Empirical Software Engineering (2022) 27: 155
https://doi.org/10.1007/s10664-022-10188-5

A Hybrid Distributed EA Approach for Energy Optimisation on Smartphones

Mahmoud A. Bokhari[1] · **Bradley Alexander**[2]

Accepted: 26 May 2022 / Published online: 6 August 2022

Abstract

For many people, mobile platforms are now an essential part of everyday life. A defining feature of mobile platforms is their reliance on battery performance. Due to this reliance, there is a pressing need for mobile applications that minimise their own impact on batteries. While mobile platforms are improving their capabilities in terms of policing the energy use of applications and rationing energy-hungry devices, mobile application developers still lack knowledge in how to write energy efficient programs. Recent work in automatic program improvement using heuristic search over randomly generated program variants has shown some promise in terms of producing reductions in programs' energy-use. A challenge in this work is accurately measuring the energy-use of program variants. One approach to measurement is to use each platform's internal meter to assess variants on the device itself. This approach has advantages in terms of measuring actual energy-use on each platform but is not ideal for the search for program variants that perform well across multiple platforms. The work in this paper addresses this problem by using an island-like evolutionary search mode to simultaneously evolve variants on multiple platforms. Island models of evolutionary search conduct search on multiple platforms in parallel and share promising variants. The results show that this approach has advantages over isolated evolution in terms of speeding up evolution on each platform and improving the efficiency of search. Validation results show that the island-inspired model is able to evolve variants with good cross-platform performance. In addition, it evolves a solution that outperforms best found solutions using a sequential evolutionary algorithm on it is native platform with an effect size greater than 90%.

Communicated by: Aldeida Aleti, Annibale Panichella and Shin Yoo

This article belongs to the Topical Collection: *Advances in Search-Based Software Engineering (SSBSE)*

✉ Mahmoud A. Bokhari
mabokhari@taibahu.edu.sa

Bradley Alexander
bradley.alexander@adelaide.edu.au

[1] Software Engineering Research Group, Computer Science Department, Taibah University, Medina, Kingdom of Saudi Arabia

[2] Optimisation and Logistics, School of Computer Science, University of Adelaide, Adelaide, Australia

Keywords Search-based software engineering · Genetic improvement of software · Distributed evolutionary algorithms · Islands model · Energy optimisation · Android

1 Introduction

Mobile computing platforms are ubiquitous and essential devices for everyday life. Though mobile platforms are becoming increasingly capable in terms of their processing capacity, they differ from other platforms in terms of their resource constraints, including memory, network cost/capacity, and, most pressingly, battery life (Ferreira et al. 2011). These constraints increase the importance of the non-functional properties of the software running on the platform, such as memory footprint, cpu-utilisation, network and GPS usage, and energy-use. Of these properties, energy use is key, partly because energy use is impacted by most other non-functional properties and also because energy use directly impacts how long the device can be used on a single battery charge.

Unfortunately, software developers for mobile platforms have very limited awareness of how to write programs that minimise energy use (Pressman 2005). One alternative approach is to use automated search to modify existing software to be more energy efficient (Bokhari et al. 2018; Bruce et al. 2019; Morales et al. 2018). One broad version of this approach is called Genetic Improvement (GI): it performs evolutionary search by generating variants of an existing program through random small random edits to the program and selecting the best variants for further exploration by measuring their performance. When using GI to improve energy use, a key challenge is obtaining a reliable proxy for the amount of energy used by a program variant. For mobile platforms, past work has used energy models (Bokhari et al. 2018; Linares-Vásquez et al. 2015; Morales et al. 2018), external meters(Bruce et al. 2019), and the platform's internal fuel-gauge (Bokhari et al. 2019). All of these measurement approaches have drawbacks. Models have issues in terms of fidelity,external meters are increasingly hard to install without impacting platform behaviour,and internal meters can be noisy. Layered on top of this underlying measurement problem is the increasingly complex and state-dependent behaviour of hardware/OS combinations in terms of energy use (Bokhari et al. 2020a). These factors make it difficult to reliably search for software variants that use less energy across platforms and across time.

One promising approach, which partially addresses the above problems, is to perform evolutionary search *in-vivo* on *each* platform using its own internal meter. Such platform-specific search, if done across a representative set of system states, can find a variant that at least performs well on its local platform. This type of search still carries the risk that the system states encountered during evolution were unrepresentative, and the evolved individual is biased by these system states. This risk can be mitigated by a careful approach to validation, which can compare evolved variants over a range of system states (Bokhari et al. 2020a) and pick only those that perform well across this range.

However, there are still drawbacks to this setup of *in-vivo* optimisation + validation. These are:

1. a very limited budget for evaluations, which limits search, and
2. the evolved program variants are specialised to each platform, which, in the worst case, means we have to maintain a software variant for each target platform.

The reason for 1. above is that noise in energy measurement and changing system states requires that we run our variants for an extended period to properly compare the energy use of variants (Bokhari et al. 2019). Another constraint is that system states can vary a lot between system reboots (Bokhari et al. 2020a; Kalibera et al. 2005) so evolutionary runs work best on a single battery charge. These factors mean that we are often limited to testing a few hundred variants during a search. For 2. above, we can limit the number of variants to maintain by validating variants across multiple platforms. However, because the variants are not evolved to work on multiple platforms there are limits to how well selection of good cross-platform variants during validation can work.

This work aims to address the issues above by evolving program variants on multiple platforms simultaneously and, periodically, sharing the best variants on each platform with others. Under this distributed evolutionary model (dEA), which shares characteristics with island models, copies of evolved variants are migrated to other platforms during evolution with the aim of both speeding up the evolutionary process and generating individuals that generalise better.

To test the dEA model, we conducted experimental runs to optimise the energy use of Facebook's Rebound Animation library [1] – which models spring dynamics for use in GUI applications. In our previous study (Bokhari et al. 2017a), we reviewed open-source mobile applications and libraries published in Github.com to find applications that: are widely used; compilable under one minute; have energy hogs; have a ready-to-use test suite that contains test cases that permit deviation from the test oracle. This is crucial to our experiments since we aim to evolve solutions that minimise energy consumption without drastically degrading the functional requirements of the app under evolution. We only found the Rebound library which meets our requirements. The main function of Rebound, which generates animations, uses a while-loop to implement spring animations using Hook's law. The computations include, among other, Euler integrations and calculations of derivatives.

Our experimental results from this approach show that:

- the dEA model speeds up the average time of convergence and improves the average local energy performance of evolved variants during evolution compared to isolated evolution of individuals;
- there is, on average, a sustained improvement in program performance, in most cases, on the acceptance of migrants from other platforms;
- there are variations in the rate at which migrants are accepted between devices;
- there is a considerable gap between the measured performance of individual variants during evolution and during validation – though there are variants produced by the island model that generalise well across platforms.
- the dEA model is able to find a robust solution that have a large effect size (90%+) compared to the best found solutions using sequential EA (isolated model) on its native platform. In addition, none of the isolated model's champion outperform this robust solution on their platforms.

The rest of this paper is laid out as follows. The next section (Section 2) describes related work in energy measurements techniques, and the optimisation of non-functional properties of software on mobile platforms. Section 3 describes the dEA algorithm and the motivations for its design in detail. Section 4 describes the experimental setup and results. Section 5

[1]Rebound Spring Animations for Android: https://github.com/facebookarchive/rebound-js, Accessed April 2021.

presents our validation results, followed by Section 6 which reports the threats of validity. Finally, Section 7 presents conclusions and future work.

2 Related Work

This section starts by presenting energy measurement techniques. It then describes work in the optimisation of non-functional properties of software on mobile platforms.

2.1 Energy Measurements Techniques

To measure the energy consumption of smart devices, one can use external meters such as the Monsoon device, internal meters and energy profilers. Each method has its advantages and disadvantages. For example, although external meters provide accurate readings and can bypass the battery which can save considerable time in our experiments, they are expensive and require specialised technical skills. This is because practitioners need to open the device and attach the meter leads to the battery or to the device board (Cruz and Abreu 2021; Hoque et al. 2016). Therefore, external meters cannot be a feasible solution to many mobile app developers.

Energy profilers that are based on energy models can be used to obtain noise-free readings. There are several methods for creating such models. For example, utilisation-based models can estimate the energy consumption by relating software and HW counters to enrgy use (Shye et al. 2009; Murmuria et al. 2012). Other works extend this methodology to integrate the different HW states and system calls into the model creation process (Pathak et al. 2011). The last category relates the executed line of codes (LOC) to energy (Li et al. 2013; Hao et al. 2013)

The conundrum here is which model creation solution to use? Although energy models are convenient and more accessible to practitioners, they are less accurate compared to HW meters. Utilisation-based models fail to capture all factors that contribute to eneregy usage. For instance, these models do not take into account the different states of network interface controller during data transmission (Pathak et al. 2011). State-based and system call models are generally infeasible solutions because HW manufacturers do not publish the relavent information about their products(Pathak et al. 2011). While LOC-based models are conceptually simple, they are difficult to maintain. This is owing to the thousands APIs in the Android SDL. Furthurmore, these APIs evolve rapidly at rate of 115 API updates per month (McDonnell et al. 2013). Moreover, the energy consumption of software is platform dependent and varies in complex, state-dependent, and individual-device-dependent ways that are challenging to model (Bokhari et al. 2020a; Hoque et al. 2016; Hindle 2016)

To alleviate these issues, we utilise the internal meter of smart devices and compare the energy consumption of software variants in-vivo. Unlike external meters, internal fuel gauges are easy to use and accessible to practitioners. Even though internal meters are less accurate than external meters, they can be precise (Dong and Zhong 2011; Bokhari et al. 2017b). Therefore, they can be an adequate alternative to obtain real measurements for optimisation contexts, if the measurement periods are sufficiently long (Langdon et al. 2016; Bokhari et al. 2017b). In addition, internal meters do not suffer from the inherent inaccuracy of models as they can show what a software variant actually uses at a particular moment. Furthermore, they do not abstract away the behaviour of the hosting platforms.

