Search@Home: A Commercial Off-the-Shelf Environment for Investigating Optimization Problems

Erik M. Fredericks[1] and Jared M. Moore[2]

[1] Dept. of Computer Science & Engineering, Oakland University
Rochester MI, 48309, USA, fredericks@oakland.edu
[2] School of Computing and Information Systems, Grand Valley State University
Allendale MI, 49401, USA, moorejar@gvsu.edu

Abstract. Search heuristics, particularly those that are evaluation-driven (e.g., evolutionary computation), are often performed in simulation, enabling exploration of large solution spaces. Yet simulation may not truly replicate real-world conditions. However, search heuristics have been proven to be successful when executed in real-world constrained environments that limit searching ability even with broad solution spaces. Moreover, searching *in situ* provides the added benefit of exposing the search heuristic to the exact conditions and uncertainties that the deployed application will face. Software engineering problems can benefit from *in situ* search via instantiation and analysis in real-world environments. This paper introduces *Search@Home*, an environment comprising heterogeneous commercial off-the-shelf devices enabling rapid prototyping of optimization strategies for real-world problems.

Keywords: Real-world systems, evolutionary search, in-situ search, search-based software engineering

1 Introduction

Commercial off-the-shelf (COTS) microcomputers (e.g., Raspberry Pi, Arduino, BeagleBone, etc.[3]) are democratizing access to many-core distributed computing environments. Such devices are generally constrained in terms of available processing power and hard drive space, limiting their ability to individually carry out complicated tasks quickly. Moreover, such devices will generally consume far less power than a typical device used for simulation (e.g., server blade). We posit that a microcomputing environment is therefore beneficial in terms of rapidly prototyping search heuristics, in real-world conditions, yet may require additional time to complete complex computing tasks. Specifically, we highlight search-based software engineering (SBSE) as an attractive domain for real-world search.

[3] See https://www.raspberrypi.org/, https://www.arduino.cc/, and https://beagleboard.org, respectively.

Optimization performed in real-world situations allows the system to analyze relevant information from data specific to its operating context (i.e., combination of system and environmental parameters) rather than simulating such parameters [2]. Self-adaptation has been applied to cyber-physical systems enabling reconfiguration at run time in response to uncertainty, including those systems considered to be safety critical [8]. Online evolutionary optimization has been previously performed in fields such as robotics, where a (1+1) evolutionary strategy searches for optimal neural network configurations [3]. Given the difficulties in performing speculative, evaluation-driven optimizations at run time, continuous optimization methods such as Markov chains and Bayesian methods have also been applied [4, 10]. Regardless of the method, an online optimization strategy must consider the implications of updating an executing system within its production environment.

This paper introduces and demonstrates *Search@Home*, a framework for quickly prototyping in-place search-based software engineering (SBSE) techniques using COTS hardware. *Search@Home* is intended to provide a low-cost testbed that can be deployed in a target environment to rapidly prove out online search heuristics (e.g., run-time requirements monitoring/optimization). While a longer evaluation time is to be expected with low-power hardware, the benefits of performing search in a real-world environment far outweigh the speed gains of using a simulation that may be misconfigured or inaccurately describe the environment-to-be. We next describe background and related work, demonstrate *Search@Home* on a proof-of-concept optimization problem, and outline future experiments.

2 Real-World Systems

This section presents relevant information on microcomputers and how SBSE techniques can be effective within constrained environments.

Microcomputers: Consumer-grade microcomputers have been widely propagated at inexpensive price points with the recent explosion of interest in Maker-related topics (e.g., hacking home electronics, 3D printing, etc.), many of which are targeted at STEM education and non-production projects. Moving from simulation to reality in constrained environments requires addressing concerns in:

– ***Power***: Beyond power optimization, microcomputers often supplied by sub-standard power cables, resulting in *undervoltage* (i.e., slow/erratic behavior).
– ***Temperature***: Heat can negatively impact devices without active (i.e., fans) or passive (i.e., heat sinks) cooling, leading to CPU throttling.
– ***Memory***: Microcomputers are constrained with the amount of available memory for handling computing tasks. While the newest Raspberry Pi (4B) has up to 8GB of available RAM, older models have significantly less memory. Edge devices exist specifically for heavy-duty computing (e.g., Google Coral[4]), however they are generally used for only specialized purposes.

[4] See https://www.coral.ai/.

- ***Disk space***: Depending on the device, permanent storage (e.g., EEPROM, ROM, flash memory, etc.) is often at a premium in terms of availability. Devices such as the Arduino and BeagleBone rely on programmable memory space for long-term storage, whereas the Raspberry Pi uses a microSD card for its storage.
- ***Timing constraints***: While the devices used in this paper do not use real-time operating systems, timing constraints must be explicitly handled by the engineer, else faults can occur when software deadlines are violated.

