests comprising PSO evaluations of seven benchmark functions were carried out per mobile OS. A statistical evaluation of time performance of the test set running on three similar devices –each running a different mobile OS– is presented and discussed. Our findings show that PSO running on Windows Mobile 10 and iOS devices have a better performance in computation time than in Android.

1 Introduction

The aim of this paper is to compare the performance of three implementations of a single C# mobile application which examines the Particle Swarm Optimization (PSO) algorithm running on three mobile devices, each with a different mobile operating system installed, specifically Android, iOS, and Windows Mobile 10. One thousand simulations of PSO evaluating seven benchmark functions were carried out per mobile OS, thus 21, 000 tests were performed in total. Xamarin platform made it possible to develop a single application in C# and deploy it to different mobile OS devices.

Smartphones are multitasking mobile devices in which installed applications should have an efficient use of resources, including CPU, memory, and sensors, in order to increase performance and maximize their battery life. Most of the time, if a mobile application is slow or consumes too many resources, e.g. mobile data, battery, or storage, it is either uninstalled from the mobile device or scored with a low review by the user in the mobile app store. Moreover, decision-making applications, such as travel destination recommenders and games, are may require intensive computing in order to get the best choice, or at least an optimal one. Even though most of the time a more powerful device, e.g., a server, deals with this task –giving the mobile device the assignment of only presenting the results to the user–, sometimes the mobile device requires to process information on its own, with no aid of external server. In other words, dealing with data and finding the best choice among different options are problems which mobile applications are not excluded to solve.

PSO is an optimization technique developed by Kennedy and Eberhart [1] inspired by the collective behaviour of animal groups, such as swarms of insects, in order to build a swarm of particles, i.e., a set of candidate solutions which flow through the parameter space generating trajectories driven by the best individuals. The initial population (swarm) consists of random solutions (particles) for the problem and is considered as a population of homogeneous agents which interact locally with other individuals without any central control. As a result, collective behaviour is generated, thus evolution relies on cooperation and competition among individuals through the different epochs (generations). Each particle defines trajectories in the parameter space according to a motion function which is affected by velocity, inertia, cognitive coefficient and social coefficient. The objective is to find the global best solutions by stochastic weighting of the aforementioned elements. The process is iterative until a stopping criterion is met.

Xamarin Platform is a mobile application development tool used to build native and cross-platform applications which can be deployed to Android, iOS, and Windows Mobile devices with native-level performance, native user interface, and a full access to the APIs of each platform. These applications are written once in C# language, sharing the same code across multiple mobile operating systems. Not only the business logic can be shared when building a mobile application with Xamarin, but also the user interface can be distributed by writing it either in C# or in eXtensible Application Markup

[a] Corresponding author: beltran_prieto@fai.utb.cz

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

Language (XAML). When deployed and tested into a specific platform, the application presents a native user interface, e.g., a Xamarin.Forms.Button is depicted as a UIButton on iOS, as an Android.Widget.Button on Android, and as a System.Windows.Controls.Button on Windows Mobile devices.

This paper is organized as follows. In the first section, a theoretical background on smartphones, particle swarm optimization, and Xamarin platform is presented. Afterwards, the methods and methodology that was used for this comparison are described. Then, evaluation results, statistical comparison and discussion are presented together. Finally, conclusions are shown at the end of the paper.

2 Background information

Smartphones are an important lifestyle device. There are almost 3.5 billion of smartphone subscriptions worldwide [2], and the trend is expected to double in size in the next lustrum. People use their smartphones on a daily basis in order to make their lives easier anytime and anywhere, for example, online purchases can be done while commuting thanks to a mobile application, thus it is not necessary to wait until a destination is reached and then use a PC. Another example of the advantages given by a smartphone is that it allows people to be communicated all the time.