2.2 Optimisation

The optimisation of non-functional properties of software is of growing importance in mobile computing. Researchers and software practitioners have developed sound approaches to enhance each aspect of software non-functional requirements. For example, tasks offloading, pre-loading and code refactoring using heuristic-based methods have been utilised to increase applications responsiveness, decrease software runtime and memory consumption. For a comprehensive literature review on optimising the non-functional properties of mobile applications, we refer the interested reader to (Hort et al. 2021).

There are several studies that use a broad range of techniques from software engineering and GI to optimise the energy consumption of software on desktop platforms (Bruce et al. 2015; Schulte et al. 2014; Burles et al. 2015). These studies focus on the single-objective of reducing energy consumption and estimate energy use by the means of energy models built using external meters. The use of such models, in contrast to our current work, allows for a large number of evaluations, however, it has been shown that energy models can be misleading on mobile platforms (Bokhari et al. 2019; Bokhari et al. 2020a).

In terms of work targeting portable platforms, Bruce et al. (2019) applied a multi-objective optimisation technique trading-off program accuracy with energy use (measured by an external meter) on a Raspberry Pi. In addition, Linares-Vásquez et al. (2015) built energy models of OLED energy use on Android devices and used to optimise the energy consumption of applications' GUIs. Bokhari and Wagner (2016) proposed a framework for optimising the energy use of smart devices by finding an optimal set of features to switch off depending on users behaviours, whereas Pascual et al. (2015) opted to adaptively generate device configurations (e.g., WiFi and GPS frequency strength) of an Android application. Additionally, Bokhari et al. (2019) proposed a conceptual framework to determine the minimum number of samples required to show a statistically significant difference when comparing two solutions in a tournament during the evolutionary process.

There are studies in the literature that discuss the impact of the system states on measured software performance. For example, software execution time can be greatly affected by OS environment variables (Mytkowicz et al. 2009),the state of cache memory (Alameldeen and Wood 2003),thread context switches (Pusukuri et al. 2012),JIT compilation settings (Georges et al. 2008; Sachindran and Moss 2003),and garbage collection (Huang et al. 2004; Blackburn et al. 2006).

There is also work in the search-based software engineering literature that reports system state impacting the optimisation process. For example, Burles et al. (2015) and Brownlee et al. (2017) alleviated the problem of changing system states interfering with individual evaluations by disabling the JIT and the garbage collection during the optimisation process, and re-enabling them during the validation experiment. Another technique proposed by Bruce et al. (2019) involves running the experiments immediately after a fresh system reboot.

Energy optimisation work on mobile platforms has, to date, focused either on cheap-to-run energy models with limited fidelity or on much more expensive *in-vivo* evaluations that are better at capturing the ground-truth of energy consumption on the platform. Our current work aims to improve the latter approach by allowing faster convergence of solutions and offering the prospect of evolving solutions with better cross-platform performance. The mechanism we use is a distributed algorithm dEA model for concurrent *in-vivo* evolution. We describe this model next.

3 Framework/Methodology

The following section describes our setup for evolution of reduced-energy program variants for mobile platforms. Our distributed EA (dEA) model is based loosely on the island model, where individual variants are periodically shared between different platforms that are otherwise performing optimisation locally and independently from each other. The heterogeneous nature of these platforms and the need to generate clean energy signals constrains the design of our framework. In fact, we deliberately exploit the heterogeneity of platforms to achieve our goal, whereas other island models typically only exploit the additional compute power and the potentially diversified population.

These design aspects are described in this section, starting with the motivation and overview of our approach, followed by descriptions of specific components of the evolutionary process.

3.1 Energy Measurement

Obtaining energy measurements from mobile devices comes with challenges that need to be addressed. Table 1 divides these challenges into expected and unexpected challenges. The former refers to challenges that are caused by the system behaviour and the battery fuel gauge. The used operating systems contain complex interaction between their software and HW components such as Bluetooth and Wi-Fi adaptors. Therefore, we deactivate all unused HW components (i.e. use airplane mode and turning off the screen) during experiments.

Sampling the battery stats every 250ms can induce some sampling error. This is because some noise can be introduced by gaps between the start and finishing time of the measured process (i.e. the framework), and the time the battery is sampled (i.e. battery fuel gauge). We minimise this jitter effect by the use of dummy-loops during the optimisation experiments. In the validation experiment we repeat each test case 1000 times to magnify the energy signal and to reduce the jitter effect. We will discuss each method in the following sections.

During setting-up the experiments, we encountered unanticipated challenges. Android OS enters the Doze-mode after switching the screen off and leaving the device unused for a certain amount of time depending on the manufacture setting. During the Doze-mode, the OS suspends the background processes which interfere with our experiments and data

Table 1 The list of the used devices and their specifications

Challenges		Solutions
Expected challenges	System behaviour	Airplane mode. Turn off the screen
	Sampling-induced error	Use dummy loops during optimisation. Repeat test execution during validation.
Unexpected challenges	Doze-mode	Use CPU wakelocks. Periodically turn on the screen.
	Android Debug Bridge	Drop and restore the connection.
	Temperature	Desk-fans.
	Process throttling	Use of a special CPU governor "userspace". Reduce CPU frequency on each device.

The stock ROM is used in each device

collection. To solve this issue we use a special APIs called CPU wakelocks which white-lists our framework's processes. However, this is only a partial solution, since other required processes get suspended, therefore, we periodically pause the running experiments, and turn on the screen to prevent the device from entering the Doze-mode.

In our framework, Android Debug Bridge (ADB) is used to control and communicate with the used devices. Deploying an app (i.e. compiling, transferring and installing it) can take a minute or more which is an expensive task in the context of GI. Moreover, app transfer and installation can fail due to ADB server instability. Failure modes includes the connected device going offline, interference with the communication port by other Android services, and ADB stops responding. To solve the problem of deployment time, we modified Rebound library to read its code parameter from a configuration file. We also use a programmable USB hub to automatically drop and restore the link to the device once it goes offline. In addition, the framework restarts ADB every time it freezes.

Running such experiments on mobile devices can raise their temperature notably. Consequently, more noise is induced in energy measurements. This is due to the positive relationship between energy usage and temperature (Peltonen et al. 2015). In addition, to protect the CPU, the CPU governor reduces the CPU frequency when its temperature raises above a predefined threshold. This indeed interferes with our controlled experiments. To overcome these issues, we use several cooling fans placed under the devices. In addition, we throttle the CPU speed using the CPU governor "userspace" on each device model. Using several experiments, we found 1.428 GHz on Nexus 9, 1.162 GHz on Nexus 6 and 1.152 GHz on Moto G Play allow the benchmark to run in a tolerable time-frame whilst minimising variability and temperature increase.

3.2 A Hybrid Distributed Approach

The aim of our search process it to find program variants that use less energy on mobile platforms. As mentioned earlier, problems with the use of models and external meters mean that it is desirable to make use of the internal battery meter (fuel-gauge) on each platform. Our proposed setup for the search process, dEA, evolves program variants simultaneously on a number of different mobile platforms and exchanges variants (immigrants) between platforms with the aim of both speeding up evolution and producing general program variants that use less energy across platforms.

There are a number of technical challenges that inform the design of the search setup in dEA. *In-vivo* optimisation requires that we evaluate variants live on each target platform using its own fuel gauge. Limits to the accuracy of the fuel gauge mean that we have to run each program variant for a substantial amount of time (Bokhari et al. 2020b). In addition, mobile platforms exhibit unpredictable, time-variant, system states that impact on background energy consumption (Bokhari et al. 2020a). These issues persist even when steps are taken to minimise the number of active devices on the platform. To cope with this variation dEA has to re-run tournaments to compare new variants to the current variant in the current system state. Tournaments have to run long enough to reliably rank variants and this, in turn limits the number of evaluations available in a given time.

Another constraint on dEA is the limited battery life of each platform. Earlier research has shown that the background energy consumption of a platform can change radically between re-charges (Bokhari et al. 2020a) and, thus, a recharge and re-start can stymie comparisons of variants in an evolutionary process. In order for evolution to take place in a more stable environment we are restricted to running during a single charge on each device. In our experiments, this constraint limits each device to between 200 and 1500 evaluations depending on platform, speed and efficiency and the efficiency of evolved program variants being run.

This limited evaluation budget precludes the use of large populations of variants and large edit-spaces in our search processes. To this end, dEA, uses a (1+1)EA. The (1+1)EA is an evolutionary algorithm which has a population of just one, where the variant in the population is replaced if the newly generated variant is better. In our experiments we also limit our space of potential edits to be the floating-point values of just five key variables in the target program.

Finally, the disparate platforms' evaluation budgets mean that evolution will run at different speeds. This means that, in order for each platform to contribute and receive comparable numbers of variants, the exchange of variants has to take place asynchronously. We achieve this by using a shared pool where variants are deposited and received. This setup shares some features with the pool model of distributed evolution (Gong et al. 2015). The triggering event for access to the pool is when a platform reaches a 10% interval in its battery life. Thus, each platform performs eight exchanges during evolution (albeit at different wall-clock intervals). The setup of dEA is shown in Fig. 1.