Online/hardware-based optimization: One common theme across optimization algorithms is that there exists no "free lunch" as there are always limiting factors in the application or environment [13]. Limitations for online optimization will be readily-noticeable in run time, memory overhead, etc. In constrained systems these impacts are exceedingly noticeable given their lower operating capabilities. Care must be taken when performing optimization in constrained systems.

In situ optimization has been applied to wireless sensor network applications, where run-time reconfiguration and programming models enable optimization [11]. Li *et al.* minimized power consumption of test suites in low power systems using an integrated linear programming approach [6]. Continuous optimization is a common feature in other domains as well. Wang and Boyd applied an online optimization technique (a primal barrier method) to model-predictive control applications [12]. Mars and Hundt combined static and dynamic optimization strategies to direct online reconfiguration based on scenarios [7]. Optimization has also been applied online in a data-driven capacity, where uncertainty models can inform decision making [2].

Search heuristics have also been deployed *in silica*. Genetic algorithms (GA) have been deployed to field-programmable gate arrays (FPGA) for hardware-based optimization [9], enabling rapid prototyping of fast, low-power search. Energy management is another concern for metaheuristics, specifically those involving smart power grids [5]. With additional hardware modules, *Search@Home* can be extended to use *in silica* search and act instead as a controller, and moreover, monitor energy consumption with appropriate sensors.

3 *Search@Home* Overview

For the purposes of this paper (and its initial implementation), *Search@Home* was implemented as an IoT environment comprising two Raspberry Pi 3B devices, a Raspberry Pi 4B (4GB model), a spare 2010-era netbook (acting as a point of entry to the network), and a wireless router (flashed with Tomato firmware) to provide a sandboxed network. Devices can be interchangeable – as long as the device can connect to the network it can participate in the environment. For example, an Arduino Duo could be included as long as an appropriate communication channel to the rest of the network is established (i.e., a WiFi shield is installed).

We applied a string optimization problem demonstrating the feasibility of deploying a search algorithm to a heterogeneous collection of devices. The algorithm was developed with Python 3.7 and each Raspberry Pi used Raspbian as its operating system. Given that Raspbian is a full Linux-based operating system, libraries such as `DEAP` and `DisPy` can facilitate development of multiple search heuristics or distributed cluster computing, respectively.[5] To allow for replication, we have made our parts list, source code, and results publicly available on GitHub.[6]

String search configuration: The string search application leverages a standard GA to search for a given string, where this particular GA was released as a GitHub.[7] To increase the difficulty of this task, we specified a complex string (`OPT`) comprising `ASCII` characters sampled between indices 32 and 64 and a total length of 84.[8] `OPT` is defined in Equation 1:

$$\begin{split} OPT = {}' 12th\ Symposium\ for\ Search - Based\ Software \\ Engineering \mid http://ssbse2020.di.uniba.it/' \end{split} \tag{1}$$

The fitness function defined for this study is defined as follows in Equation 2, where `ord` represents a character's Unicode integer value:

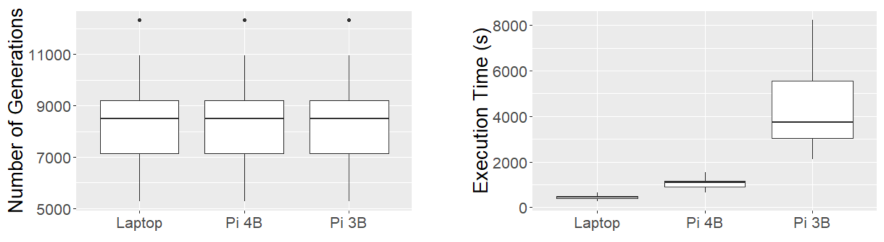
$$ff_{string} = \sum_{i=0}^{len(OPT)} |ord(TARGET[i]) - ord(OPT[i])| \tag{2}$$

This particular GA uses single-point crossover, single-point mutation, and a weighted-fitness selection function (i.e., fitness directly correlated to the probability that an individual is selected). Mutation is automatically applied to the children generated by the crossover operation, where a random gene is modified. The GA was configured to run for a maximum of $500,000$ generations, with a population size of 500, a crossover rate of 0.5, and a mutation rate of 1.0. The GA can converge early if the correct string is discovered. We performed 25 experimental replicates to ensure statistical significance, using the Wilcoxon-Mann-Whitney U-test with a significance level of $p < 0.001$. We compared the GA to random search as specified by Arcuri *et al.* [1].