Despite of the fact that smartphones are limited devices, they offer a PC user-experience, that is, they provide capabilities such as multitasking, video streaming, audio streaming, and web browsing, among others [3]. Therefore, they can be used intensively through the day while requiring a substantial usage of resources in order to work. When there are many applications working in the background, available memory can be significantly reduced. As a consequence, the smartphone's usability isn't user-friendly. In order to overcome this problem, the dispatcher, which is a privileged component of the operating system, terminates low-priority applications and frees resources until it determines that the available memory is enough to continue working. Moreover, battery life can be drained faster if there are too many programs running concurrently or if an application does not perform optimally. If a mobile app is killed by the dispatcher, it is not guaranteed that it would be properly closed as it didn't complete its lifecycle, so non-stored information will be lost and unrecoverable.

Artificial Intelligence (AI) algorithms, methods and techniques can be applied to build smart applications which delivers valid results to a user's request [4]. For instance, a geospatial analysis can be obtained by combining the use of geometric design with sensors and components from a smartphone, such as GPS, accelerometer, and camera [5]. Moreover, a deep learning neural network can perform effective human activity recognition with the aid of smartphone sensors [6]. Furthermore, Genetic Programming (GP) has been used to create a customized smartphone user-experience [7]. As can be seen, AI methods provide solutions for today's mobile world challenges.

2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an example of stochastic optimization inspired by the social behaviour of animal groups [8]. Initially developed by Kennedy and Eberhart in 1995 [1], this technique has proven to be effective for neural networks weight calculation [9], business optimization [10], and parameter estimations [11]. Both behaviour and efficiency of the algorithm rely on the parameters shown in Table 1 [1]. Meta-optimization of the parameters has been used to tune them and find the best values which benefit PSO's performance in particular scenarios [12-15].

Table 1. Particle Swarm Optimization parameters.

Parameter	**Meaning**	**Typical Range** [8]
ω	Inertia weight	[0.8, 1.2]
c_1	Cognition learning rate	[0, 4]
c_2	Social learning rate	[0, 4]
N	Number of particles	[20-40]

The algorithm works as follows: First, a population of random N candidate solutions $\{x_1, x_2, \ldots, x_N\}$ is generated. Each individual is an N-dimensional vector (N > 1) with values within the problem bounds. Moreover, each particle contains a velocity vector v_i, which is also randomly initialized. By performing fitness evaluation, the best position at the moment, b_i, is obtained for each individual along with the best global position h_i. The iterative process starts by generating two random values, r_1, r_2, which are used to update each particle's velocity and position according to equations (1) and (2), respectively:

$$v_i = \omega v_i + c_1 r_1 (b_i - x_i) + c_2 r_2 (h_i - x_i) \quad (1)$$

$$x_i = x_i + v_i \quad (2)$$

Fitness evaluation is performed again in order to determine both the best position for each particle and the best global particle. The iterative process is repeated until a stopping criterion, such as a predetermined number of generations, is met. There are several variants of the PSO algorithm [16]. For instance, v_i in (1) is not affected by ω in the original version of the algorithm. Another variation consider the initialization of the particles as the most important element in order to improve the performance of the algorithm [17]. Opposition-based learning, a term that describes an individual's exact contrary, has also been considered as an enhanced variant of PSO which accelerates convergence [18-20] by replacing individuals which are far to the optimal solution by their opposite, which is closer in distance to the solution.

2.2 Xamarin Platform

Xamarin is a novel cross-platform mobile application development tool which can be used to build native Android, iOS and Windows applications that rely on a shared C# codebase. One of the main advantages of Xamarin is that allows developers to write an application once and deploy it to different mobile OS platforms without having to rewrite it in an completely different programming language, operating system or application programming interface (API) [21]. Even though all the platforms share similarities, such as graphical user interface (GUI) presentation, device sensors, and gestures interaction, each of them incorporates many differences, including navigation, user-experience, and fragmentation [22]. Xamarin's development of cross-platform solutions is possible mainly because of the Mono project, an open-source implementation of Microsoft's .NET Framework that can run on Linux, Windows, and Mac OS X [23] which is considered as the core of Xamarin, comprising three sets of .NET libraries: Xamarin.Mac, Xamarin.iOS, and Xamarin.Android. Xamarin.Forms (XF) allows developers to write code capable of being compiled and deployed to mobile devices, regardless of their OS. A basic XF application contains separate projects per desired target mobile platform -currently, Android, iOS, Windows Phone 8.1, Windows 8.1, and Universal Windows Platform platforms are supported [24]- plus another project which contains the common code that will be compiled and used at runtime by each specific platform together with the code contained in the project platform. Both the user-interface (UI) and the business logic (BL) can be developed in C# language, while eXtensible Application Markup Language (XAML) can also be used for the UI. On the one hand, the UI refers to the views or controls that will be shown on the device screen. On the other hand, the BL includes all the functionality of the application. It is also worth mentioning that platform-specific code can be added to the project to address key platform differences, such as screen sizes, navigation issues and push notifications, and device differences, for instance, sensors functionality, social networks interaction, and geo-location support.