This concludes our broad description of the evolutionary process. Next, we describe the key components in detail including: the detail of the search algorithm, the solution representation, the fitness function, energy signal amplification and solution accuracy constraints.

3.3 Evolutionary Algorithm

Algorithm 1 shows the mechanism of evolving new solutions on each island. The process starts with the original, hand-coded program. Mutated variants are compared to the current variant program in a tournament and, if the new variant uses less energy, it replaces the

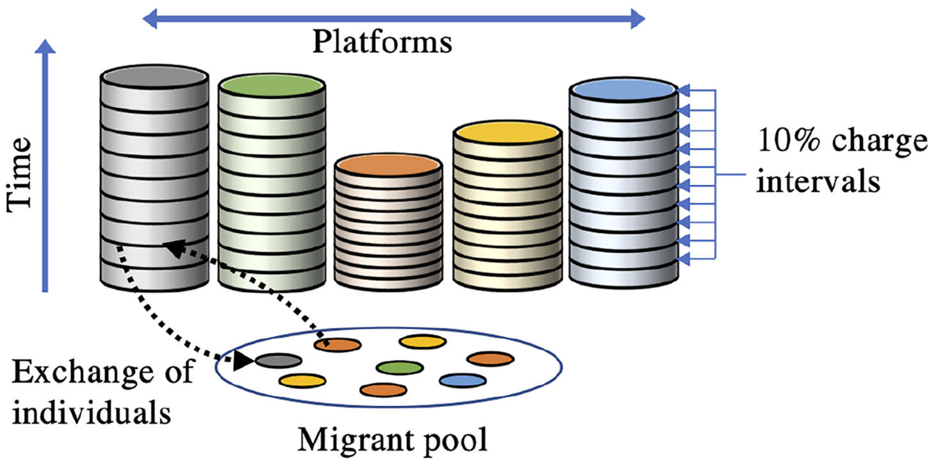

Fig. 1 An overview the dEA framework. Each platform executes a local (1+1)EA to evolve the parameters of a software variant. On each platform, at each 10% interval of battery life the current program variant is deposited to the migrant pool and, at the same step, a randomly selected variant (immigrant) from another platform is taken, without replacement, from the pool. The immigrant is immediately compared with the current program variant. If the immigrant uses less energy it becomes the current program variant on that platform. Note that different platforms have different battery capacities and run at different speeds so exchange of individuals via the pool is asynchronous.

current program. A simple (1+1)EA (Schwefel 1977) is used to create the solution (line 9) on each island. The mutation probability is set to $1/n$, where n is the number of decision parameters ($n = 5$ in these experiments, see below).

Algorithm 1 A pseudo- code for evolving new solutions on islands.

```
   input: batteryLevel, originalConfiguration, betteryLimit
1  currentBest ← originalConfiguration
2  currentBest.results ← evaluate(currentBest)
3  while batteryLevel ≥ betteryLimit do
4      if batteryLevel ≠ 100% ∧ batteryLevel%10 = 0 then
5          connectToServer(“exchange”)
6          send(currentBest)
7          newSolution = getForeigner()
8      else
9          newSolution ← createNextSolution(currentBest)
10     end
11     newSolution.results ← evaluate(newSolution)
12     if validateSolution(newSolution.results) then
13         if newSolution.results.fuel ≤ currentBest.results.fuel then
14             currentBest ← newSolution
15         end
16     end
17     batterLevel ← getBatteryLevel()
18 end
```

The distributed part of the algorithm is triggered when the battery interval reaches 10% of the battery life (lines 4 to 7). When this point is reached, the island pauses the evolutionary process. It then connects to a personal computer (PC) via the Android Debugging Bridge (ADB) to send a solution (the current best in our case) to the central pool residing on that PC.

Algorithm 1 shows the process of exchanging the current best solution on an island to the server (i.e. PC) and receiving a foreigner solution. The server randomly picks a solution from the pool of foreign immigrants without replacement and sends it to the connected island to evaluate. A platform only accepts the immigrant if the immigrant outperforms the current local solution. The use of a central PC as the medium of exchange, avoids, issues with direct communication between platforms including the energy cost of WiFi communications (Pathak et al. 2011). The low to moderate frequency of exchanges is selected to both minimise the overhead of having to connect by the ADB (Bokhari et al. 2017a) and to help maintain the diversity of solutions on the islands (Skolicki and De Jong 2005).

Algorithm 2 A pseudo- code for exchanging an immigrant.

```
  input: island
1 currentBatteryLevel ← island.getBatterLevel()
2 islandCurrentBest ← island.getCurrentBest()
3 updateCurrentBestPool(islandCurrentBest)
4 if currentBatteryLevel ≥ 20% then
5     foreignerSolution ← pickSolutionFromCurrentBestPoolRandomly()
6     pushToIsland(island, foreignerSolution)
7 end
```

3.4 Solution Representation

In the experiments described here, we evolve variants of Facebook's Rebound library. This library consists of over 1200 lines of Java code. The most direct representation of representing individuals as variants of source code would give an intractably large search space (Petke et al. 2019) given the small number of evaluations available. To avoid this problem, we focus on optimising the values of key constants within the Rebound simulation. This technique, called deep-parameter-optimisation (Bruce et al. 2016), automatically lifts hardwired constants out of code and performs a sensitivity analysis to identify those that have the most impact on the non-functional requirement of interest. In earlier work (Bokhari et al. 2018) we isolated the five floating-point parameters that exhibited a clear impact on energy use of Rebound using sensitivity analysis. In this work, we represent each variant using an Android array map that associates each variable with a value. The small number of elements in this vector helps keep the search space manageable. We choose the array map data structure due to its memory efficiency over other map data structures on Android platforms (Sillars 2015). In addition, the use of the array map triggers less garbage collections during the application's lifetime; therefore it consumes less energy (Hecht et al. 2016).

3.5 Fitness Function

Fitness is measured in terms of energy use. To measure this, we sample each platform's internal battery meter before and after a trial run. In our experiments, our target platforms are the HTC Nexus 9, Motorola Nexus 6 and Motorola G4 Play, all running the Android operating system. The special feature of these devices is that they are equipped with the Maxim MAX17050 fuel gauge chip that compensates measurements for temperature, battery age and load (Maxim Integrated 2016), which provides an adequate substitute of an external meter if the measurement periods are sufficiently long (Bokhari et al. 2017b). During evolution the platforms are carefully configured to reduce the systematic noise as mentioned in Section 3.1.

At each step of the evolutionary process we compare a pair of program variants in a tournament by conducting a trial run of each variant. For each trial run we run the eight of the 44 tests that came with the Rebound library on a program variant. These eight tests are the ones that allow the functional accuracy of the Rebound variants to be measured.

Unfortunately, these tests are very short lived so the energy signal of a single run is very small. As pointed out in (Langdon et al. 2016), test duration is inversely proportional to the smallest detectable impact. Compounding this issue in our experiments, the running time and memory footprint of the Rebound test harnesses are much larger than the actual Rebound functions being measured. This means that a simple repetition of the tests to increase the energy signal has the unwanted impact of the large test harness footpring inducing a greater rate of energy consumption through interactions with memory – which adds more noise to the collected energy signal (Bhadra et al. 2009; Hussein et al. 2015). To avoid this issue, we linearly amplify the running time of the Rebound functions by inserting a dummy loop between each line of function code. This instrumentation amplifies the energy impact of any change the rebound code and allows the signal from this change to be detected from a single run of the Rebound test suite.

Note that, by using dummy loops, we make the assumption that all lines of code are roughly equal in terms of CPU usage. Of course, on real platforms different instructions can have small differences in energy consumption (Sinha and Chandrakasan 2001; Steinke et al. 2001). However, in this setting, where the target code exclusively uses CPU and RAM, the assumption of uniform usage does not adversely affect search. In addition, we perform a final validation of the optimised configuration using the non-amplified code and the results of this validation are presented at the end of this paper.

3.6 Functional Accuracy

Since Rebound is a spring physics library, we need to preserve its accuracy during the optimisation process. For Rebound, accuracy is measured in terms of how well the produced spring position sequences are preserved. To calculate the accuracy of the evolved solutions, we use the Mean Average Error (MAE) of the test results for every test case. This helps to compute the deviation from the test oracle (the original Rebound library), which defines the correct animation behaviour for each test case. We then take the average of all calculated MEAs. We use Eq. 1 to compute this average MAE:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |result_i - oracle_i| \tag{1}$$

Solutions with an MAE greater than 1.0 are considered invalid and and cannot win in a tournament. We use this threshold to ensure that evolved solutions produce acceptable animations (Bokhari et al. 2018).

4 Experimental Setup

To evaluate our approach, we conducted an experiment on five different platforms. The devices used in our experiments differ in their hardware and system specifications as shown in Table 2. We conducted ten experiments on each device (for a total of 50 runs) to evaluate our proposed approach. In addition, ten isolated runs (where no immigration was allowed) were conducted on each device to compare the performance of the evolutionary process using the two models (i.e distributed and isolated).

The experimental run time and recharging time differ significantly from device to another. The runtime of only one experimental run on the Nexus 6 running Android 6 is roughly 4.7 hours, whereas the same device running Android 7 takes 7.7 hours to finish one run. To fully recharge Nexus 6, seven hours is needed. The MG4 Play requires three hours of recharging and it takes about 11.8 hours to finish one experiment. The Nexus 9 runs for about six hours and requires roughly eight hours recharging. The required time for recharging and the runtime of the experiments on each device makes these evolutionary runs quite time-consuming.

