Mind the gap – a distributed framework for enabling energy optimisation on modern smart-phones in the presence of noise, drift, and statistical insignificance

Mahmoud A. Bokhari
[1] *Optimisation and Logistics*
University of Adelaide, Australia
[2] *Computer Science Department*
Taibah University
Kingdom of Saudi Arabia
mahmoud.bokhari@adelaide.edu.au

Lujun Weng, Markus Wagner, Bradley Alexander
Optimisation and Logistics
University of Adelaide, Australia
lujunweng@outlook.com
markus.wagner@adelaide.edu.au
bradley.alexander@adelaide.edu.au

***Abstract*—Smartphones are becoming essential to people's everyday lives. Due to the limited battery capacity of smartphones, researchers and developers are increasingly interested in the energy efficiency of these devices and the software applications that run on them. In the most basic setting, a developer might be interested in knowing which of two program variants might consume more energy, whether this is for use in regression testing or for use in full-scale evolutionary optimisation. To perform such comparisons (tournaments) reliably, we need a model of the number of trials needed to discern between two variants to a desired level of statistical significance. To enable this, we present a conceptual framework based on tournaments which we use to compare a range of test workloads on different combinations of phones and operating systems. Our results quantify the number of trials required to resolve different variants to different levels of fidelity on a range of platforms.**

***Index Terms*—Smart phones, energy consumption, non-functional properties, tournaments.**

I. INTRODUCTION

Smartphones are becoming an essential part of their users everyday lives. A limiting factor in smartphone capabilities is battery lifetime, and a key determinant of battery lifetime is the energy consumption of software running on the phone. Consequently, there is increasing interest from researchers and developers in the optimisation of energy consumption. In order to optimise software energy consumption, the optimiser needs energy measurements. The gold standard for obtaining energy measurements is external meters. These directly and accurately measure the electrical current drawn by the smartphone. The best meters can sample at up to 125kHz with only 0.7% estimation error [1]. Such meters have been successfully applied to energy optimisation on mobile devices [2]–[4]. However, as meter accuracy increases, the price of the meter also increases. Moreover, setting up an external meter involves a complicated process, requiring specialised technical skills since practitioners need to open the device to attach the meter to its battery [5]. This factor makes it infeasible for most software developers to use high-standard external meters.

Another measurement solution is the use of energy models. There are several techniques under this umbrella. Models can estimate the energy use by utilising the hardware (HW) and software components' counters [6], [7]. Other works extend this methodology to include the different states induced by HW components and system calls [8], [9]. The last category utilises the executed line of codes (LOC) for energy estimation [10], [11]. Energy models are less accurate compared to the hardware meter but more convenient and accessible to practitioners.

However, the dilemma here is which model to use? Energy models based on HW utilisation models do not capture all factors that contribute to the energy consumption. For instance, these models do not consider the states of network interface controllers (NICs) during data transmission. State and system call models cannot be easily applied when HW manufactures do not publish the relevant information about their products [8]. LOC based models, while conceptually simple, are difficult to maintain. There are thousands of APIs in the AndroidJava SDKs, and they evolve rapidly at rate of 115 API updates per month [12]. Moreover, the energy use profile of software is very platform dependent and can vary in complex, state-dependent and even individual-device-dependent ways that are very challenging to model [4], [5], [13].

A solution that alleviates these issues is to utilise the internal meter of smart-devices and compare the energy consumption of software variants *in-vivo*. In contrast to external meters, internal meters do not require advanced electrical engineering skills and are accessible to software practitioners. Internal meters are less accurate compared to external meters, however, they can be precise [14], [15] and can be an adequate substitute to obtain real measurements if the measurement periods are sufficiently long. Battery readings from the internal meter are easy to access through the battery APIs, such as BatteryManager in the Android SDK [16]. In addition, internal meters do not suffer from the inherent inaccuracy of models – they show what software variants actually use at a particular moment.

However, like all measurements of real systems, measurements of energy consumption using the internal meter are

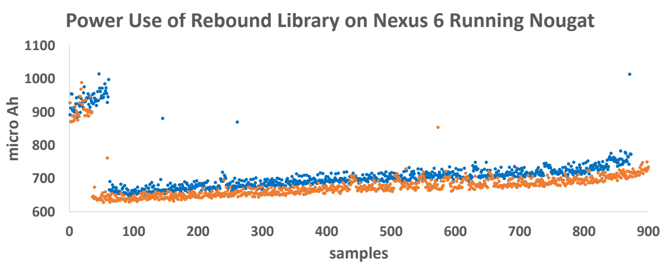

Fig. 1: Power use of repeated runs ("samples") of the original Rebound library on Nexus 6 running Android 7. Setup: Rebound was run 900 times (blue), then the device was recharged, then Rebound was run again 900 time.

subject to noise. As an example of this noise Fig. 1 shows repeated measures of energy consumption, as measured by the internal meter, for the same variant of a physics library running on a Nexus 6 using Android 7 – in particular, the only difference between the two repetitions (one orange one blue) was that the battery was recharged between the two repetitions. As can be seen, the measurements exhibit significant random and systematic noise. Such noise can be created by a variety of factors including, battery state, system state, and temperature, among others. This noise has the potential to impact on the accuracy of fitness evaluations performed during comparisons of program variants which, in-turn, can affect any optimisation process informed by these comparisons.