3 Methods and methodology

The objective of this experiment is to evaluate the performance of an implementation of the Particle Swarm Optimization algorithm running on 3 smartphones with different mobile OS each (Android, iOS, and Windows Mobile) which evaluates 7 well-known benchmark functions for optimization (Ackley [25], Goldstein-Price [26], Rastrigin [27], Rosenbrock [28], Rotated Hyper-Ellipsoid [29], Sphere [30] and Sum Squares [31]). Table 2 provides a brief overview of the tested benchmark functions, while Table 3 outlines the relevant specifications of the 3 mobile devices [32] used for the analysis. Except for the Goldstein-Price function, which by definition operates only in a 2-dimensional space, all benchmarks were evaluated in a 10-dimensionall space

A C# version of the PSO algorithm was programmed and included in a mobile application developed with Xamarin platform using the Visual Studio Community 2015 IDE. Figure 1 shows the structure of the solution. Four projects were included in the solution. The first one, PSO, contains all the shared code. The other three, PSO.Droid, PSO.iOS, and PSOUWP represent the specific projects that were deployed to the particular mobile OS, namely Android, iOS, and Windows 10, respectively. It is worth mentioning that no code was added to the three projects due to Xamarin's code-sharing leveraging technique.

In all cases, $\omega = 0.729$, $N = 40$, $c_1 = 1.49445$, $c_2 = 1.49445$ values were used. For the benchmark functions, the search domain was set for the range indicated in Table 2. Finally, during the experiment, each mobile had neither processes working in the background nor background services and sensors active, except for the PSO mobile application developed in Xamarin and the Wi-Fi signal because the computation times were sent to an Azure SQL cloud database for further analysis.

One thousand simulations of the Particle Swarm Optimization full algorithm per benchmark function evaluation and per mobile OS were carried out, i.e., each time a new random initial population was generated and evolved in order to solve a benchmark function. Figure 2 depicts the process of a single run of the app in each device, while Figures 3 and 4 show the application being executed on Windows Mobile 10 and Android devices, respectively.

4 Results and discussion

The findings of the analysis are summarised in Table 4. A comparison of the average running time of 1000 simulations of PSO evaluating one benchmark function at a time on a mobile device is presented. The best (fastest) results in each case are marked in bold, while the worst ones are shown in italics. On the one hand, it is clearly seen that the Android device was far outperformed by both iOS and Windows Mobile 10 devices approximately two times, except for Rosenbrock and Rotated functions, in which average times are closer. On the other hand, the iOS and Windows Mobile 10 devices had similar performances in terms of time, with the latter being the fastest in 4 out of the 7 benchmark evaluations.

As iOS and Windows Mobile 10 computation times look similar, a statistical analysis can be performed to determine if there are significant differences between them. Firstly, an extended Shapiro-Wilk test [33] was conducted in order to assess the normal distribution of both samples. Table 5 and Table 6 show the findings of this test at the significance level of 0.05. The null hypothesis, H_0, in each case is set as "the computation times of a given benchmark-function evaluation on a mobile device are normally distributed".

Table 2. Tested benchmark functions.