Each experiment runs from full battery to 20%. We restrict the experiments run in this way, because we observed abnormal behaviours from the used devices when the battery

Table 2 The list of the used devices and their specifications. The stock ROM is used in each device

Device	Android	SoC	CPU	Memory
Nexus 6	6 & 7	Snapdragon 805	Q-core 2.7 GHz Krait 450	3 GB
MG4 Play	6 & 7	Snapdragon 410	Q-core 1.2 GHz Cortex-A53	2 GB
Nexus 9	6	Nvidia Tegra K1	D-core 2.3 GHz Denver	2 GB

drops below 20%. These behaviours include random reboots, more background processing and different system states that interfere with our experiment results.

To allow asynchronous communication between platforms, we used a Windows 10 machine with 24GB memory and Intel i7-6700 CPU clocked at 3.4GHz. This machine also hosts the pool of solutions in our dEA algorithm.

To analyse results, we track the fitness of each run over the evaluations carried out in each run and examine the effects of immigration on the local evolutionary processes. We also validate the evolved individuals and measure how they perform on each platform.

4.1 Results

This section presents the results of our empirical evaluation of our distributed EA [2][3][4].

4.2 Objective Space

Figure 2 shows the energy use (as measured on the device during evolution) of the tournament winners evolved using the islands and isolated models during all experimental runs (10 runs per device per model). The y-axis shows the energy use of the corresponding evaluation number on x-axis. The island model results are presented in blue while the isolated model results are shown in red.

As can be seen, the use of islands increases the number of evaluations possible in most experiments. This is because once a fit solution that improves the energy use is found and exchanged, the evolutionary search process is directed into a region of the search space where neighbouring solutions *consume less energy*, consequently, the remaining battery is consumed efficiently to generate more solutions (on the original island and other islands once such information is propagated). This extended run-time from the dEA model is a promising indication that the individuals evolved under this model are using less energy.

4.3 Solution Space

This section presents examples of solution structures represented by their decision variables, and how these decision variables are distributed in the solution space. We use the best found solutions (champions) evolved on each platform using each evolutionary model. The validation of these solutions is presented in Section 5.

Figure 3 illustrates the values of the decision variables in each champion. To compare these values with the original configuration of Rebound, we provide the solutions as factors applied to the original configuration. The x-axis shows the champions and the y-axis shows the magnitude of change applied to the original values in a log scale. The name of a parameter denotes the class it is found in, its type, the line of code, and its position/order in the line.

Quite interestingly, the champions of the ten experiments are relatively similar. For instance, there is a notable increase in Spring_DOUBLE_46_1 value in all solutions. The value of this decision variable is the threshold that determines when a spring is at a

[2]Our results can be retrieved from https://github.com/mahmoud-bokhari/distributed_GI

[3]The optimisation framework that runs on the PC can be found at https://github.com/mahmoud-bokhari/SmartDevicesEnergyOptimiser

[4]The smart devices and logger can be found at https://github.cs.adelaide.edu.au/a1600329/BatteryMonitor

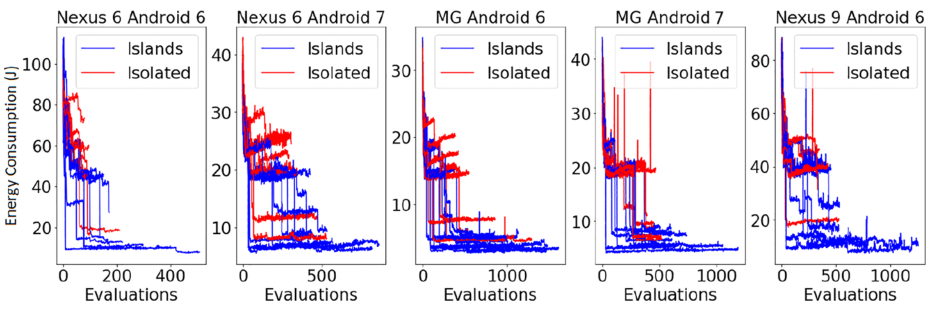

Fig. 2 The results of the 20 runs (10 per each method) on each device. Blue and red lines shows the fitness values of the evolved solutions using the distributed and isolated models, respectively. Generating fitter solutions at early stages increases the number of evaluations.

rest state. A spring's initial velocity is zero, and as the spring is animated, the velocity increases if it is stretching, or decreases if it is compressing. When the velocity reaches the value of Spring_DOUBLE_46_1, the spring stops and therefore the animation task finishes. Therefore, increasing the threshold can result in smaller periods of animation tasks.

4.4 Immigration

Figure 4 shows the overall success rate of accepting an immigration on each island. The y-axis presents the source island where immigrants evolved, and the x-axis shows the destination islands. As can be seen, there is no one island for which the majority of its solutions were successfully accepted on other islands. On the contrary, we observe that the Nexus 9 A6 island rejects most of the sent immigrants (i.e., immigrants fail to outperform locals).

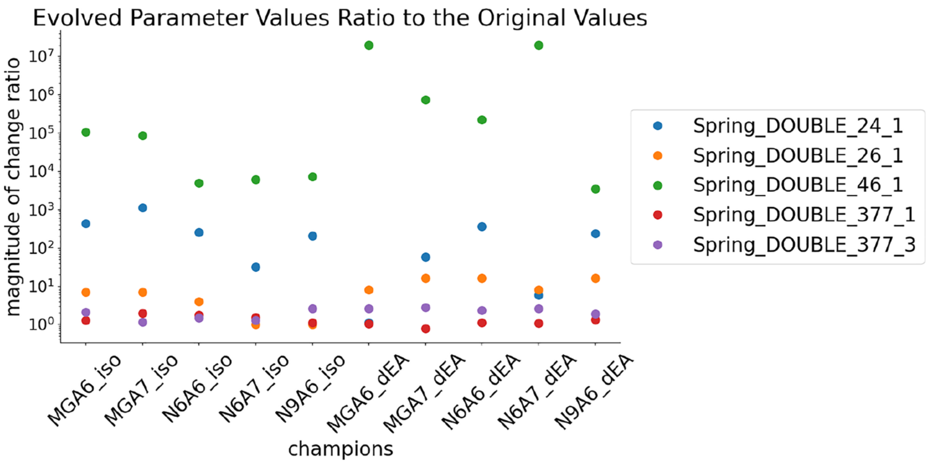

Fig. 3 The magnitude of changes applied to each decision variable in each champion compared to the original values. The y-axis is in a log scale. Champions' names denote the platform and the evolutionary model used to evolve it. For example, N6A6_iso is the best found solution found on Nexus 6 running Android 6 using the isolated model.

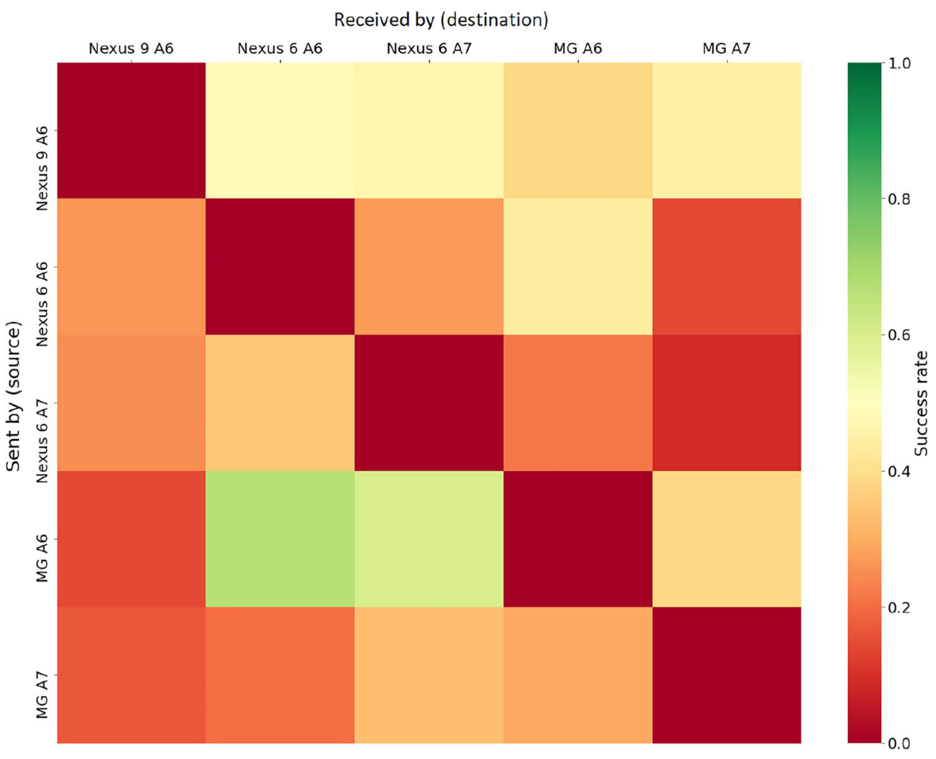

Fig. 4 Cross-Platform success rate for immigrants. Rows are the source platforms and columns are the destination platforms.

Besides the HW differences between this platform and other platform, its evolutionary process is the fastest among other devices, it can evaluate twice as many solutions as the other islands during the same amount of time.

Other islands have more mixed results with some platform combinations resulting in more moderate acceptance rates. Although the difference is moderate, we observe that islands running Android 7 generate solutions that do not survive on foreign islands. On average, an immigrant from an island running Android 7 has a success rate of 20% ± 0.1%. On the other hand, solutions generated by islands running Android 7 has on average a success rate of 31% ± 7%.