A common approach to coping with noise in fitness evaluations is re-sampling [17]–[28]. When an optimisation process uses re-sampling, variants are run multiple times with the number of runs (samples) determined by some function of an estimation of system noise. In work to date, re-sampling strategies have assumed that noise can be characterised by an underlying normal distribution.

In this work we explore the problem of running a tournament to compare the energy use of two variants of a software application on a mobile platform by using that platform's internal meter. We explore the distributions of measured energy produced by known workloads on different platforms in different battery states. We demonstrate that these distributions are often complex and, none so far, conform to normal distributions. From these observed distributions we then infer the number of consecutive samples required to rank two variants to a given degree of confidence at a given battery state. We show that the number of samples required not only depends on the difference between the running times of each variant, but also on the battery state, and, very importantly, the HW/OS combination on the platform. We use this information to propose a number of re-samples for each platform and battery state. Finally, we run a simple, proof-of-concept optimisation experiment on a simple benchmark to compare our re-sampling strategy to two other published strategies. These early experiments produce promising indications of the potential effectiveness of our approach.

II. Related work on re-sampling

Optimisation processes for real systems have to be robust to noise in fitness evaluations of individuals. There are three basic approaches to coping with noise in evolutionary optimisation. The first approach is implicit-averaging [29]. Under this approach we simply rely on a large population of individual variants to contain similar individuals. When very similar individuals are evaluated we are approximating a re-sampling [28], [30]. One drawback of this approach is the requirement for large populations which are not easily supported when fitness evaluations are expensive.

A second, more direct, approach is simply to re-sample a fixed number of times according to some characterisation of noise [30]. A drawback of this approach is that it is not able to adapt the re-sampling rate according to circumstances as an optimisation run progresses. For example, changes in system state during optimisation may induce more or less noise [31].

The third strategy, dynamic sampling (DS) addresses this problem by allowing the re-sampling strategy to adapt during optimisation. A simple DS technique is time-based sampling (TBS) [17], [18]. Under TBS the re-sampling rate increases as optimisation progresses based on the observation that individuals in a population approach similar levels of fitness in later generations. Confidence-based DS [22] determines the current re-sampling rate based on a Welch confidence interval test [32]. Under this scheme individuals are re-sampled a minimum number of times to calculate a value for mean and standard deviation of an assumed underlying normal distribution. A Welch test is then applied to these parameters and if a significant difference (to a user-specified level) is not found then the individuals are re-sampled. This re-sampling is repeated until a significant difference is detected or a predefined maximum number of samples is reached. Cantu-Pazcite [25] used a one-sided t-test to decide between two candidate solutions in a tournament. After the initial sampling, the solution with the highest variance is re-sampled one more time and the t-test is run to check for a significant statistical difference. As with Confidence-based DS this sampling is repeated until the desired p-value is reached or the maximum sample number is reached. Other work [23], [24], determines the number of re-samples using the difference between the solutions' fitness value δ, the environmental noise σ and known properties of Gaussian distributions. In Gaussian distributions, 95.4% of samples fall in an interval whose width is 4σ. In case the distance between the two solutions is greater than 2σ, it is likely that the fitter solution is truly the best performing in the tournament. On the other hand, having a small δ means the two distributions overlap and therefore more samples are required. This is done by calculating the normalised measure of the overlap v and the critical value of Gaussian distribution with a confidence level of 0.95.

Di Pietro et al. [26], [27] used the Standard Error (SE) to decide between solutions by re-sampling until SE estimates converged. These estimates are then used to characterise noise which is then used, to determine the re-sampling rate.More

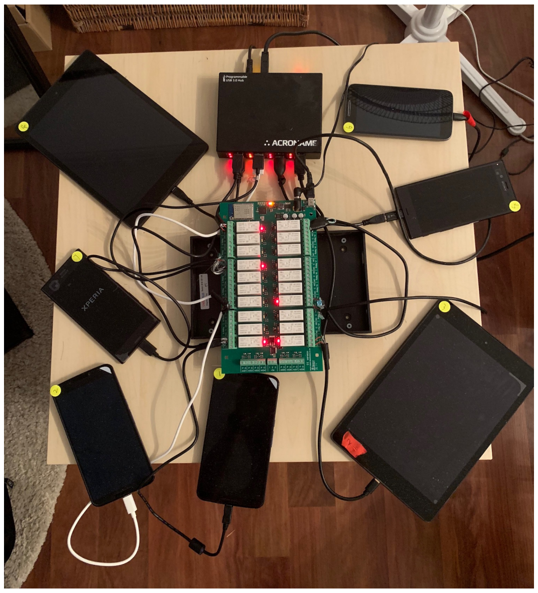

Fig. 2: Phone farm in action, multiple experiments running in parallel. The relay in the centre is responsible for physically disconnecting the phones from the charger. Screens are off to minimise noise from the screen's consumption, and wake-locks are used to prevent the from entering various standby modes.

broadly, re-sampling has also been applied to multi-objective optimisation problems [20], [21]. For more works on handling noise in optimisation, we refer interested readers to [28], [30].

The main shortcoming of the static and dynamic re-sampling techniques described above is the assumption that the noise is normally distributed. As can be seen from Fig. 1 this clearly is not the case due to systematic noise and and a (mostly) upward drift of power consumption for the same variants as optimisation progresses. Moreover, even when variants are freshly compared in a tournament, at the same point of optimisation process, we *still* find that the distributions of sampled energy use are not normally distributed when using the Kolmogorov-Smirnov and Jarque-Bera normality tests. This lack of normality in noise distributions introduces the risk that existing static and dynamic resampling strategies will not perform as expected in this setting.