Benchmark function	Equation	
Ackley	$f(x) = -a\,exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$	(8)
	Recommended values	a = 20, b = 0.2, c = 2π
	Global Minimum	f(x*) = 0, at x* = (0, …, 0)
	Search domain	$x_i \in [-32{,}768, 32768]$
Goldstein-Price	$f(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	(11)
	Minimum	f(x*) = 3, at x* = (0, –1)
	Search domain	$x_i \in [-2, 2]$
Rastrigin	$f(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)]$	(10)
	Minimum	f(x*) = 0, at x* = (0, …, 0)
	Search domain	$x_i \in [-5.12, 5.12]$
Rosenbrock	$f(x) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	(12)
	Minimum	f(x*) = 0, at x* = (1, …, 1)
	Search domain	$x_i \in [-5, 10]$
Rotated Hyper-Ellipsoid	$f(x) = \sum_{i=1}^{d}\sum_{j=1}^{i}x_j^2$	(13)
	Minimum	f(x*) = 0, at x* = (0, …, 0)
	Search domain	$x_i \in [-65.536, 65.536]$
Sphere	$f(x) = \sum_{i=1}^{d}x_i^2$	(13)
	Minimum	f(x*) = 0, at x* = (0, …, 0)
	Search domain	$x_i \in [-5.12, 5.12]$
Sum Squares	$f(x) = \sum_{i=1}^{d}ix_i^2$	(13)
	Minimum	f(x*) = 0, at x* = (0, …, 0)
	Search domain	$x_i \in [-5.12, 5.12]$

*W** stands for the observed test statistic value. As can be seen, the statistical test clearly indicates that all benchmark-function evaluations on both mobile OS follow a normal distribution because our proposed null hypothesis is accepted, as $p(W^*) > 0.05$ in all cases.

Secondly, a Bartlett's test for variances homogeneity [34] can be performed. In this case, our null hypothesis, H_0, is set as "The variances of two samples of given benchmark-function evaluations running on different mobile OS devices are homogeneous".

Table 3. Relevant specifications of the smartphones used in the analysis.

Model	Brand	Mobile OS	Chipset	CPU	Memory
Lumia 930	Nokia	Windows Mobile 10	Qualcomm MSM8974 Snapdragon 800	Quad-core 2.2 GHz Krait 400	2 GB RAM
Nexus 5	LG	Android 6.0	Qualcomm MSM8974 Snapdragon 800	Quad-core 2.3 GHz Krait 400	2 GB RAM
iPhone 5s	Apple	iOS 8.4	Apple A7	Dual-core 1.3 GHz Cyclone (ARM v8)	1 GB RAM

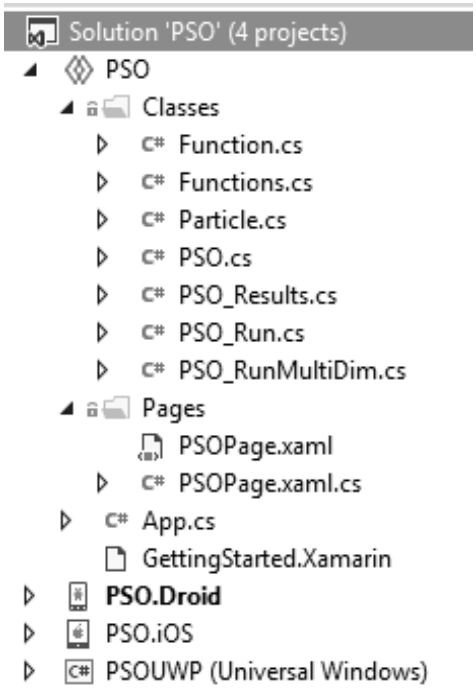

Figure 1. Solution structure of the PSO mobile application.

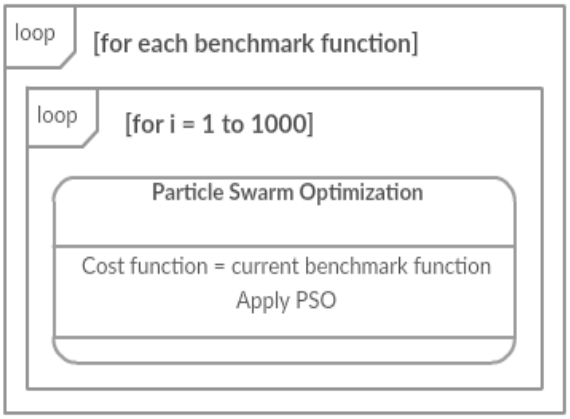

Figure 2. Solution structure of the PSO mobile application.