To study the correlation between the number of successful immigrants and the evolutionary speed of islands, we used Spearman correlation (Spearman 1987). To calculate the island evolution speed, we counted the number of solutions each island generated during 100% to 90% of the battery level in the ten experiments. We chose this interval because no immigration occurred between the islands during that interval. We then took the evolution speed median of the ten experiments for each device. We then counted the number of successful immigrants during the ten experiments (i.e. immigrants that survived on other islands). The result of Spearman correlation analysis is R = 0.72 with p-value of 0.17. This means although there seems to be a correlation, by normal standards, the association between the two variables would not be considered statistically significant. It is worth mentioning that our sample size is small and could impact the result.

4.5 Causality: Relative Impact of Successful Immigrants

This work is premised on the belief that successful immigrants will have a, predominantly, positive impact on the trajectory of evolution on the receiving platform. This impact would be visible as a reduction in energy consumption on the receiving platform in the evaluations after an immigrant is accepted and supplants the current best individual. This reduction can be measured relative to the evaluations prior to the individual being accepted. In order to confirm that a measured reduction is not simply due to the normal course of evolution, we can also sample around the non-accepted immigrants as a baseline. Such non-accepted immigrants can have no impact on the evolved population and, as-such, they can assume an analogous role to placebos. The number of samples on either side of the immigration event needs to be large enough to sample a reasonable portion of evolutionary history but small enough not to overlap frequently with other immigration events. In this study, a window of 20 samples on either side of the accepted immigrant was chosen.

To visualise the differential impact of accepted and unaccepted immigrants on energy use, we plot each immigration event in a 2D space where the x-dimension is the sampled energy use *before* the immigration event and the y-axis is the energy use *after*. With this setup any event on a 45 degree line extending from the origin is energy-neutral. Any event below this line has reduced energy use and any event above this line has increased energy use. Figure 5 plots these events across all trials for all experimental platforms. The accepted (successful) immigrants are in blue and the more common, unaccepted, immigration events (that do not impact the local population) are in red. The larger dots represent events where the energy use is significantly more or less than previously measured (using a single-tailed Wilcoxon rank statistic p<=0.01).

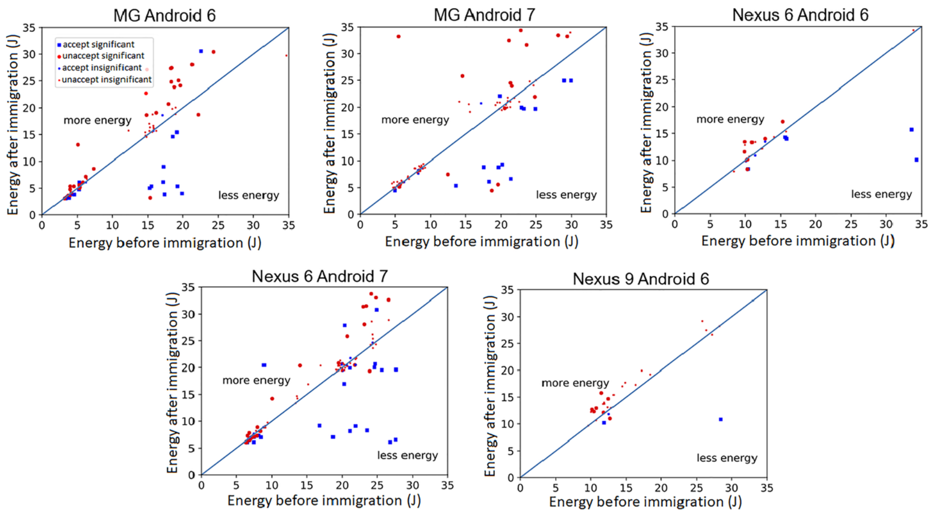

Fig. 5 Scatter plots mapping the before-and-after energy use of accepted (blue dots) and unaccepted immigrants (red dots) during evolution on five platforms. For each dot the x-position is the average energy consumption of the 20 sampled variants *before* the immigration event and the y-position is the average energy consumption of the 20 samples after the immigration event. Because the unaccepted immigrants (red dots) do not change the current variant they act as a placebo. Accepted immigrants (blue dots) are much more concentrated *below* the 45-degree energy-neutrality line and are measured as using significantly less energy than prior to the immigration event.

While the outcomes are noisy on some platforms, it can be observed that a larger proportion of blue dots is below the line than red dots. In some cases, the successful immigrants lead to a very large reduction in power consumption. Correspondingly large reductions for unaccepted immigrant events (due to noise or the natural course of evolution) are relatively infrequent. This indicates that accepted immigrants have a medium-term positive impact on energy efficiency.

5 Validation

In order to see if the results obtained during evolution persist when the busy loop is removed and are maintained across platforms we conducted a series of validation experiments. For validation, we removed code amplification from the evolved variants and, for each sample run, we repeat each test case 1000 times to increase the energy signal. Then, to help ensure a fair comparison between variants, we use our R3-VALIDATION approach to conduct the validation experiment (Bokhari et al. 2020a).

The R3-VALIDATION approach evaluates program variants in a rotated-round-robin fashion at a frequency that improves the chances that each program variant is run over a similar set of system energy state as the others (Bokhari et al. 2020a). This setup helps compensate for the fact that mobile platforms have highly variable energy states where the most significant changes are observed on moderate time scales of minutes to hours.

Our R3-VALIDATION validation runs also embed careful statistical analyses using the Wilcoxon rank test (Wilcoxon 1992) and the Vargha and Delaney $\hat{A}_{12}$ effect size (Vargha and Delaney 2000). Solutions are compared to each other (i.e. pairwise fashion) using these statistical measures, We use these tests (i.e. Wilcoxon and Vargha Delaney) because we observe that the collected samples of energy use of most solutions come from non-normal distributions. The Vargha and Delaney effect size calculates the expected probability that solution 1 consumes less energy than solution 2. For instance, if $\hat{A}_{12} = 0.8$, then solution 1 is expected to consume less than solution 2 80% of the time. To test for normality, we used the Shapiro-Wilk test (Shapiro and Wilk 1965), which tests if a sample comes from a normal distribution with unknown mean and variance, against the alternative that it does not come from a normal distribution. This test has been rigorously evaluated in the literature (Bee Wah and Mohd Razali 2011; Mendes and Pala 2003; Keskin 2006)

We start validating best-found solutions in every experiment using each evolutionary model (isolated evolution vs. island model) on each device. We then take the best validated solutions from both models and run them against each other in a tournament on the platform that they completed evolution on. The results of this step determine whether dEA-evolved solutions outperform solutions evolved in isolation. Next, we take the winners of this process from each device, and run them in tournaments on all devices to determine whether a robust solution can be achieved using either the dEA or the solutions evolved isolation.

5.1 Model Validation per-Device

Figure 6 shows the results of validating the best found solutions using the isolated model against the original configuration. The results are sorted incrementally based on the median of each dataset. Based on the Wilcoxon test, the evolved solutions improve significantly the energy consumption compared to the original configuration, with a maximum *p_value* of 7.5e-08. In addition, the effect size is large in all cases ($\hat{A}_{12} > 0.71$).

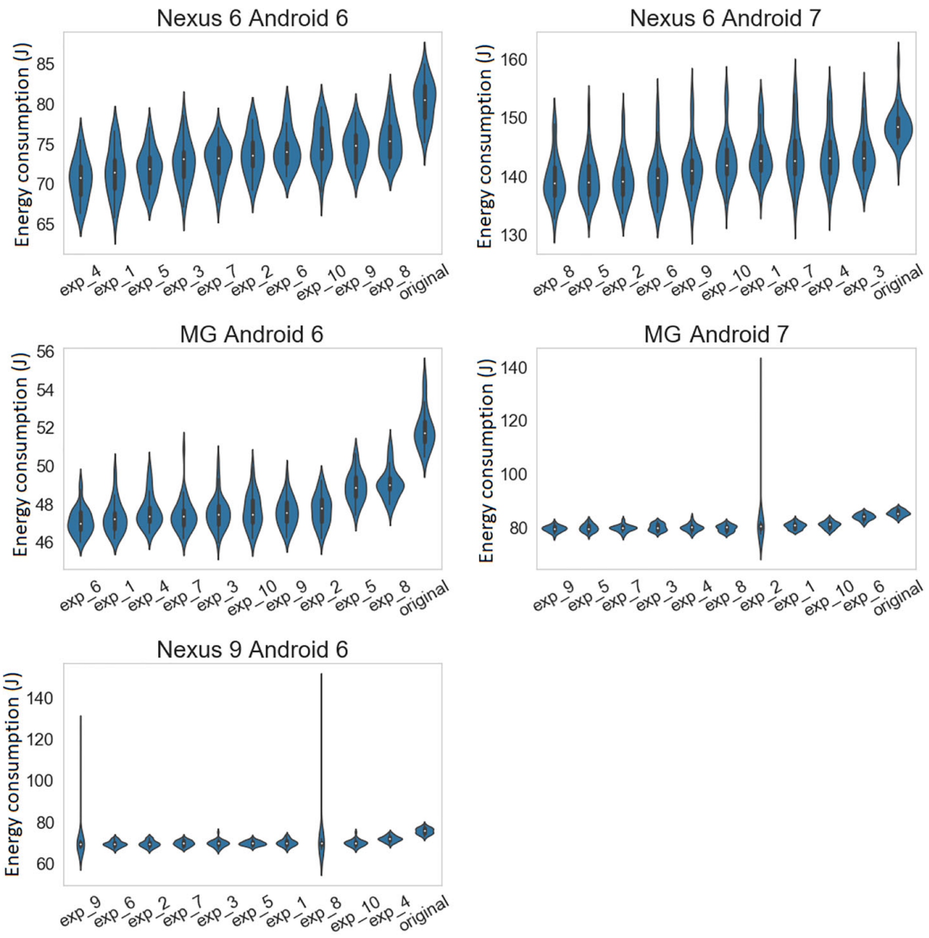

Fig. 6 Validation results of best found solutions in the ten experiments using the isolated model to the original solution.