III. Phone farm for energy measurements

One objective of our overall research project was to build a framework that can help researchers to measure mobile phone's energy consumption easily and accurately. Figure 2 shows the hardware setup of our framework in action. In the following, we present the high-level design of the framework. We begin with an outline of how the framework can be applied from a user's perspective. After that, we briefly outline how we control the hardware and which hardware we use in our study.

A. Operation

Our framework supports a number of modes of operation. In the most complex mode, the user can interact with the system as follows:

1) A user issues multiple commands to measure multiple applications in multiple devices.
2) The framework queues and runs these applications in the devices and records the energy consumption using the internal meters.
3) The framework gets back all results from these devices.

A core function of the framework that can be identified here is that it needs to be able to run an application, record the energy consumption and get back the result in a particular device.

In our framework, this is implemented as follows. In order for successful communication to happen, there are two services running in the PC and devices respectively, and a communication protocol for the two services. The service running in the PC is called "the client" and the service running in devices is called "the control service". These two services, along with the "app under test", consist of three main components of the framework. In general, the client is responsible for interacting with users and communicating with the control service; the control service is responsible for measuring energy consumption of the app under test and communicating with the client.

B. Controlling Android Devices

In order to reduce the noise from the system during the measurement of energy consumption, it is required that some aspects of Android devices be controlled. The aspects include charging, CPU frequency, screen brightness, network connection, air-plane mode etc. Control over these aspects is an important part of the framework, especially control over charging. To achieve as much control as possible, we follow the recommendations developed by Bokhari et al. [33], [34].

Control over charging is also essential for restoring battery capacity automatically. Therefore, whether control over charging can be achieved is a key factor that determines if the whole measurement process can be automated. In our implementation, we use the Wifi relay WIFI8020 from Devantech Limited [35] to physically disconnect the phones from the charging source. This has the advantage that both the client and the device can control this by sending simple `HTTP GET` requests.

C. Smartphones used

We use four different models of Android smartphones in our experiments, ranging from entry-level to top-of-the-class. The operating systems that we consider are Android 6, 7, and 8. In combination, these platforms cover 76.1% of Android's worldwide market share in December 2018.[1] We list all devices in Table I.

The devices vary significantly in their specifications. As we shall see later, this contributes – together with some "quirks" specific to certain OS versions – to an unexpectedly wide range of behaviours, even of the very same test application.

[1]Statcounter: Mobile & Tablet Android Version Market Share Worldwide, http://gs.statcounter.com/os-version-market-share/android/mobile-tablet/worldwide, visited on 25 January 2019.

Parameter name	Android version	SoC	CPU	Memory
Sony Xperia ZX	7 & 8	Qualcomm MSM8996 Snapdragon 820	Quad-core (2x2.15 GHz Kryo & 2x1.6 GHz Kryo)	3 GB
Nexus 6	6 & 7	Qualcomm APQ8084 Snapdragon 805	Quad-core 2.7 GHz Krait 450	3 GB
Moto G4 Play	6 & 7	Qualcomm MSM8916 Snapdragon 410	Quad-core 1.2 GHz Cortex-A53	2 GB
Nexus 9	6 & 7	Nvidia Tegra K1	Dual-core 2.3 GHz Denver	2 GB

TABLE I: The list of the used devices and their specifications.

To facilitate readability, we will from now on refer to phone-OS combinations just as “phones” or “devices”, unless we want to explicitly highlight a particular combination.

IV. Case Studys

In our case study, we first characterise our devices for the use in tournaments for which we seek statistical significance. Then, we extract recommendations from our measurements which are then used in a proof-of-concept of on-device optimisation and in comparison with other methods for noisy environments.

A. Characterisation of Phones

The goal of this first part of the case study is to characterise an understandable base case in order to establish which phone-OS combinations we might be able to use later for optimisation purposes.

The experimental design is influenced by the internal meters in the Nexus 6 and Nexus 9, which have a temporal resolution of 4Hz. Our goal is to investigate first whether it is possible to distinguish between processes that are identical in nature but slightly different in duration.

For this, we use a simple dummy load that we call BusyLoop app, as it just runs a busy loop for a pre-defined duration. The durations chosen are 10 seconds as the base case, 10.25 seconds and 10.50 seconds (being equivalent to the base case plus 1 or 2 samples of the 4Hz-energy meter), and 12 seconds. For each data collection, we start with a full battery (denoted as 100%) and repeatedly run BusyLoop until the battery has only 10% charge left.

We have to highlight one important detail of the implementation before we can continue. As shown in the motivating example in Section 1, the system can exhibit different energy consumption behaviours even when the only change in system state is a battery recharge. Our approach to mitigate this is as follows. To average out the effect of different states between different recharges and of varying rates of drift, we interleave the execution of different BusyLoop variants. This means, instead of running (10s, 10s, 10s, 10s, ...) in the first run (from 100% battery down to 10%), then running (10.25, 10.25s, 10.25s, ...) and so on, we rotate through the four configurations: first run (10s, 10.25s, 10.5s, 12s, 10s, 10.25s, ...), second run (10.25s, 10.5s, 12s, 10s, 10.25s, ...), third run (10.5s, 12s, 10s, 10.25s, ...) and so on. Then, by pulling apart and reassembling the actual measurements, we have exposed the measurement of each of the four configurations to the effects of different states as well as to varying drift speeds.

