On Improving Green Mining For Energy-Aware Software Analysis

Stephen Romansky, Abram Hindle

Department of Computing Science
University of Alberta
Edmonton, Canada
{romansky, abram.hindle}@ualberta.ca

Abstract

Consumer demand for longer lasting battery life in mobile computers, as well as industry interest in energy efficient cloud infrastructure, creates a need for hardware and software energy efficiency improvements. One way to tackle this problem is from a software perspective. If it were known which software changes influenced energy consumption, then tools could be created to help software professionals create more energy efficient software. The process of extracting energy consumption information, Green Mining, is time demanding because researchers must run many tests, with sufficient coverage, on each revision in a software product multiple times. The time required for testing acts as a barrier to extracting energy consumption measurements from new software systems. Therefore, this work proposes, implements, and evaluates a search-based approximation method that trades some precision for a speed-up in the mining process. This speed-up enables researchers to study additional software systems that were too costly to investigate before.

1 Introduction

The energy efficiency of software and hardware has become a hot topic for mobile consumers and software service providers. Consumers tend to own many battery-limited mobile devices such as laptops, smart phones, and tablets. Battery-limited devices would all benefit from higher energy efficiency. In industry, there are service providers that run the same software on hundreds of computers. For these service providers, a small software energy efficiency gain could lead to large cumulative savings including reductions in cooling expenses, electricity costs [1], carbon emissions, and carbon emission taxes. If engineers and developers had more energy awareness of how software changes impacted program energy efficiency, they would be able to better set and manage software energy consumption goals. Engineers and developers could then meet the demands of industry and consumers. Collecting reliable measurements and profiles of software energy use is difficult for most software professionals because development tool support, instrumentation, and information logging support are lacking for hardware and software available to many developers. Thus, this work explores a method to speed up energy profile collection to help researchers find relationships between software change and energy consumption.

Industry and researchers have studied methods to improve energy efficiency by modifying both hardware and software. One such study in industry is the usage of power management software [2] on workstations to enforce practices such as screen savers and hardware idling after period of inactivity. Savings can also be gained from using network manager software [1] that remotely starts, or wakes up, work stations when users try to access their workstations instead of leaving the machines running while users are away. Researchers have studied additional software practices that reduce power consumption by examining the interaction of soft-

ware with the CPU, and hard drive [3]. Thus, developers can gain more energy efficiency by organizing tasks like disk access and CPU usage in their software. Other work has looked at profiling software [4, 5] to locate areas with potential optimizations with respect to energy consumption. The aim of this work is to improve the Green Mining technique so that researchers are able to access old energy consumption measurements quickly in order to locate unused energy information from software repositories.

Green Mining [6, 7] is the study of how software change affects the energy consumption of software. Green Mining provides the original methodology for extracting historic energy consumption profiles from computer programs. The original method is expensive with respect to time. A software product with many revisions takes many more test runs to profile the software's energy consumption in relation to its change over time. An *energy profile* is a profile of the past or historical energy consumption utilization of an application. In terms of Green Mining, an energy profile is the measurement of energy consumption over different revisions of a program. The research question that motivates this work is, "Given a required level of accuracy how many revisions need to be measured in order to extract a historical software energy consumption profile from a software product?" If a solution exists Green Mining of software energy profiles could be sped up, enabling practitioners to spend less time extracting profiles of their own software. In turn, researchers could use this new historical data to build corpus-based models to reason about the relationship between software change and energy consumption.

This work proposes a solution based on *search-based software engineering* (SBSE) [8], which aims to solve difficult software engineering problems with search-based, or optimal, method selection techniques. A piece-wise linear interpolation method is used to search for sets of optimal revisions that give the best historic energy consumption estimates. The best historic energy profile estimate has the least time requirement and highest accuracy. Linear interpolation, between revisions, is chosen to approximate energy consumption data because there have been no prior attempts to optimize the extraction time of Green Mining profiles. The piece-wise linear interpolation approximation technique allows useful historic green mining energy profiles to be collected much more quickly than the current method. The current method examines each revision of a software product with out approximating data. This work demonstrates

- the time consuming nature of harvesting historic energy consumption information,
- how to trial an energy consumption approximation technique on collected measurements,
- a method for evaluating the new technique against the original, and
- potential areas to explore in future works.