For this feasibility study, we are interested in the amount of time required before convergence to the expected solution and the number of generations necessary to reach that value, as the number of generations is set very high to ensure convergence. To focus on execution time, we intentionally did not introduce parallelism or distributed processing, however such a procedure can be used on a microcomputer (e.g., Python's `multiprocessing` package). Figure 1(a) compares the number of generations required for the algorithm to converge and Figure 1(b) compares the amount of time (seconds) required to reach convergence between a current-generation laptop[9] and the Raspberry Pis. As can be seen from these

[5] See https://deap.readthedocs.io/ and http://dispy.sourceforge.net/, respectively.
[6] See https://github.com/efredericks/SearchAtHome.
[7] See https://gist.github.com/josephmisiti/940cee03c97f031188ba7eac74d03a4f.
[8] There exist $2.7 * 10^{126}$ possible combinations based on string length and characters.
[9] Intel Core i7 quad-core 2.8GHz 64-bit processor, 16GB RAM, 1TB hard drive space.

figures, there exists no difference in the number of generations required to converge to the optimal solution. However, a significant difference exists between the execution times required for convergence between each device ($p < 0.001$). Moreover, each experimental replicate resulted in convergence. These results are expected given the disparity in processing capability. However, the Raspberry Pi was able to successfully execute optimization in all cases within a reasonable amount of time, proving feasibility of optimization on constrained devices.



(a) Number of generations. (b) Execution time.

Fig. 1: Comparison of string search results between laptop and Raspberry Pis.

SBSE Implications: We now highlight three future experiments for *in situ* optimization research.

1. Distributed processing of search algorithms
2. Power concerns resulting from search
3. Implications of SBSE in production environments

Item (1) can be an interesting study of offloading SBSE tasks (e.g., requirements monitoring, fitness evaluation, etc.) to distributed nodes. Item (2) demonstrates the implications of modeling power consumption as a first-class citizen in a software model (e.g., non-functional requirements). Item (3) uses optimization *in situ* to investigate the interaction of search, software artifacts, and expressed behaviors.

4 Discussion

This paper presents *Search@Home*, an open-source framework for enabling *in situ* SBSE research within constrained environments. *Search@Home* uses inexpensive COTS hardware providing an environment in which students and researchers can quickly and effectively prototype and deploy applications that would benefit from using real-world data to support an online search procedure. For this paper, we targeted evolutionary computation, however extension to other optimization domains (e.g., continuous optimization) is feasible as well.

We demonstrate the effectiveness of *Search@Home* on a string search exemplar to demonstrate basic search feasibility, where optimal solutions are discovered in a reasonable amount of time. Future research directions for this project include incorporation of low-cost robotics environments (e.g., iRobot Roomba, Turtlebot, Lego Mindstorms, etc.), usage of a compute cluster (e.g., Beowulf cluster, high-performance compute cluster, etc.), and incorporation of cloud technologies offsetting the cost of heavy evaluations (e.g., Google Cloud Functions, Amazon Web Services Lambda functions, etc.).

Acknowledgements. This work has been supported in part by grants from the National Science Foundation (CNS-1657061), Oakland University, and Grand Valley State University. The views and conclusions contained herein are solely those of the authors.

References

1. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proc. of the 33rd Intl. Conf. on Software Engineering. pp. 1–10. ICSE '11, ACM (2011)
2. Bertsimas, D., Thiele, A.: Robust and data-driven optimization: modern decision making under uncertainty. In: Models, methods, and applications for innovative decision making, pp. 95–122. INFORMS (2006)
3. Bredeche, N., Haasdijk, E., Eiben, A.: On-line, on-board evolution of robot controllers. In: Artificial Evolution. Springer Berlin Heidelberg (2010)
4. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic qos management and optimization in service-based systems. IEEE Trans. on Software Engineering **37**(3), 387–409 (2010)
5. Lezama, F., Soares, J., Vale, Z.: A platform for testing the performance of metaheuristics solving the energy resource management problem in smart grids. Energy Informatics **1**(1), 35 (2018)
6. Li, D., Jin, Y., Sahin, C., Clause, J., Halfond, W.G.: Integrated energy-directed test suite optimization. In: Proc. of the 2014 Intl. symposium on software testing and analysis. pp. 339–350 (2014)
7. Mars, J., Hundt, R.: Scenario based optimization: A framework for statically enabling online optimizations. In: 2009 International Symposium on Code Generation and Optimization. pp. 169–179. IEEE (2009)
8. Muccini, H., Sharaf, M., Weyns, D.: Self-adaptation for cyber-physical systems: a systematic literature review. In: Proc. of the 11th Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 75–81 (2016)
9. Peker, M.: A fully customizable hardware implementation for general purpose genetic algorithms. Applied Soft Computing **62**, 1066–1076 (2018)
10. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proc. of the IEEE **104**(1), 148–175 (2015)
11. Taherkordi, A., Loiret, F., Rouvoy, R., Eliassen, F.: Optimizing sensor network reprogramming via in situ reconfigurable components. ACM Trans. on Sensor Networks (TOSN) **9**(2), 1–33 (2013)
12. Wang, Y., Boyd, S.: Fast model predictive control using online optimization. IEEE Trans. on control systems technology **18**(2), 267–278 (2009)
13. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. on Evolutionary Computation **1**(1), 67–82 (1997)