Based on these measurements, we perform two analyses. First, we simulate tournaments of varying length in which we

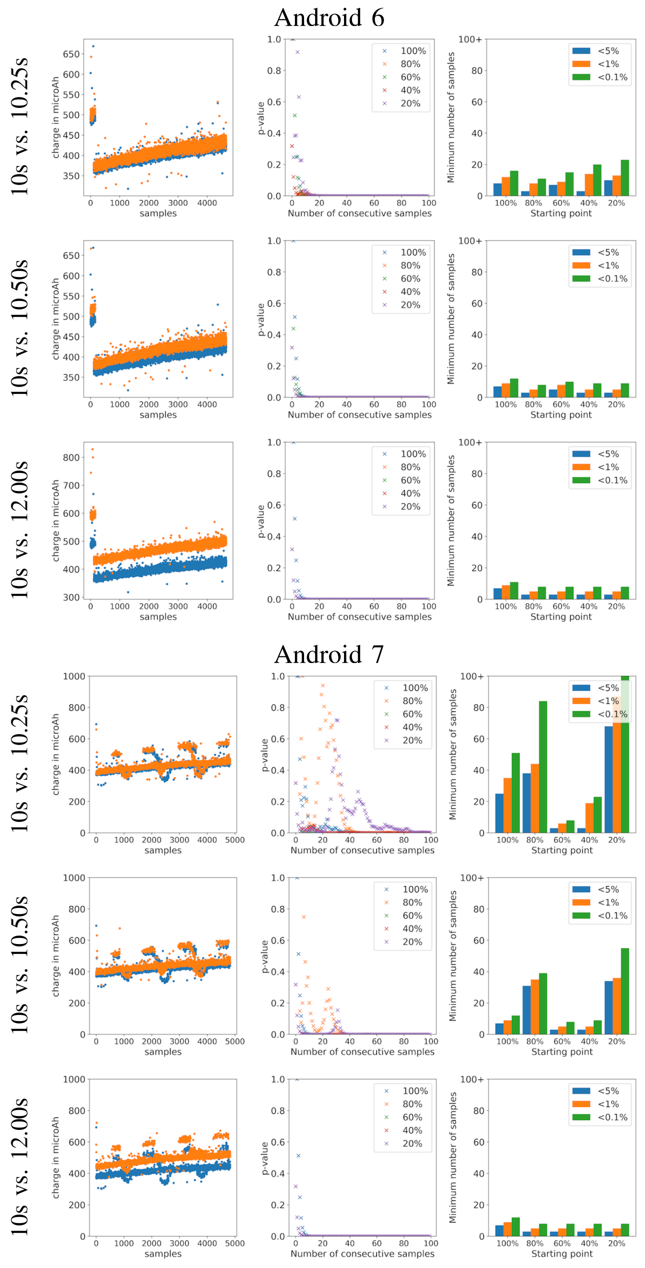

Fig. 3: Power use of repeated runs of BusyLoop app on Moto G4 Play. Blue: 10s, orange: as listed.

are attempting to discriminate between variants, to a given level of statistical confidence, using the Wilcoxon ranksum test. Through this process we are able to estimate the minimum number of samples needed on a device to discriminate between two variants to a given confidence level. The use of the

Wilcoxon ranksum test is important – this test does not assume normality in the underlying distribution.

As an example, we demonstrate the above process here using the Moto G4 Play. The raw results of running the four configuration are shown in the left column of Figure 3. It is interesting that the upgrade to Android 7 removed the initial phase of high energy consumption. However, additional higher and lower levels of energy consumption were introduced.

The middle column of Figure 3 shows, as scatter-plots, the results of the Wilcoxon ranksum tests, simulated at different points in time[2], and for various numbers of consecutive samples (from 1 to 100). As expected, the outcomes become statistically more and more significant (p-values decreases) as the number of samples increases. Similarly, we can see that we need more samples to determine a significant difference if the difference in duration is small (i.e., in the case of 10s vs. 10.25s).

Lastly, the rightmost column of Figure 3 attempts to extract an initial recommendation from the results of the statistical tests. These bar charts show, for each level of battery charge, and for each significance level the least number of samples needed from which on no test has returned a higher p-value. This conservative estimate can be used to inform experimental settings, for example, in regression testing of software and also when running tournaments in a evolutionary process.

For the other three devices, the observations are as follows. The Nexus 6 and the Nexus 9 (Appendix: Figures 6 and 7) largely follow the trends observed for the Moto G4 Play, but they are missing the additional states that were introduced with Android 7. For the Nexus 6, we can see short burn-in phases with the energy consumption being roughly twice as high, and we observed this independent of the experiment and of the operating system version. For the Nexus 9, these burn-in phases are just a few samples long, but result in energy consumption observation that are a factor of seven larger than subsequent ones.

The Sony XZ (Appendix: Figure 8), behaves quite differently. Although we employed the very same steps to attempt to control the system, we can observe at least six different levels of energy consumption of the system – despite our test application being just a simple busy loop. Essentially, we cannot make statements about statistical significance when the runtime difference is less than 5%, even when repeating a 10 second experiment 100 times.

B. Extracting Recommendations

Table II shows the results of running the BusyLoop variants on a Moto G4 Play running Android 7 with a full battery (i.e., starting at 100%) and significance level 5%. For each combination of the four variants, the table lists the minimum number of samples needed to distinguish two variants at the desired confidence level. This shows that the smaller the difference between variants the more samples are required.