2 Prior Work

Energy consumption needs a unit of measurement and a method for visualizing the measurements. Researchers perform energy measurements using Joules, a unit of measure from the International System of Units (SI) that provides a measurement of energy consumed. Watts, another SI unit, provides a measure for the rate at which work is done and the rate of energy consumption. Watts per second are equivalent to Joules. Researchers can visualize captured energy consumption measurements as watts over time. Visualization of captured energy consumption measurements can help with data analysis when identifying which part of hardware or software had an interesting influence on energy consumption.

Figures 1 and 2 depict collected energy consumption measurements, otherwise known as energy profiles, sorted by software revision. Figure 1 shows an energy profile for the Android Fennec web browser application. The Fennec energy profile took 33 hours to extract. Figure 2 shows an energy profile for the Android calculator application[1] which took 268 hours to extract. The energy profile measurements are collected by creating a set of robust tests for a software product. These tests are run repeatedly while energy consumption is measured. Hindle *et al.* [6] introduced the methodology for extracting these energy profiles and explored the instrumentation required to get empirical results from testing.

[1] By Xlythe on GitHub, for Cyanogen an open source Android mobile operating system. https://github.com/Xlythe/android_packages_apps_Calculator

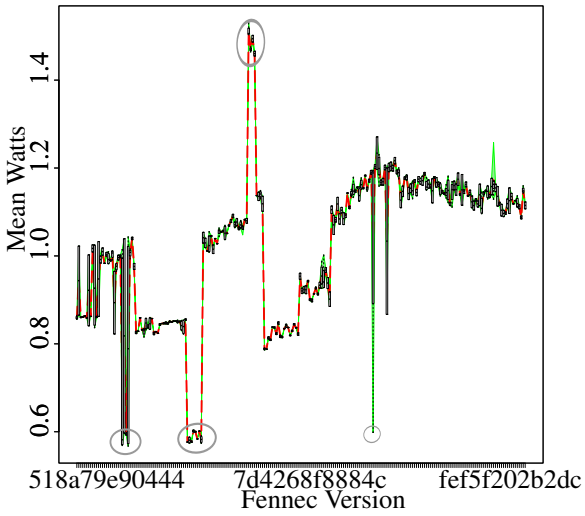

Figure 1: Fennec Green Mining Model

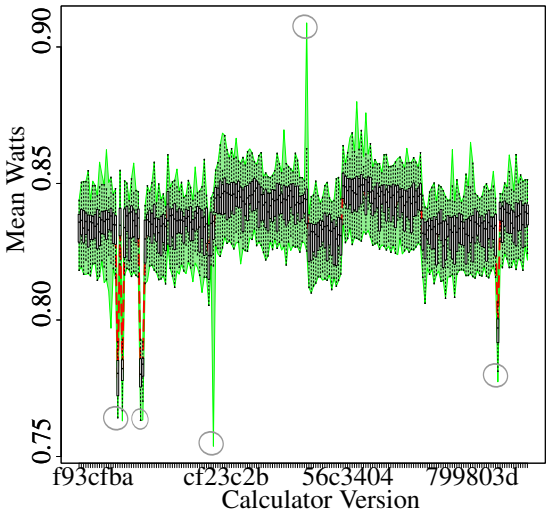

Figure 2: Calculator Green Mining Model

Green Mining [6] was proposed to investigate the relationship between software change and software energy consumption changes. The relationship between software energy consumption change and software change is studied by extracting energy profiles from software applications like Fennec, shown in Figure 1, and Calculator, shown in Figure 2. These energy profiles are then examined with respect to software changes. Green Mining is not currently practiced in industry. An energy profile contains measurements of the mean watts for each software revision examined. Energy profiles enable developers to understand their past performance and regressions. Such energy profiles have been extracted from both desktop [9] and mobile [10] platforms. The energy profiles enable researchers to study and model how software change relates to change in software energy consumption.