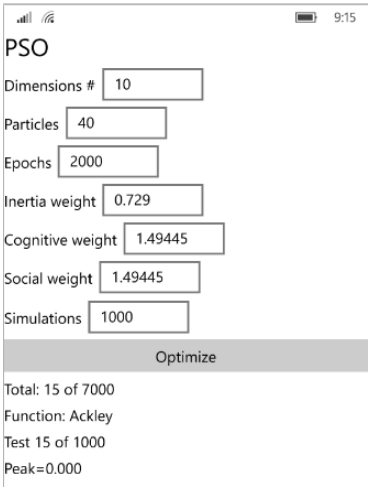

Figure 3. PSO app running on a Lumia 930.

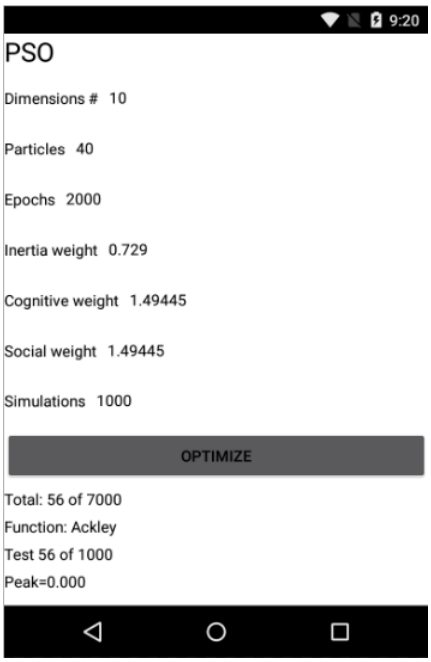

Figure 4. PSO app running on a LG Nexus 5.

Table 4. Average (mean) computation time (in miliseconds) taken by each mobile device to evaluate PSO using a benchmark function.

Benchmark Function	Android	W10	iOS
Ackley	*1496.77*	**711.696**	742.21
Goldstein-Price	*1205.98*	457.78	**432.64**
Rastrigin	*1482.86*	**685.69**	697.83
Rosenbrock	*1895.84*	**1433.79**	1452.47
Rotated	*3002.09*	2774.83	**2608.03**
Sphere	*1368.26*	**666.28**	682.59
Sum Squares	*1389.05*	617.17	**603.72**

Table 5. Shapiro-Wilk test for normal distribution of mean computation times (in miliseconds) in Windows 10 device.

Function	Mean	W*	p-value	H_0: p > 0.05?
Ackley	711.70	0.99825	0.4012	**Yes**
Goldstein-Price	457.78	0.99855	0.5887	**Yes**
Rastrigin	685.69	0.99734	0.09953	**Yes**
Rosenbrock	1433.8	0.99791	0.2471	**Yes**
Rotated	2774.8	0.99856	0.5924	**Yes**
Sphere	666.28	0.99759	0.1505	**Yes**
Sum Squares	617.17	0.997	0.05737	**Yes**

Table 6. Shapiro-Wilk test for normal distribution of mean computation times (in miliseconds) in iOS device.

Function	Mean	W*	p-value	H_0: p > 0.05?
Ackley	742.21	0.99787	0.2324	**Yes**
Goldstein-Price	432.64	0.99897	0.8563	**Yes**
Rastrigin	697.83	0.99788	0.2367	**Yes**
Rosenbrock	1452.47	0.99815	0.3528	**Yes**
Rotated	2608.03	0.99785	0.2255	**Yes**
Sphere	682.59	0.99769	0.1758	**Yes**
Sum Squares	603.72	0.99799	0.2787	**Yes**

For example, we want to test whether the evaluations of the Ackley function in Windows Mobile 10 have the same variance as the evaluations of the same benchmark function in iOS or not. Table 7 outlines the variances of each benchmark function per mobile device as well as the Bartlett's test at the 0.05 significance level, with one degree of freedom ($k - 1 = 1$, where k is the number of samples, in this case, 2), i.e., $\chi_{0.95,\ 1} = 3.8414$. B^* stands for the observed test statistic value. As can be seen from the table, the test clearly indicates that all except one pair of benchmark-function evaluations on both mobile OS, namely the Rotated-function evaluation, don't have homogeneous variances because our proposed null hypothesis is rejected, as $\chi_{0.95,\ 1} > B^*$ in 6 out of 7 cases. In other words, only the variances of the Rotated-function comparison between iOS and Windows Mobile 10 are homogeneous.