[2]expressed by the percentage of battery charge used: 100% (full battery), 80%, 60%, 40%, 20% (nearly depleted battery)

	BusyLoop duration			
	10s	10.25s	10.50s	12s
10s		25	7	7
10.25s	25		8	7
10.50s	7	8		3
12s	7	7	3	

TABLE II: Comparing the four BusyLoop variants in pairs with starting point 100% (fully charged) and significance level 5.0%.

In this study, for each significance level of 5%, 1% and 0.1% and each battery level range, a 4-by-4 table is generated for each device-OS combination.

Having large numbers of re-samples such as 25 is expensive in real-world applications and can slow down the search process. Therefore, we minimise the number of samples further by taken the *median* of required samples taken from the variants at each battery level range for each significance level. The choice of the median is based on the assumption that the variants sampled at a given battery level are in some sense representative of the population that will be seen during optimisation. If the sampled variants are representative then the re-sampling estimate will be high enough approximately 50% of the time. Ultimately, the relative rank of the variant chosen to determine the re-sampling will need to be tested and calibrated empirically in optimisation experiments. Table III shows the median of the minimum number of samples needed for the desired significance level at every battery range. Each element in the table represents the median of samples found in the 4-by-4 corresponding table. For example, when running a tournament between two variants in battery range 100% to 80% and confidence level of 5% is required, it is recommended to sample each solution 7 times. As can be seen, 7 is the median of the element in Table II.

		Android 7				
		starting point				
		100%	80%	60%	40%	20%
significance level	5%	4.0	3.0	3.5	3.0	3.0
	1%	9.0	5.0	5.5	5.0	5.0
	0.1%	12.5	8.5	8.5	8.0	8.0

TABLE III: Recommended number of samples based on the BusyLoop case study. Set up is the Moto G4 running Android 7. Entries are the recommended minimum number of samples for different battery charge levels points and the desired significance level.

V. Optimisation Experiment

In this section we report on an optimisation experiment that serves as a proof-of-concept. We use an adaptive 1+1 Evolutionary Strategy (ES) algorithm, the mutation operator is uncorrelated mutation with one step size described in [36]. For comparison, we run optimisation with our recommendation-table based dynamic sampling (*table-based DS*) and compare to the re-sampling techniques described in [22] (*confidence-based DS*) and [23] (*2-sigma DS*) described in Section II). For our *table-based DS*, we accept the newly created solution if the statistical test either favours it or if no significant difference to the parent can be determined (based on the number of samples given in the lookup table).

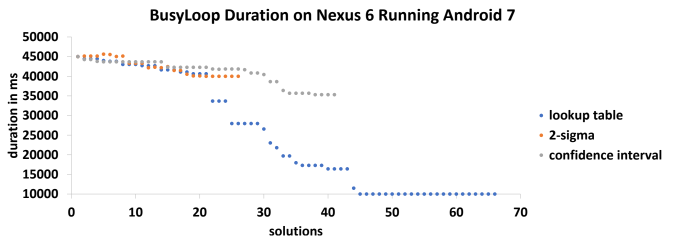

Fig. 4: Solutions' duration of busy loop optimisation experiment, shown are tournament winners. The duration equals the value of the decision variable.

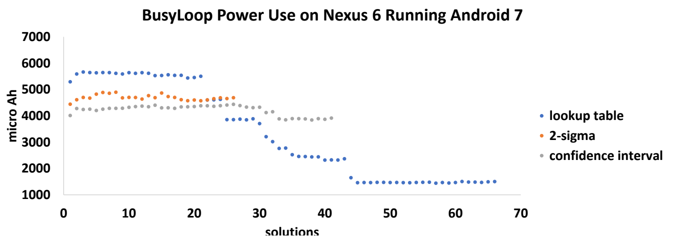

Fig. 5: Solution fitness values of busy loop optimisation experiment. The measurements are the ones returned by the internal meter for the solutions shown in Figure 4 and thus exhibit a slight upwards trend due to sensor drift.

We used our framework described in Section III to run the optimisation experiments on a Nexus 6 running Android 7. We generated the table prescribing re-sampling rates using the process described in Section IV-B. The experimental runs took approximately 4 hours and were started with a full battery and we stopped when reaching a battery level of 10%. The target for optimisation was the busy-loop app – a very simple benchmark where the current best loop iteration count is mutated according to a normal distribution with $\sigma = 0.25$ seconds. The duration of the starting variant was 45 seconds. It is important to note that this experiment is designed to explore the feasibility of different re-sampling approaches in this optimisation environment, rather than to definitively compare re-sampling techniques (which would require a larger study).

Figures 4 and 5 illustrate the results of applying each re-sampling technique to optimisation of the busy-loop benchmark for equivalent four-hour runs. As can be seen, the *table-based DS* technique generated considerably more variants than *2-sigma DS* and *confidence-based DS* techniques. The *2-sigma DS* re-sampled solutions 18 times on average which considerably slowed its progress. In fact, since the distribution of the generated solutions were not normal and strongly overlapped, *2-sigma DS* unrealistically requested up-to 4639 samples per solution in one of the tournaments. However, the upper limit is set to 30 samples per solution [23]. This shows that demanding the difference between two solutions to be at least 2σ to tell them apart is costly in this environment. For *confidence-based DS*, solutions were sampled 5 times on average which is the upper limit of the algorithm [22]. Again, *confidence-based DS* assumes that samples are taken from a normal distribution, a condition not met in this setting.