The time required to manually extract a comprehensive energy profile is large and cumbersome because in the original Green Mining method all software revisions are tested. Therefore the Green Miner [10] testbed was used to extract the energy profiles in Figure 1 and 2. The Green Miner provides automated energy consumption measurement, testing, and result reporting, reducing the effort, but not the time, required to harvest these measurements. Figures 1 and 2 show box plots depicting the distribution of mean watts and joules consumed when testing the respective Android applications, sorted by software revision. Peaks and troughs were manually annotated. The annotated regions may have spiked or dipped due to poor test or software run-time behaviour such as a thread hanging or failing to exit a loop. If the cause of the variance could be identified in the energy profiles, then tools could be created to monitor and predict the changes in energy consumption that a changeset could introduce.

The Green Miner [10] testbed uses kits consisting of a Raspberry Pi, an Arduino, an Android phone, and an INA219 IC. The kits are used to measure the energy consumption of software under test running on the mobile phone. A power supply is wired through the INA219 IC to the battery contact of the mobile device. When test cases are being performed on the mobile device, the INA219 IC measures current and voltage of the mobile device to produce approximately 50 readings per second. An Arduino receives data from the INA219 IC and relays this information to the Raspberry Pi. Once testing has been completed the Raspberry Pi assembles a tarball of the collected data and sends it to a web service. The web service is responsible for storing data from multiple kits, it is also capable of scheduling test batches and performing data analysis to aid researchers.

To improve the method employed by Green Mining for extracting energy profiles SBSE was investigated. SBSE was proposed by Harman and Jones [8] as a means to solve difficult SE research problems using search-based techniques. Harman

and Jones show how to translate SE problems [8] into search problems by defining three ideas. The first is a representation of the potential solutions to a problem, the second is a fitness function that distinguishes which solution is better between two search results, and the third is a set of operators that can be manipulated to control the results of the chosen search techniques. Harman *et al.* [11] provides a field summary of some of the work done since the initial proposal. Harman *et al.* also encourages the application of experimental search-based techniques to SE fields that SBSE has not previously been applied to, such as software energy consumption profile extraction. This approach was investigated since no prior optimizations had been applied to the Green Mining methodology.

3 Method

The original Green Mining [6] methodology requires a software product that has a sequence of revisions that are compilable, and a set of tests covering use cases of the software product. An energy profile can be extracted from the software product by running the set of use-case tests on each revision and measuring energy consumption. The result of this process can be seen in Figure 1 and 2. What the figures do not show is how time consuming this energy profile extraction process is. Consider that it may take several minutes to measure the energy consumption of a single software revision with a set of tests. This one revision must then be tested and measured multiple times to reduce the error and increase the accuracy of the energy consumption measurement. For example, if it took 3 minutes to test one revision, and each revision was tested 40 times, and there were 200 revisions in the software product under investigation, it would take 400 hours to extract an energy profile for the product.

To obtain an improvement in the current rate of Green Mining this work focuses on the idea of spending less time by examining fewer revisions. The energy consumption measurements from the skipped revisions is instead approximated. This speeds up the Green Mining methodology by spending less time in the testing phase.

To implement this idea, a variable is defined to control how many revisions are skipped in the approximation technique. The revision control variable is named the *neighbour distance*. If a revision is picked for examination then the next *neighbour distance* revisions are not examined. For example, if the *neighbour distance* is 2 and the set of software revisions is $R = \{0, 1, 2, .., n\}$ then the examined revision E will be $E = \{0, 3, 6, ..\}$. Generally let there exist a set of software revisions $R = \{0, 1, 2, .., n\}$. The revisions to be examined will then consist of the elements E as defined by $E = \{e \mid e \bmod (neighbourdistance + 1) = 0, e \in R\}$

The *neighbour distance* is the closest neighbour to an examined revision. For a neighbour distance of 2, one must travel past 2 neighbours after a revision has been examined. Green Mining where all revisions are explored has a neighbour distance of 0. A function for picking revisions for examination can then be defined which takes a set of revisions and a neighbour distance as parameters. Let the function be,

$$revisionPicker(R, d) = \{e \mid e \bmod (d + 1) = 0, e \in R\}$$

where R is a set of revisions from a software product and d is a chosen neighbour distance.

With a set of examined revisions E, and the knowledge of which revisions were not examined while building E, linear interpolation can be applied to the set $R \backslash E$ to approximate all missing energy measurements from the skipped software revisions. This is done by taking two consecutive revision test results from E and creating a vector from the earlier revision point to the older revision point. The skipped values can then be approximated by linear interpolation based on the distance between that revision and the closest revisions in E.