Table 7. Bartlett's test for variances homogeneity of computation times (in miliseconds).

Benchmark function	Variance W10	Variance iOS	B*	H_0: $\chi_{0.95,\ 1} > B^*$?
Ackley	159.6535	289.12	100.99	**No**
Goldstein-Price	664.7554	1352.602	169.52	**No**
Rastrigin	311.917	218.3957	40.683	**No**
Rosenbrock	2477.584	5458.145	91.039	**No**
Rotated	1847.163	1578.721	0.48789	**Yes**
Sphere	348.5762	271.1668	30.404	**No**
Sum Squares	1310.597	1759.304	9.9097	**No**

In order to use the Analysis of Variance (ANOVA) statistical test between the running times in each mobile OS device, there are two assumptions for the populations: first, they have to follow a normal distribution; second, their variances have to be similar. On the one hand, the previous Shapiro-Wilk test proved that the first requirement is accomplished. On the other hand, the aforementioned Bartlett's test demonstrated that the second condition is not met, though. For that reason, a non-parametric Friedman statistical test [35] can be used to detect differences between our samples. Table 8 outlines the results of the Friedman test at the 0.05 significance level with one degree of freedom ($k - 1 = 1$, where k is the number of samples, in this case, 2). This test also relies on the chi-square critical value, i.e., $\chi_{0.95,\ 1} = 3.8414$. Our null hypothesis, H_0, for each case is set as "there is no significant difference between Windows Mobile 10 and iOS evaluations of a given benchmark function". As shown in the previous table, the null hypothesis is certainly accepted in all evaluations, meaning that there is no significant difference between the computation times of the benchmark-function evaluations in both mobile OS.

Table 8. Friedman test for benchmark-functions running times in each mobile OS device.

Benchmark function	Q*	H_0: $\chi_{0.95,\ 1} > Q^*$?
Ackley	0.784	**Yes**
Goldstein-Price	0.4	**Yes**
Rastrigin	0.256	**Yes**
Rosenbrock	0.576	**Yes**
Rotated	0.004	**Yes**
Sphere	0.016	**Yes**
Sum Squares	0.064	**Yes**

5 Conclusion

Smartphone applications require a careful management of resources in order to provide a proper user-experience. Despite of the fact that huge workloads are often assigned to servers, mobile devices are also capable of accepting time-consuming tasks which most of the time run in the background as the response time of a mobile app should be fast. In this study, we implemented Particle Swarm Optimization algorithm in order to test and compare their performance when being executed on three devices with different mobile OS, namely Android, iOS and Windows Mobile 10. Our findings show that Android got the worst performance, while a statistical analysis of iOS and Windows Mobile 10 computation times revealed that there was no significant difference between them. Future research will be focused on implementing PSO in Java and Swift, which are Android and iOS programming languages used for developing native apps respectively. Further, another statistical comparison of both the native and the C#-cross-platform current implementation in Xamarin that was shown in this paper can be done in terms of computation time and performance.

Acknowledgement

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Programme project No. LO1303 (MSMT-7778/2014) and also by the European Regional Development Fund under the project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089, further it was supported by Grant Agency of the Czech Republic—GACR 588 P103/15/06700S and by Internal Grant Agency of Tomas Bata University in Zlin under the project No. IGA/CebiaTech/2016/007.