Figure 7 presents the results of validating the best found solutions (sorted by the median of each dataset) using the dEA model against the original configuration. Interestingly, we found that the last tournament winner in the tenth experiment on MG Android 7 and the last winner in the fourth experiment on Nexus 9 Android 7 do not have a significant difference compared to the original configuration with *p_value* of 0.81 and 0.51, respectively. In addition, the magnitude of the difference is negligible in both cases with $\hat{A}_{12} < 0.56$. The rest of the evolved solutions have significantly improved the energy consumption compared to the original with large effect sizes.

Figure 6 shows that Nexus 9 running Android 6 had two outliers during validating the results of experiments number nine and eight, and MG 6 running Android 7 had also an outlier during the validation of the second experiment's results. Additionally, Fig. 7 shows an outlier in the validation results of experiment eight on device MG 6 running Android 7. After further inspections, we found that the number of background processes increased notably due to the Google Services' processes (such as Google Play and Google Mobile

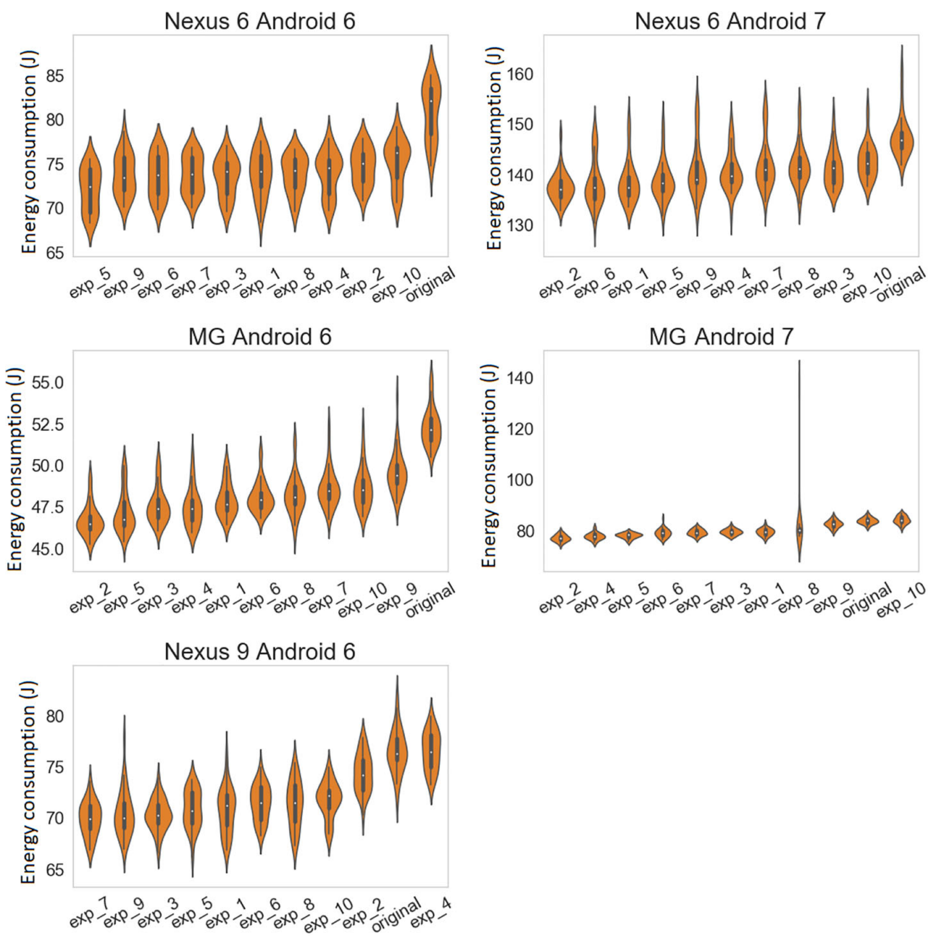

Fig. 7 Validation results of best found comparing solutions in the ten experiments using the dEA model to the original solution.

Services) were trying to perform sync operations. Unfortunately, although we deactivated periodic update operations, such behaviour is unavoidable in smart devices environments. This is because terminating these processes can result in unexpected behaviours such as device rebooting.

5.2 Winners Validation

Now, we present the results of running the best found solution using the isolated model against its counterpart evolved using the dEA models on each device shown in Fig. 8. The datasets in each plot are sorted by their medians. As can be seen, the distributions of the collected samples do not conform to the normal distribution. For instance, the distribution of Nexus 6 Android 7 data is bi-modal.

In terms of energy improvement, solutions generated using the dEA model on both MG devices significantly improve the energy consumption with large effect sizes. On the other

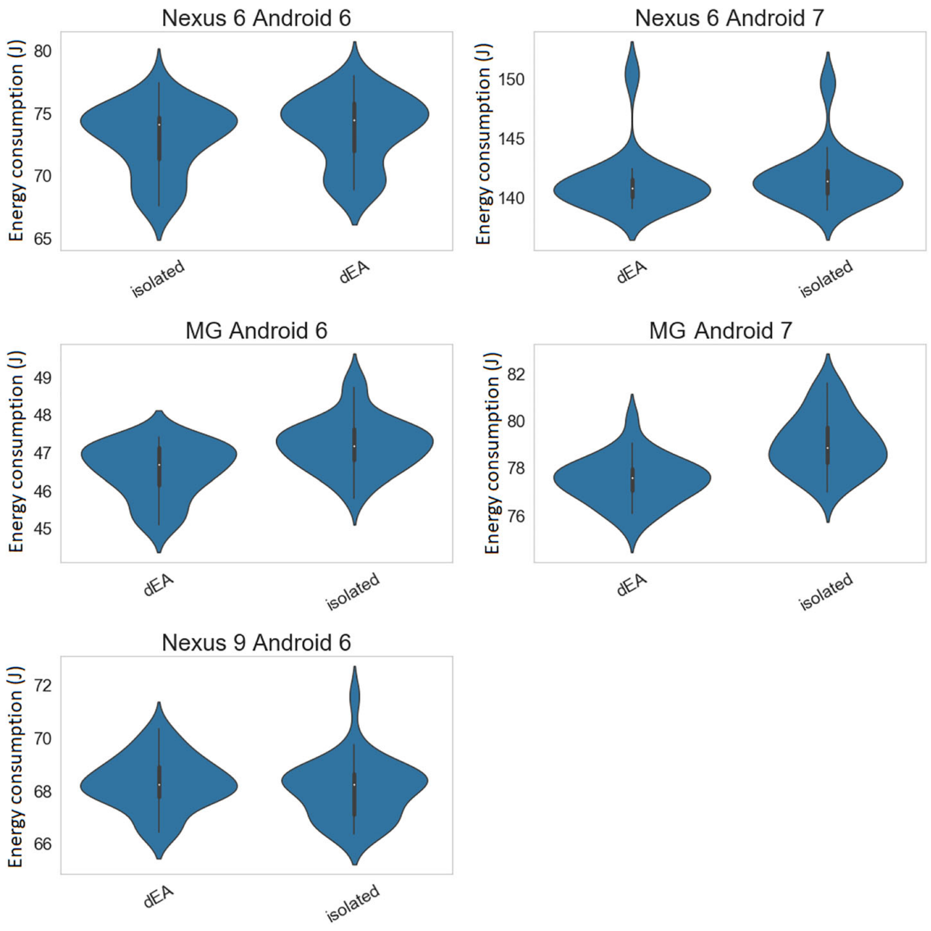

Fig. 8 Validation results of best found solutions using isolated and dEA models running on the device on which they evolved.

hand, there is no significant difference between the dEA and isolated results in the rest of the devices.

5.3 Champions Validation

After determining the best found solution using isolated and dEA models on each device, we combine all the solutions and run them against each other on all devices. Figure 9 shows the result of validating all winners on each device. As can be seen, the solution *MGA6_dEA* outperforms all other solutions on three platforms based on the median energy consumption (left column of Fig. 9).