To understand the accuracy of different neighbour distances, approximation technique output was compared with the complete energy profile of an application. The complete energy profile of an application is the energy profile created from testing every revision of the application. This complete energy profile is the same as the profile obtained from the original Green Mining technique. Multiple approximated energy profiles can be created by altering the neighbour distance parameter of the revision picker function and applying the function to an existing energy profile. The linear interpolation method can then be applied to the revision picker function output sets to create approximated energy profiles for comparison.

Two metrics are used to identify the best neighbour distance for approximating energy profiles for a given amount of effort or time spent: a metric for determining the accuracy of each approximation, and a means to calculate how long it takes to collect the measurements needed to approximate the energy profile. To determine energy profile model accuracy with respect to the original Green Mining technique, which examined all revisions, the root mean squared deviation (RMSD) is calculated between an approximated energy profile and the original energy profile. RMSD is calculated using the following formula,

$$RMSD(\hat{M}, M) = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{m}_i - m_i)^2}$$

where $\hat{m}_i, m_i$ are elements of $\hat{M}$ and M respectively, $\hat{M}$ is the approximated model, M is the original model, and n is the number of elements in M. The approximated profile with the lowest RMSD is the most accurate. The amount of time to generate each approximated energy profile can be calculated from the number of revisions used to create the approximated energy profile.

To extend this method to real scenarios, a partial set of examined revisions, P, is defined from a whole set E such that $P \subseteq E$. The linear interpolation method can then be applied to the partial set of examined revisions P to create an approximated profile. An online procedure for maintaining an approximated green mining profile would be to initially choose a set of revisions to evaluate, E, to choose the earliest and latest revision in E, to measure the revisions, and to add them into P. Revisions from E can be selected and measured, the results can be added to P. At any point in this process the approximate model will be the piecewise linear interpolation of P until $P = E$, calculated the same way as E would be calculated. For the partial approximated energy profiles it is also possible to record the RMSD with respect to the original Green Mining technique and the amount of revisions examined to produce the approximated model. The number of revisions used to create the approximation and the RMSD of the approximation allow the approximated profile to be compared with other energy profiles of the same application.

3.1 Method Discussion

The development of the proposed approximation method was guided by the 3 points used to convert a software engineering problem into an SBSE problem. A representation of the potential solution to the Green Mining speed-up problem was chosen as approximated energy profiles computed via piece-wise linear interpolation. A fitness function, the RMSD, for identifying which approximated energy profile is the most accurate was found and applied. Then an operator to manipulate the results of the approximated energy profile generating function was defined as the neighbour distance.

If revisions are picked in order when extracting partially approximated energy profiles the RMSD values will be skewed. For example, if there are 100 revisions and revisions 1, 5, and 9 have been selected to build an approximated energy profile, then there will be approximated measurements for revisions 1 through 9, and no measurements for 10 through 100. Therefore to better fit the approximated energy profile to the whole energy model of a given software product, a custom examination order function is created given a set of revisions to be examined. The first 2 revisions examined by this function are the first and last revisions respectively. In the previous scenario where 100 revisions were present, revisions 1 and 100 would be examined and linear interpolation would approximate values for revisions 2 through 99. The ordering function then takes the middle revision of the examination set and adds it to the examination order. Using the middle revision, the examination set is split into two subsets. The same procedure is then applied to the new subsets 4 times until 15 middle revisions have been picked. 15 revisions was found to give reasonable results. Consider the example of 100 revisions with a neighbour distance of 2, the order of revisions evaluated would be first be: $\{1, 100, 49, 73, 85, 91, 79, 61, 67, 55, 25, 37, 43, 31, 13, 19, 7, ...\}$. When using a higher number than 4 in the bisection routine there was little visible change in the RMSD value of the approximated energy profiles. Therefore, any remaining untested revisions in the E set would be examined in order to smooth out the approximated model from left to right. Partial approximated energy profiles in this work are all generated following this custom examination ordering method.