References

1. J. Kennedy, R. C. Eberhart. *Proceedings of IEEE Int'l. Conf. on Neural Networks*, pp. 1942-1948 (1995)
2. Ericsson 2016 Mobility Report. http://www.ericsson.com/mobility-report [Online: accessed 16-Abr-2016]
3. S. Li, S. Mishra. J. of Par. and Dist. Comp., (2016)
4. R. Malaka. Workshop Notes of AIMS200, pp. 5-6 (2000)
5. S. Higuera de Frutos, M. Castro. Transp. Res. Part C Emerg. Technol. **38** (2014)
6. C. A. Ronao, S.-B. Cho. Expert Syst. Appl. **59**, pp. 235-244 (2016)
7. P. Valencia, A. Haak, A. Cotillon, R. Jurdak. Appl Soft Comput. **25** pp. 86-96 (2014)
8. X. Hu. Particle Swarm Optimization Home Page http://www.swarmintelligence.org [Online: accessed 16-May-2016]
9. M. Meissner, M. Schmuker, G. Schneider, BMC Bioinformatics **7**, 125 (2006)
10. X. S. Yang, S. Deb, S. Fong. *Proceedings of the 3rd Int'l. Conf. NDT2011*, pp. 53–66 (2011)
11. R. Susuki, F. Kawai, C. Nakazawa, T. Matsui, E. Aiyoshi. *Proceedings of SICE Annual Conf.*, pp. 1981-1988, (2008)
12. M. Schwaab, E. C. Biscaia Jr., J. L. Monteiro, J. C. Pinto. Chem. Eng. Sci. **63**, 1542–1552, (2008)
13. F. Marini, B. Walczak. J. Chemometr. **25**, 366–374, (2011)
14. A. N. Skvortsov. J. Chemometr. **28**, 727–739, (2014)
15. Q. Shen, J.-H. Jiang, C.-X. Jiao, G.-L. Shen, R.-Q. Yu. Eur. J. Pharm. Sci. **22**, 145–152, (2004)
16. M. Imran, R. Hashim, N. E. A. Khalid. Procedia Eng. **53**, 491-496, (2013)
17. N. Q. Uy, N. X. Hoai, R. McKay, P. M. Tuan. *Proceedings of IEEE C EVOL COMPUTAT*, pp. 1985-1992 (2007)
18. H. Jabeen, Z. Jalil, A.R Baig. *Proceedings of GECCO 2009*, pp. 2047- 2052 (2009)
19. M. G. H. Omran, S.al-Sharhan. *Proceedings of IEEE SIS 2008*, pp 16 (2008)
20. C. Zhang, Z. Ni, Z. Wu, L. Gu. *Proceedings of IFITA '09*. pp. 325-330 (2009)
21. C. Petzold. *Creating Mobile Apps with Xamarin.Forms. Cross-platform C# programming for iOs, Android, and Windows Phone*. (Microsoft Press, 2015)
22. S. Mansfield-Devine. Network Security, **2012**, 10, pp. 5-12 (2012)
23. Mono-Project. About Mono. http://www.mono-project.com/docs/about-mono [Online: accessed 22-Abr-2016]
24. Xamarin Inc. Xamarin + Universal Windows Platform. https://developer.xamarin.com [Online: accessed 24-Abr-2016]
25. D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing* (Kluwer Academic Publishers, 1987)
26. P. Serra, A. F. Stanton, S. Kaisb. J Chem Phys, **106** pp. 7170–7177, (1997)
27. L. A. Rastrigin. In Theoretical Foundations of Engineering Cybernetics Series (1974)
28. H. H. Rosenbrock. Comput. J., **3**, 3, pp. 175–184 (1960)
29. A. K. Qin, V. L. Huang, P. Suganthan. IEEE Trans. Evol. Computat. **13**, 2, pp. 398-417 (2009).
30. D. Simon. *Evolutionary Optimization Algorithms: Biologically Inspired and Population-Based Approaches to Computer Intelligence* (Wiley, 2013)
31. G. A. Jones, J. M. Jones. *Elementary Number Theory* (Springer SUMS, 1998)
32. GSMArena.com Finder. http://www.gsmarena.com [Online: accessed 22-Abr-2016]
33. A. C. Elliott, W. A. Woodward. *Statistical analysis quick reference guidebook with SPSS examples.* (Sage Publications, 2007)
34. M. S. Bartlett. J R Stat Soc Ser A Stat Soc. **160**, pp. 268–282 (1937)
35. M. Hollander, D. A. Wolfe. *Nonparametric Statistical Methods* (John Wiley & Sons, 1973)