The result of the pairwise comparison between the solutions using the Wilcoxon test and Vargha-Delaney effect size is presented in Fig. 10. I shows a heatmap for the effect

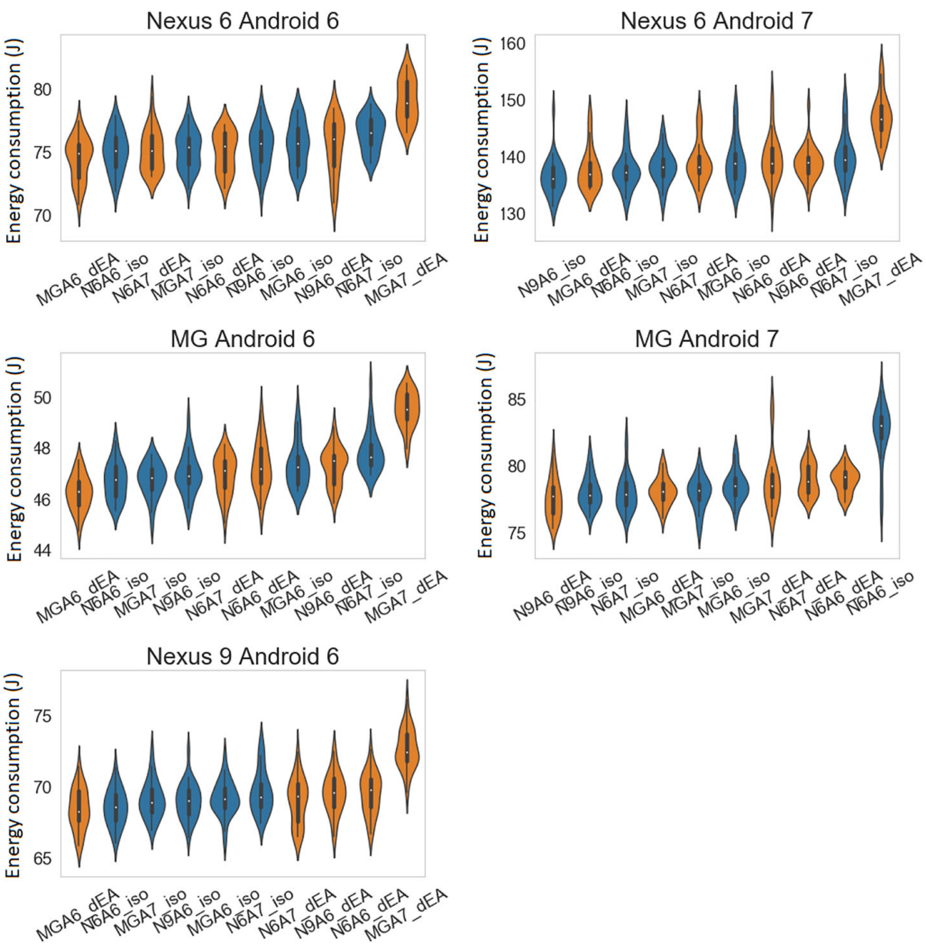

Fig. 9 Isolated (iso) and dEA winners from each device are validated using R3-validation approach.

size results, and each cell is labelled with I or S to indicate whether a significant statistical difference exists. The null hypothesis in each test is *left_solution* is significantly less than *right_solution*.

Generally speaking, we observe that no one solution dominates all other solutions on all devices. However, the results show that none of the generated solutions outperform solution *MGA6_dEA* on all platforms. In addition, it outperforms all solutions in its native platform (i.e. MG Android 6).

In addition, we observe that regardless of the evolutionary optimisation model and HW used, islands running Android 6 generate better solutions. For example, Nexus 6 Android 6 and MG Android 6 produced fitter solutions that outperform their counterparts evolved on Android 7. It is of note that there was no corresponding advantage for the variant evolved in isolation on Nexus 6 Android 6. This highlights the potential for the island model for evolution to produce variants with superior cross platform performance.

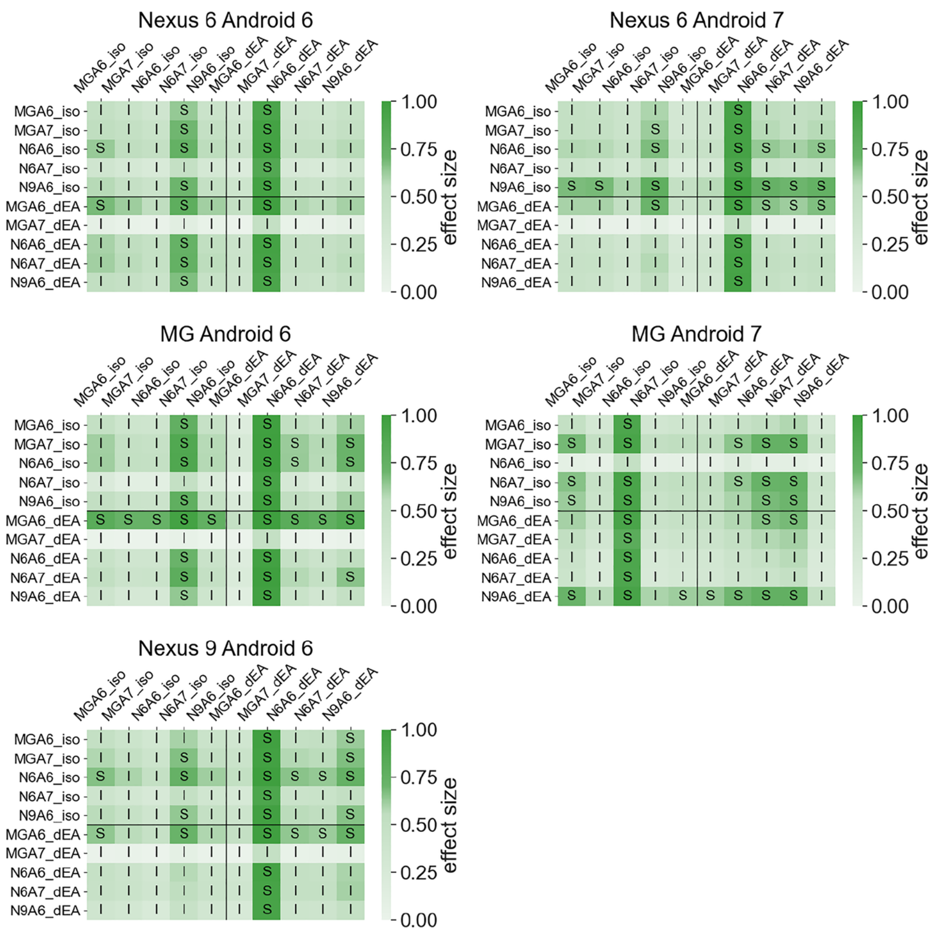

Fig. 10 Isolated (iso) and dEA winners: pairwise comparison. Colours are for effect size, the letters "I" or "S" denote, respectively whether the difference between the variants compared in that cell is insignificant or significant on that platform ($p <= 0.05$). As can be seen the variant evolved on Nexus 6 Android 6 with the dEA island model dominates over other variants except on MG Android 7.

6 Threats to Validity

A limitation of this study is the use of a single library, albeit a popular one, as a target for optimisation. Libraries, as opposed to programs, make excellent targets for optimisation because any gains found are amplified across all applications in which they are used. An example of this benefit is seen in (Couto et al. 2020) where replacing an energy hungry HashMap with a more efficient ArrayMap produces potentially widespread benefits. However, by targeting a single application it is not possible to tell if the results and recommendations produced by the work will generalise to applications with different features. In particular, the target application used in this work is CPU-bound. Further work would be needed to explore how our approach generalises to software whose primary energy consumption comes from non CPU devices.

The biggest challenge to the evolution of software variants on and for mobile platform is their time variant energy state. Selection during an evolutionary process can be biased by the influence of system state on energy readings. To minimise the effect of energy state during validation we used the R3-VALIDATION approach to help maximise the chance that variants are measured in similar energy states. However, it is not possible even with multiple samples to eliminate all chance of bias (though the domination of one variant (N6A6_dEA) across many sets of tournaments give strong surety that observed advantage of this variant is real).

Another challenge to address is what happens when other devices such as wireless and network are switched on. Will the new energy states induced by these devices impact on the relative ranking of solutions. Future work will be required to verify that observed improvements persist on devices in the wild. Such work will depend on the development of instrumentation to detect small energy effects across a large number deployed devices.

Finally, as new versions of systems are released, they use ever more sophisticated interventions to manage energy consumption. These interventions will interact with optimised solutions in ways that could affect the rankings of solutions. Validation of solutions will have to be extended to new systems as they become available.

7 Conclusion

The results presented in this work demonstrate the practicality and the potential benefits of using a distributed island-inspired algorithm to improve in-vivo search for program variants that use less energy on heterogeneous platforms.

Our analyses of the impact of the migration of program variants on evolutionary search shows that the use of migrants has benefits in terms of longer evolutionary run-times and improvements in local search. Careful validation of the found solutions shows that our distributed approach produces variants that perform well cross-platform - exhibiting energy efficiency that is either equivalent to or better than variants produced by isolated single-platform evolution.

While proof-of-concept in this work demonstrates the potential value of using immigration to speed up search for energy-efficient program variants there is substantial scope to build on these results. We can explore the application of this technique to the optimisation of other program variants. We can further improve the evaluation setup to further reduce noise in the energy signal during evolution and mitigate the impact of state changes on the platform on the evolutionary process. This work might be properly extended to detecting and accounting for large changes in system state between reboots. Finally, there is scope for extending this setup to the evolution of other non-functional requirements such as memory use and network use and extend this methodology to parallel optimisation on larger numbers of platforms.