Prior to this method, git bisect, a binary search like method, was explored to approximate energy

profiles. Git's bisect function is used to find a bug introducing revision in a software repository. Given the property that revisions before the bug can be considered bug-free, and revisions after the bug are considered to be buggy, the revisions can be sorted. The motivation for considering this method is that it can be used to identify revisions that introduce significant changes in a software product's energy consumption. Energy consumption changes introduced by new revisions are independent of one another. If a revision introduced an energy consumption increase, it may or may not ever be fixed in a later revision. However, a later revision may introduce an energy consumption decrease for an unrelated reason. The later revision would hide the revision that introduced the energy consumption increase, which would prevent the software revisions from being labeled with respect to the original revision's energy consumption increase being fixed. Thus, a binary search could not be applied to the software revisions since the original energy consumption increasing code may not have been fixed in the later revision. Instead of binary search, local estimates, such as piece-wise linear interpolation, that exploit the performance stability of the software are used.

4 Case Study

The approximation technique was applied to 2 Android applications to determine which neighbour distances give the highest time savings and accuracy. The Mozilla Fennec Android web browser application was mined in 2 experiments prior to this work. Fennec test batch 1 used 228 unique software revisions to test energy consumption while actively reading a web page, and Fennec test batch 2 used 252 unique software revisions to test energy consumption while reading a web page and idling on the web page. The Android calculator application was found and 201 unique revisions were built and tested for this work. On each set of green mining energy profiles a neighbour distance of 1 to 20 was applied in the approximation technique to create partial and full approximated models.

4.1 On Mining Android Calculator

The Fennec Android application was mined prior to this work. A small criteria was created to pick a software application for Green Mining to be examined in addition to Fennec. The software application needed to be a FLOSS Android application, the software had to have more than 150 revisions in its version control system, the software system needed a very small set of third party dependencies

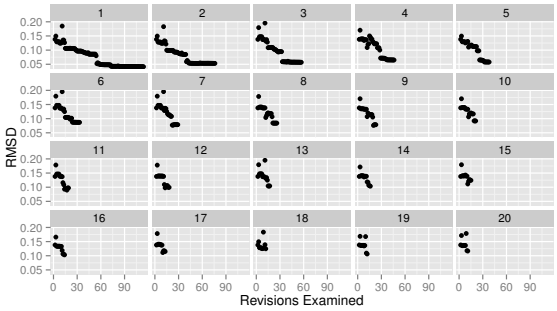

Figure 3: Fennec RMSD vs Total Revisions Examined using neighbour distance 1 to 20 on test batch 1

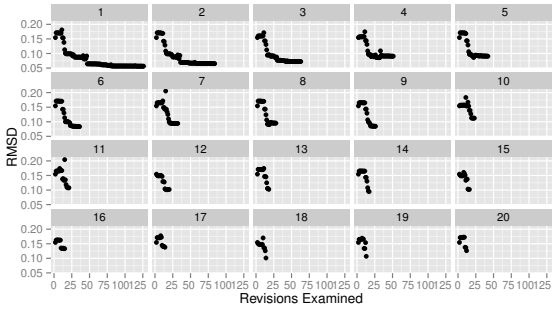

Figure 4: Fennec RMSD vs Total Revisions Examined using neighbour distance 1 to 20 on test batch 2

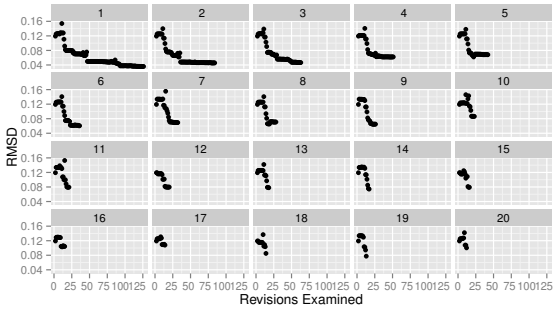

Figure 5: Idle Fennec RMSD vs Total Revisions Examined using neighbour distance 1 to 20 on test batch 2

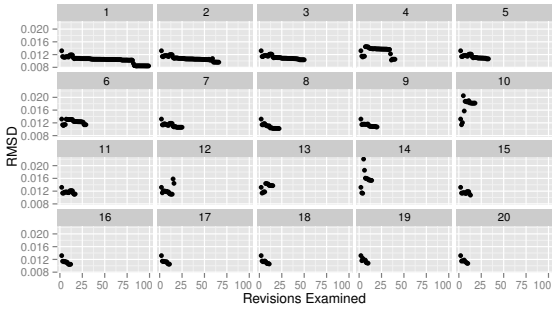

Figure 6: Calculator RMSD vs Total Revisions Examined with Neighbour Distances 1 to 20