In contrast, the number of samples were relatively small in *table-based DS* technique at most 4 samples per solution across the entire optimisation, which, in turn, gave the search process enough time to evaluate more solutions. In addition, the use of more conservative statistical test (i.e. Wilcoxon ranksum) helped distinguishing between solutions. It should be noted that, especially, toward the end of the optimisation process, *table-based DS* produced some large outliers in both execution time and energy consumption. The presence of these outliers, being over-estimates, did not impact on optimisation. However, more experiments in different settings would be needed to verify if other optimisations are similarly robust to such outliers.

VI. Conclusions and Future Work

In this paper we have examined re-sampling in tournaments in the context of in-vivo optimisation of energy use on mobile platforms. We have derived distributions of the number of samples required in two-way tournaments on a variety of mobile platforms. We have developed a process for generating preliminary recommendations for re-sampling strategies for each platform and we have applied these recommendations in a proof-of-concept experiment with promising results.

Future work will focus on carefully measuring the relationship between re-sampling recommendations and optimisation settings across a variety of devices. We also propose to examine how changes in platform state impact on the effectiveness of re-sampling recommendations.

References

[1] R. Saborido, V. V. Arnaoudova, G. Beltrame, F. Khomh, and G. Antoniol, "On the impact of sampling frequency on software energy measurements," PeerJ PrePrints, Tech. Rep., 2015.

[2] R. Morales, R. Saborido, F. Khomh, F. Chicano, and G. Antoniol, "Earmo: An energy-aware refactoring approach for mobile apps," *IEEE Transactions on Software Engineering*, Dec 2018.

[3] L. Zhong and N. K. Jha, "Graphical user interface energy characterization for handheld computers," in *Int. Conf. on Compilers, architecture and synthesis for embedded systems*. ACM, 2003, pp. 232–242.

[4] B. R. Bruce, J. Petke, M. Harman, and E. T. Barr, "Approximate oracles and synergy in software energy search spaces," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018, accepted.

[5] A. Hindle, "Green software engineering: The curse of methodology," in *IEEE 23rd Int. Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, March 2016, pp. 46–55.

[6] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "Devscope: A nonintrusive and online power analysis tool for smartphone hardware components," in *Eighth Int. Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2012.

[7] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Hardware/Software Codesign and System Synthesis*. ACM, 2010, pp. 105–114.

[8] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *European Conference on Computer Systems*. ACM, 2011.

[9] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphones using kernel activity monitoring," in *USENIX Conference on Annual Technical Conference*. USENIX Association, 2012.

[10] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Int. Symp. on Software Testing and Analysis (ISSTA)*. ACM, 2013, pp. 78–89.

[11] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Int. Conf. on Software Engineering (ICSE)*. IEEE Press, 2013, pp. 92–101.

[12] T. McDonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the android ecosystem," in *IEEE Int. Conference on Software Maintenance (ICSM)*. IEEE Press, 2013, pp. 70–79.

[13] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 39, 2016.

[14] M. A. Bokhari, Y. Xia, B. Zhou, B. Alexander, and M. Wagner, "Validation of internal meters of mobile android devices," *CoRR*, vol. abs/1701.07095, 2017.

[15] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Mobile Systems, Applications, and Services*. ACM, 2011, pp. 335–348.

[16] Google, "Batterymanager," https://developer.android.com/reference/android/os/BatteryManager, accessed on 15 October 2018.

[17] F. Siegmund, A. H. C. Ng, and K. Deb, "Hybrid dynamic resampling for guided evolutionary multi-objective optimization," in *Evolutionary Multi-Criterion Optimization*. Springer, 2015, pp. 366–380.

[18] ——, "A ranking and selection strategy for preference-based evolutionary multi-objective optimization of variable-noise problems," in *IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 3035–3044.

[19] Z. Zhang and T. Xin, "Immune algorithm with adaptive sampling in noisy environments and its application to stochastic optimization problems," *IEEE Computational Intelligence Magazine*, vol. 2, no. 4, pp. 29–40, Nov 2007.

[20] T. Park and K. R. Ryu, "Accumulative sampling for noisy evolutionary multi-objective optimization," in *Genetic and Evolutionary Computation Conference*. ACM, 2011, pp. 793–800.

[21] D. Buche, P. Stoll, R. Dornberger, and P. Koumoutsakos, "Multiobjective evolutionary algorithm for the optimization of noisy combustion processes," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 4, pp. 460–473, 2002.

[22] A. Syberfeldt, A. Ng, R. I. John, and P. Moore, "Evolutionary optimisation of noisy multi-objective problems using confidence-based dynamic resampling," *European Journal of Operational Research*, vol. 204, no. 3, pp. 533 – 544, 2010.

[23] G. Iacca, F. Neri, and E. Mininno, "Noise analysis compact differential evolution," *Systems Science*, vol. 43, no. 7, pp. 1248–1267, 2012.

[24] E. Mininno and F. Neri, "A memetic differential evolution approach in noisy optimization," *Memetic Computing*, vol. 2, no. 2, pp. 111–135, Jun 2010.

[25] E. Cantú-Paz, "Adaptive sampling for noisy problems," in *Genetic and Evolutionary Computation Conference*. ACM, 2004, pp. 947–958.

[26] A. Di Pietro, *Optimising evolutionary strategies for problems with varying noise strength*, 2007.