References

Alameldeen A. R., Wood D. A. (2003) Variability in architectural simulations of multi-threaded workloads. In: 9th International Symposium on High-Performance Computer Architecture (HPCA), IEEE

Bee Wah Y., Mohd Razali N. (2011) Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests

Bhadra S., Conrad A., Hurkes C., Kirklin B., Kapfhammer G. M. (2009) An experimental study of methods for executing test suites in memory constrained environments. In: Workshop on Automation of Software Test

Blackburn S. M., Garner R., Hoffmann C., Khang A. M., McKinley K. S., Bentzur R., Diwan A., Feinberg D., Frampton D., Guyer S. Z., et al. (2006) The dacapo benchmarks: Java benchmarking development and analysis. In: ACM SIGPLAN Notices, vol 41, ACM

Bokhari M., Wagner M. (2016) Optimising energy consumption heuristically on android mobile phones. In: Genetic and Evolutionary Computation Conference Companion (GECCO). ACM

Bokhari M. A., Alexander B., Wagner M. (2018) 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 International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous). ACM

Bokhari M. A., Alexander B., Wagner M. (2020a) Towards rigorous validation of energy optimisation experiments. In: Genetic and Evolutionary Computation Conference (GECCO)

Bokhari M. A., Bruce B. R., Alexander B., Wagner M. (2017a) 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 (GECCO). ACM

Bokhari M. A., Wagner M., Alexander B. (2019) The quest for non-functional property optimisation in heterogeneous and fragmented ecosystems: A distributed approach. In: the Genetic and Evolutionary Computation Conference Companion (GECCO). Association for Computing Machinery

Bokhari M. A., Wagner M., Alexander B. (2020b) Genetic improvement of software efficiency: The curse of fitness estimation. In: Genetic and Evolutionary Computation Conference Companion (GECCO)

Bokhari M. A., Weng L., Wagner M., Alexander B. (2019) Mind the gap–a distributed framework for enabling energy optimisation on modern smart-phones in the presence of noise, drift, and statistical insignificance. In: IEEE Congress on Evolutionary Computation (CEC), IEEE

Bokhari M. A., Xia Y., Zhou B., Alexander B., Wagner M. (2017b) Validation of internal meters of mobile android devices, vol abs/1701.07095

Brownlee A. E. I., Burles N., Swan J. (2017) Search-Based energy optimization of some ubiquitous algorithms, vol 1

Bruce B. R., Petke J., Harman M., Barr E. T. (2019) Approximate oracles and synergy in software energy search spaces, vol 45

Bruce B. R., Aitken J. M., Petke J. (2016) Deep parameter optimisation for face detection using the Viola-Jones algorithm in OpenCV. In: Symposium on Search-Based Software Engineering (SSBSE). Springer

Bruce B. R., Petke J., Harman M. (2015) Reducing energy consumption using genetic improvement. In: Genetic and Evolutionary Computation Conference (GECCO), ACM

Burles N., Bowles E., Brownlee A. E. I., Kocsis Z. A., Swan J., Veerapen N., Labiche Y. (2015) Object-oriented genetic improvement for improved energy consumption in Google Guava. In: Symposium on Search-Based Software Engineering (SSBSE). Springer

Couto M., Saraiva J., Fernandes J. P. (2020) Energy refactorings for android in the large and in the wild. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 217–228, https://doi.org/10.1109/SANER48275.2020.9054858.

Cruz L., Abreu R. (2021) On the energy footprint of mobile testing frameworks. IEEE Trans Softw Eng 47(10):2260–2271. https://doi.org/10.1109/TSE.2019.2946163

Dong M., Zhong L. (2011) Self-constructive high-rate system energy modeling for battery-powered mobile systems. In: Mobile Systems, Applications, and Services (MobiSys). ACM

Ferreira D., Dey A. K., Kostakos V. (2011) Understanding human-smartphone concerns: a study of battery life. In: International Conference on Pervasive Computing (PerCom), Springer

Georges A., Eeckhout L., Buytaert D. (2008) Java performance evaluation through rigorous replay compilation. In: ACM SIGPLAN Notices, vol 43, ACM

Gong Y.-J., Chen W.-N., Zhan Z.-H., Zhang J., Li Y., Zhang Q., Li J.-J. (2015) Distributed evolutionary algorithms and their models: A survey of the state-of-the-art, vol 34

Hao S., Li D., Halfond W. G. J., Govindan R. (2013) Estimating mobile application energy consumption using program analysis. In: International Conference on Software Engineering (ICSE). IEEE Press

Hecht G., Moha N., Rouvoy R. (2016) An empirical study of the performance impacts of android code smells. In: IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft). IEEE

Hindle A. (2016) Green software engineering: The curse of methodology. In: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 5

Hoque M. A., Siekkinen M., Khan K. N., Xiao Y., Tarkoma S. (2016) Modeling, profiling, and debugging the energy consumption of mobile devices, vol 48

Hort M., Kechagia M., Sarro F., Harman M. (2021) A survey of performance optimization for mobile applications

Huang X., Blackburn S. M., McKinley K. S., Moss J. E. B., Wang Z., Cheng P. (2004) The garbage collection advantage: improving program locality, vol 39

Hussein A., Payer M., Hosking A., Vick C. A. (2015) Impact of gc design on power and performance for android. In: 8th ACM International Systems and Storage Conference (SYSTOR). ACM

Kalibera T., Bulej L., Tuma P. (2005) Benchmark precision and random initial state. In: International Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS). SCS

Keskin S. (2006) Comparison of several univariate normality tests regarding type i error rate and power of the test in simulation based small samples, vol 2

Langdon W. B., Petke J., Bruce B. R., Hart E., Lewis P. R., López-Ibáñez M., Ochoa G., Paechter B. (2016) Optimising quantisation noise in energy measurement. In: Parallel Problem Solving from Nature (PPSN). Springer

Li D., Hao S., Halfond W. G. J., Govindan R. (2013) Calculating source line level energy information for android applications. In: International Symposium on Software Testing and Analysis (ISSTA). ACM

Linares-Vásquez M., Bavota G., Cárdenas C. E. B., Oliveto R., Di Penta M., Poshyvanyk D. (2015) Optimizing energy consumption of guis in android apps: A multi-objective approach. In: 10th Joint Meeting on Foundations of Software Engineering (FSE). ACM

Maxim Integrated (2016) MAX17047/MAX17050 ModelGauge m3 Fuel Gauge. Accessed 2 February 2020, https://datasheets.maximintegrated.com/en/ds/MAX17047-MAX17050.pdf

McDonnell T., Ray B., Kim M. (2013) An empirical study of api stability and adoption in the android ecosystem. In: IEEE International Conference on Software Maintenance (ICSM). IEEE Press

Mendes M., Pala A. (2003) Type i error rate and power of three normality tests, vol 2

Morales R., Saborido R., Khomh F., Chicano F., Antoniol G. (2018) Earmo: An energy-aware refactoring approach for mobile apps

Murmuria R., Medsger J., Stavrou A., Voas J. M. (2012) Mobile application and device power usage measurements. In: Software Security and Reliability. IEEE

Mytkowicz T., Diwan A., Hauswirth M., Sweeney P. F. (2009) Producing wrong data without doing anything obviously wrong!, vol 37

Pascual G. G., Pinto M., Fuentes L. (2015) Self-adaptation of mobile systems driven by the common variability language, vol 47

Pathak A., Hu Y. C., Zhang M., Bahl P., Wang Y.-M. (2011) Fine-grained power modeling for smartphones using system call tracing. In: European Conference on Computer Systems (EuroSys). ACM

Peltonen E., Lagerspetz E., Nurmi P., Tarkoma S. (2015) Energy modeling of system settings: A crowdsourced approach. In: 2015 IEEE International Conference on Pervasive Computing and Communications (PerCom), pp 37–45, https://doi.org/10.1109/PERCOM.2015.7146507

Petke J., Alexander B., Barr E. T., Brownlee A. E. I., Wagner M., White D. R. (2019) A survey of genetic improvement search spaces. In: Genetic and Evolutionary Computation Conference Companion (GECCO)

Pressman R. S. (2005) Software engineering: a practitioner's approach, Palgrave Macmillan

Pusukuri K. K., Gupta R., Bhuyan L. N. (2012) Thread tranquilizer: Dynamically reducing performance variation, vol 8

Sachindran N., Moss J. E. B. (2003) Mark-copy: Fast copying gc with less space overhead. In: ACM SIGPLAN Notices, vol 38, ACM

Schulte E., Dorn J., Harding S., Forrest S., Weimer W. (2014) Post-compiler software optimization for reducing energy. In: 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM

Schwefel H.-P. (1977) Numerische optimierung von computer-modellen mittels der evolutionsstrategie.(teil 1, kap. 1-5), Birkhäuser

Shapiro S. S., Wilk M. B. (1965) An analysis of variance test for normality (complete samples), vol 52

Shye A., Scholbrock B., Memik G. (2009) Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In: Symposium on Microarchitecture. ACM

Sillars D. (2015) High performance android apps: Improve ratings with speed, optimizations, and testing, O'Reilly Media, Inc.

Sinha A., Chandrakasan A. P. (2001) Jouletrack-a web based tool for software energy profiling. In: 38th Design Automation Conference (DAC)

Skolicki Z., De Jong K. (2005) The influence of migration sizes and intervals on island models. In: 7th Annual Workshop on Genetic and Evolutionary Computation

Spearman C. (1987) The proof and measurement of association between two things. The American Journal of Psychology 100(3/4):441–471. ISSN 00029556. http://www.jstor.org/stable/1422689

Steinke S., Knauer M., Wehmeyer L., Marwedel P. (2001) An accurate and fine grain instruction-level energy model supporting software optimizations. In: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)

Vargha A., Delaney H. D. (2000) A critique and improvement of the cl common language effect size statistics of mcgraw and wong, vol 25

Wilcoxon F. (1992) Individual comparisons by ranking methods. In: Breakthroughs in statistics. Springer

Mahmoud A. Bokhari is an Assistant Professor at the School of Computer Science, Taibah University, Saudi Arabia. He has done his PhD studies in Computer Science at the University of Adelaide, Adelaide, Australia. His research interests include green software engineering, search-based software engineering, ubiquitous computing, software testing, and Blockchains sustainability.

Bradley Alexander is a senior software engineer at Optimatics and an adjunct senior lecturer at the School of Computer Science University of Adelaide. His research interests involve the use of evolutionary search to explore design spaces. His work includes: optimisation of energy on mobile platforms; optimisation of water distribution systems; using search and deep learning for pipe condition assessment; optimisation of placement of wave energy converters; search based software engineering; features and constraints in evolutionary art; and fast evolution of geoscience models.