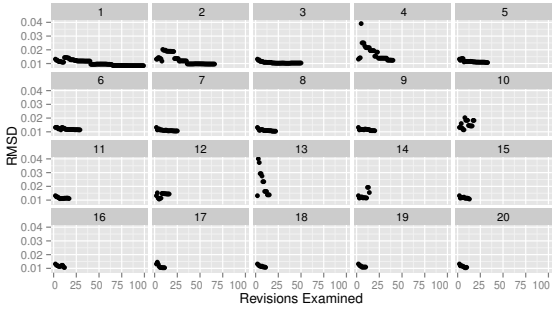

Figure 7: Randomized Ordered Calculator Examination RMSD vs Total Revisions Examined with Neighbour Distances 1 to 20

(if any), and the software needed to take less than 10 minutes for the first author to figure out how to compile a revision of the application. The FLOSS requirement gives researchers access to the software changesets that are associated with the application that is going to be mined, which are needed to identify any relationship between the software changesets and energy consumption change. The 10 minute or less compile rule was used because many FLOSS applications were often not in an easily buildable state.

These criteria were used to find 3 Android applications only 1 of which was chosen. The calculator application presented the simplest user interface for writing test cases out of the three applications found and was therefore picked for Green Mining. A set of test cases were written using the Android monkeyrunner tool for the calculator. Each compilable consecutive revision of the calculator application was then compiled into an `apk` file (a package format for the Android operating system) and sent to the Green Miner testbed with the test cases. Green Miner ran the test cases 40 times on each revision of the calculator application, and once completed it consolidated all relevant measurements into a tarball for collection.

4.2 Android Application Test Cases And Results

Fennec batch 1 and batch 2 both ran a reading test. This involved simulating a user reading a Wikipedia page on the American Idol TV show with the browser. Fennec batch 2 ran an idle browser usage test in addition to the reading test, simulating a user not interacting with the web page. Fennec is a web browser, thus it is quite complicated so two kinds of tests were used.

5 test cases were created for the calculator application and run consecutively. These tests included a gallons to litre conversion, a miles to kilometers conversion, a US dollars to Canadian dollars conversion, a sales tax calculation given an un-taxed price, and applying the quadratic equation to solve a simple quadratic. The calculator tests were written by an author of this work, thus they may not represent an "average" case. On the other hand, the use cases cover the case where users are swiping screens on the application, as well as pressing buttons on the calculator which are two interactions a user will likely do while using the calculator. The

calculator tests were run in the order listed with the following assumptions: it will take a user 2 seconds to slide to a new panel on the device, it will take 0.9 seconds for a user to press a new button in the calculator application, and it will take the user 3 seconds to clear the calculator display after performing a calculation.

4.3 Results

Figures 3 through 7 show approximated energy profiles for each neighbour distance of 1 to 20. Each dot on one of the subgraphs represents an approximated energy profile, the x axis shows how many revisions were used to approximate the profile and the y axis shows the calculated RMSD using the model. A lower RMSD corresponds to a more accurate approximation. A higher number of revisions means additional time was needed to approximate an energy profile. Figure 7 examined the first and last revision of the calculator application then applied the Fisher-Yates shuffle algorithm to randomize the neighbour distance revision examination order. Each of these graphs show the RMSD value for each partially approximated energy profile. The number of examined revisions is shown on the x-axis. The number of examinations can be converted to time spent by multiplying by the time required to run one batch of tests and how many times the test must be repeated. Readers will notice that as the neighbour distance increases, RMSD increases, and thus, accuracy decreases. As a larger number of revisions are measured, RMSD decreases in the majority of profiles. Comparing Figure 6 and 7 shows that the custom examination order function performs better than a random ordering on the calculator application. The custom examination order function is better because it has smaller RMSD values in Figure 6.

From looking at the figures it is clear that a lower neighbour distance, more examinations, leads to a better RMSD at the cost of far more testing time. *Pareto efficiency* is the property of optimal resource allocation; any energy profile with this property in the case study will have the best RMSD given that time spent. The Pareto efficiency graph or Pareto frontier was created by plotting all measurements from a test batch on one graph instead of 20. The plotted points were then ordered by the time taken to extract each energy profile, then removing any non-monotonic decreasing points with respect to RMSD from the graph. The Pareto efficiency was calculated from all the approximate energy profiles created from the measurements shown in Figures 3 and 6 and depicted in Figures 8 through 11.