[27] A. D. Pietro, L. While, and L. Barone, "Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions," in *IEEE Congress on Evolutionary Computation*, vol. 2, 2004, pp. 1254–1261 Vol.2.

[28] F. Siegmund, A. H. C. Ng, and K. Deb, "A comparative study of dynamic resampling strategies for guided evolutionary multi-objective optimization," in *IEEE Congress on Evolutionary Computation*, 2013, pp. 1826–1835.

[29] S. O. Haraldsson and J. R. Woodward, "Genetic improvement of energy usage is only as reliable as the measurements are accurate," in *Genetic and Evolutionary Computation Companion*. ACM, 2015, pp. 821–822.

[30] P. Rakshit and A. Konar, *Principles in Noisy Optimization: Applied to Multi-agent Coordination*. Springer, 2018.

[31] M. Bokhari and M. Wagner, "Optimising energy consumption heuristically on android mobile phones," in *Genetic and Evolutionary Computation Conference Companion*. ACM, 2016, pp. 1139–1140.

[32] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis*. McGraw-Hill New York, 1991, vol. 2.

[33] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner, "Deep Parameter Optimisation on Android Smartphones for Energy Minimisation: A Tale of Woe and a Proof-of-concept," in *Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 1501–1508.

[34] M. A. Bokhari, B. Alexander, and M. Wagner, "In-vivo and offline optimisation of energy use in the presence of small energy signals: A case study on a popular android library," in *15th EAI Int. Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. ACM, 2018, pp. 207–215.

[35] Devantech Limited, "WIFI8020 - 20 x 16A WIFI relay," http://www.robot-electronics.co.uk/wifi8020-20-x-16a-relay-module.html, 2018, accessed on 15 October 2018.

[36] A. Eiben and J. Smith, "Introduction to evolutionary computing (natural computing series)," 2008.

Appendix

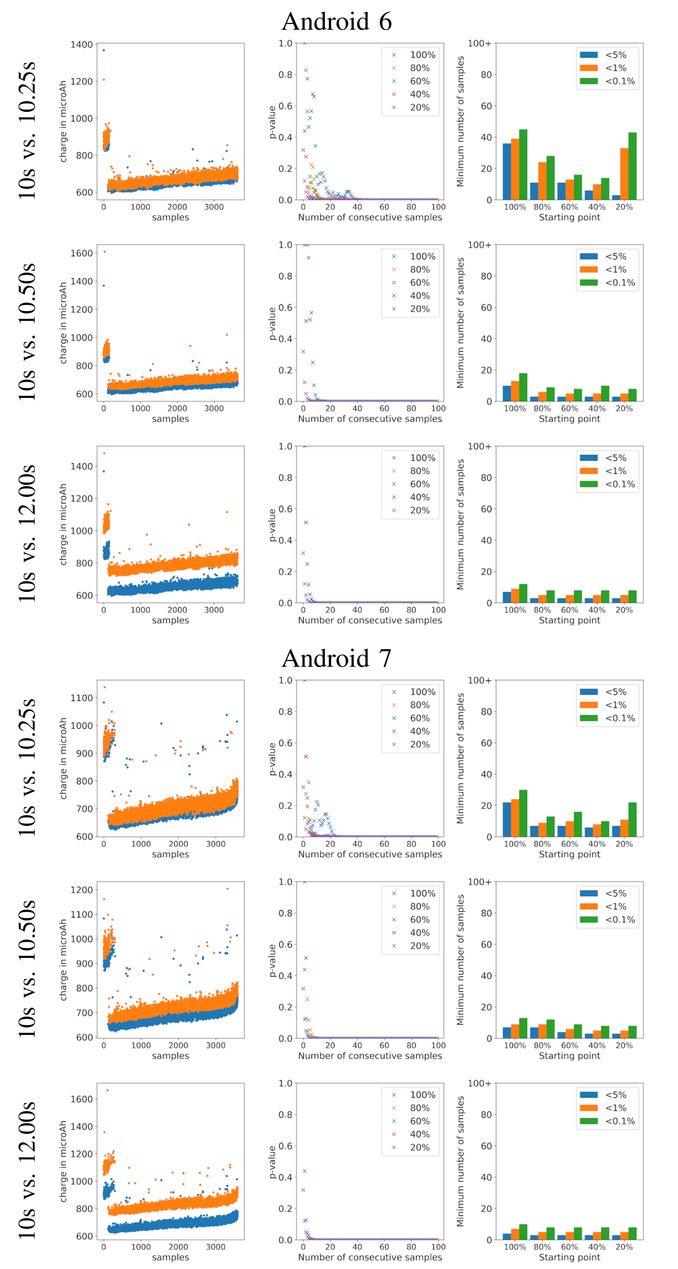

Fig. 6: Power use of repeated runs of BusyLoop app on Nexus 6 running Android 6/7. Blue: 10s, orange: as listed.