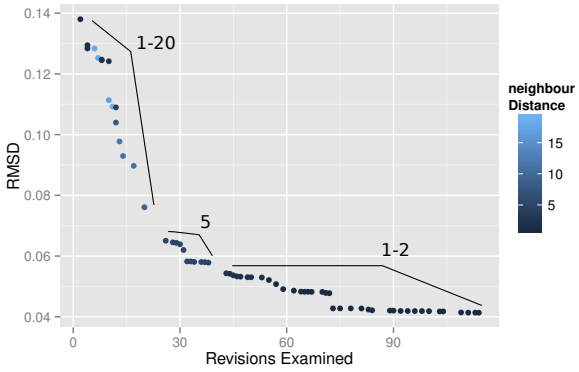

Figure 8: Test Batch 1: Fennec Reading Pareto front of accuracy versus time used while varying the Neighbour Distance between tested revisions

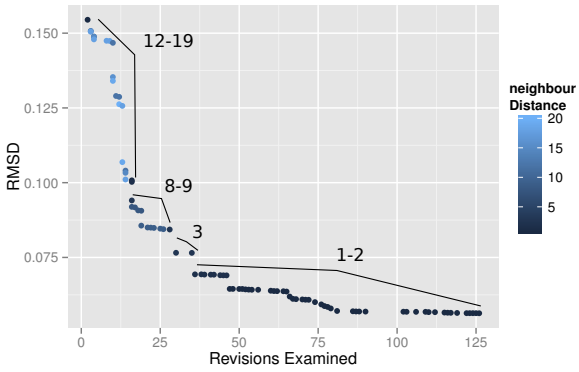

Figure 9: Test Batch 2: Fennec Reading Pareto front of accuracy versus time used while varying the Neighbour Distance between tested revisions

Figures 8 through 12 show Pareto frontiers for the approximated energy profiles. The plotted points are coloured based on the neighbour distance used to create the approximated energy profile. In addition to this, the Pareto fronts have been annotated further with labels of the neighbour distances used to create each section of approximated energy profiles.

The Pareto front graphs show that a smaller neighbour distance will give better accuracy. Specifically, a neighbour distance between 1 and 3

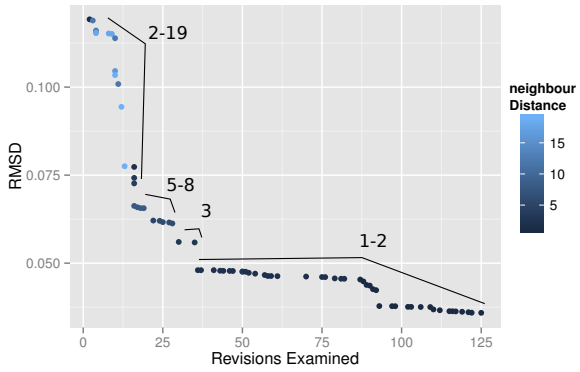

Figure 10: Test Batch 2: Idle Fennec Reading Pareto front of accuracy versus time used while varying the Neighbour Distance between tested revisions

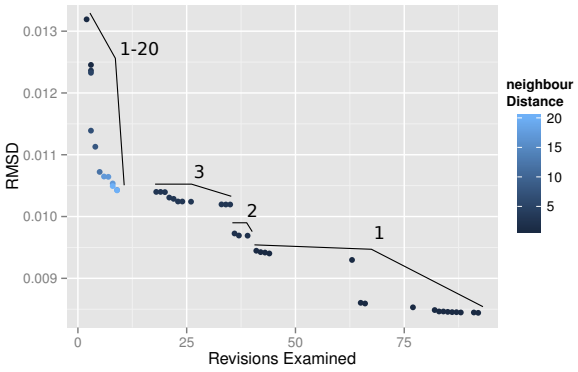

Figure 12: Randomized Ordered Calculator Examination Pareto front of accuracy versus time used while varying the Neighbour Distance between tested revisions