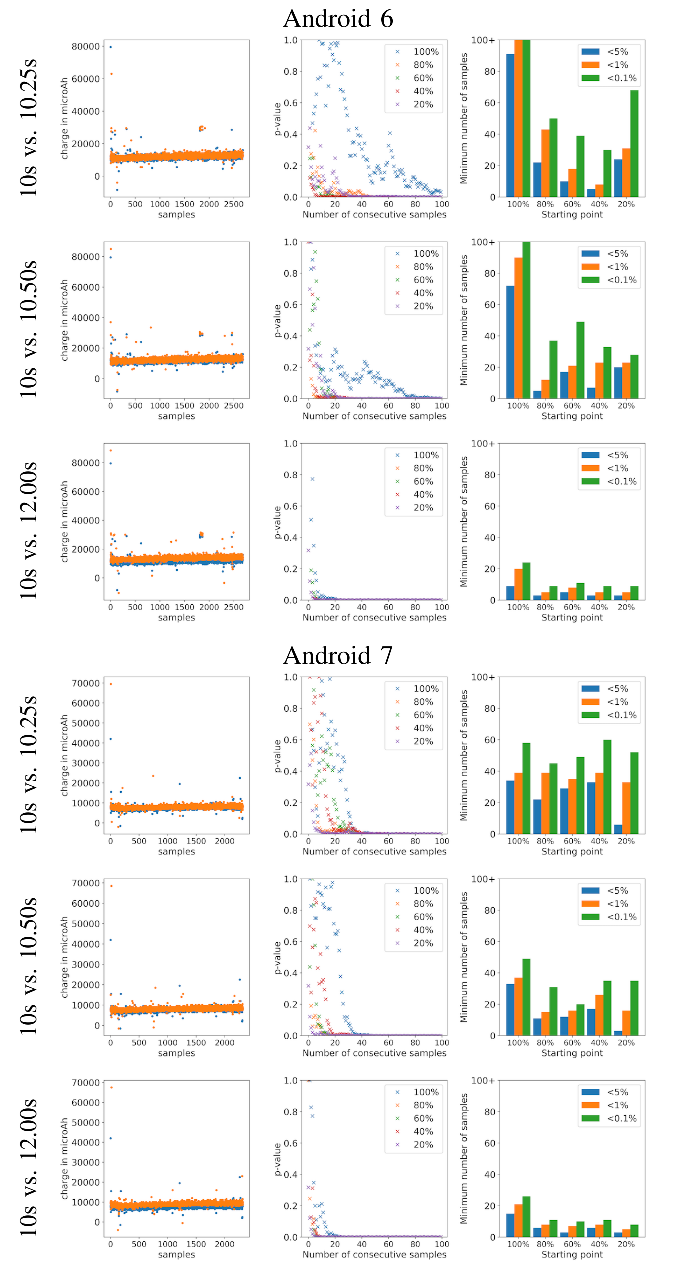

Fig. 7: Power use of repeated runs of BusyLoop app on Nexus 9 running Android 6/7. Blue: 10s, orange: as listed.

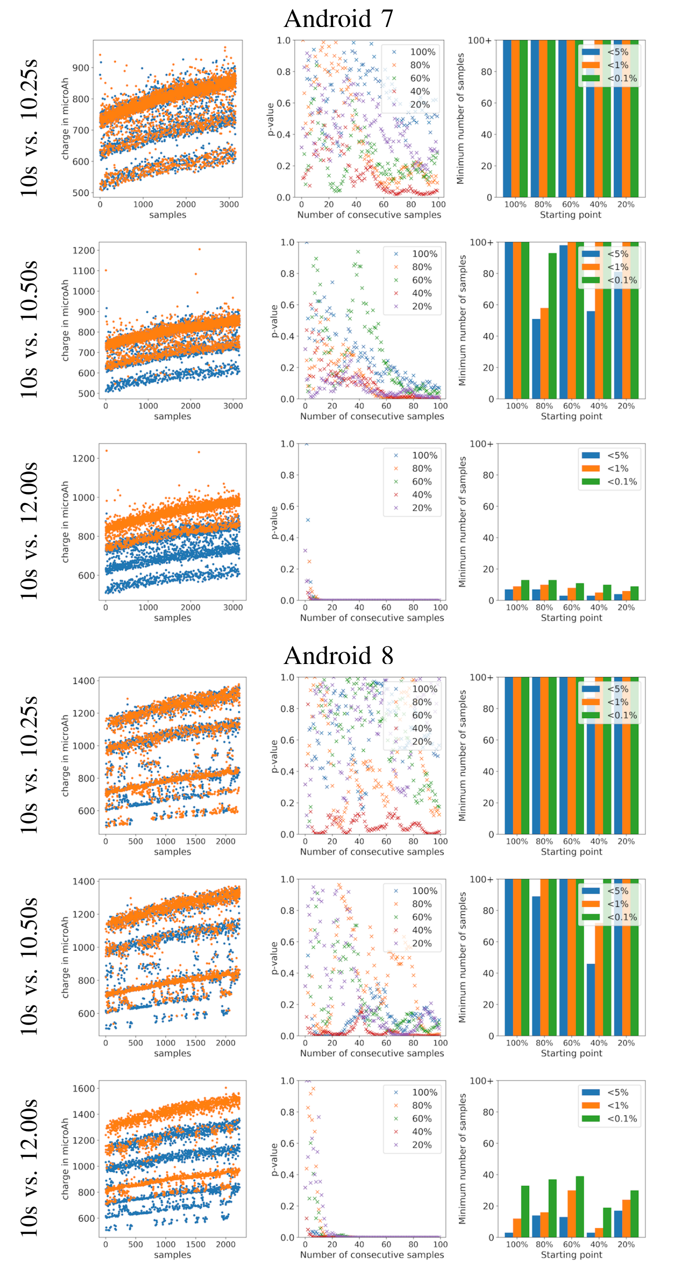

Fig. 8: Power use of repeated runs of the BusyLoop app on Sony XZ running Android 7/8. Blue: 10s, orange: as listed.