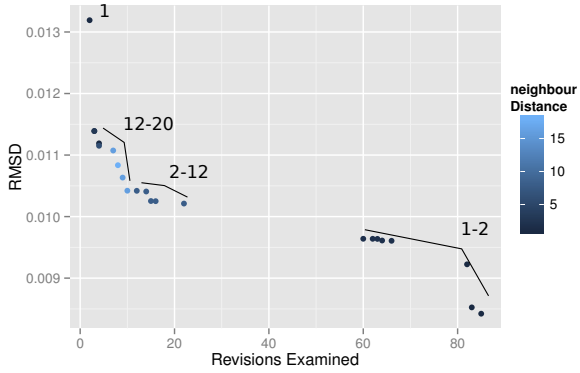

Figure 11: Calculator Pareto front of accuracy versus time used while varying the Neighbour Distance between tested revisions

give the best accuracy using the proposed method. A second range of neighbour distances appears to occur using the neighbour distances from 5 to 8 with additional gain in RMSD but a larger time savings. It is also interesting to note the difference between the Fennec Pareto fronts and the calculator Pareto front: the calculator Pareto front has fewer points and has a large gap between the energy profiles generated with a neighbour distance greater than 2 and the group generated with a neighbour distance less than or equal 2. The cause of this gap in the calculator Pareto front most likely stems from the smoother nature of the full energy profile in Figure 2 and the nature of linear interpolation. In comparison, the energy profile for the Fennec application in Figure 1 has its concavity change more sporadically and has a larger variance in mean watts. Therefore, its approximated energy profile continuously improves with each additional examination. Figure 11 compared to 12 shows that the custom examination ordering function performs better than a randomized ordering function; a Pareto efficiency is reached sooner with the custom examination order.

5 Threats to Validity

Construct Validity Unit tests written for the calculator application pose a threat to construct validity. The unit tests attempt to emulate typical user interaction with the Android application. However, no actual user interaction data was collected before hand. Therefore the unit tests may not produce representative results with respect to real usage scenarios. The work does not seek to show improvements to the average use case energy consumption of the calculator application. Instead, the work demonstrates the improvement to the methodology, which is focused on the energy profile model versus the approximated energy profile model. If this case study was looking to improve the efficiency of the calculator application in the average use case then creating better unit tests would be more important to proving the validity of the improvements.

The source code coverage of the unit tests also poses a threat to construct validity. If the unit tests do not cover enough of the code then they may

not reflect software changes being introduced by the developers. Poor unit test coverage would impact data analysis when checking for correlation between software change and energy consumption change. This work did not focus on comparing source code change to energy consumption change, however; instead the performance of the approximation method on an energy profile was the focus. Future works that explore code feature and energy consumption change relationships should keep test coverage in mind.

Internal Validity The internal validity of the work is threatened by treating the commits from each project as equal. Furthermore, this data is drawn from operational data (data created to produce software) and not from experimental data. There is an assumption made that software applications have common source code between revisions. Thus, cumulative changes to energy consumption could be identified with the approximation technique using a smaller neighbour distance for higher accuracy.

External Validity The Android applications were run only on a mobile platform. This hinders the ability to generalize results from their energy profiles to other hardware platforms. For instance, storage access may impact energy consumption differently on the mobile platforms compared to desktop platforms. This work applied the proposed approximation method to 2 software applications. This threatens external validity with respect to sample bias and generalizing the result to other software applications.

Reliability Collecting energy consumption data creates risks with respect to reproducibility. The usage of the Green Miner improves the consistency of the collected energy consumption data. Repeatedly running tests is done to reduce error introduced by the environment of the mobile device.

6 Conclusion

From the case study, a neighbour distance of 1-3 could be recommended to a researcher who wants to approximate an energy profile for a software product with the highest accuracy while still saving time. A recommended neighbour distance of 5-8 could be used when a researcher is more interested in the general shape of an energy profile instead of fine granularity. An example of a low granularity task is when a researcher is looking for a large increase or decline in the energy consumption of an application. Furthermore, with low RMSD for neighbour distances between 1 and 4, it is obvious that not all revisions (a neighbour distance of 0) need to be tested because software products under test often show stable energy consumption. If a large jump in energy consumption is observed more testing could be done in that range of revisions. This recommendation answers the proposed research question of how few software revisions can be tested while extracting an energy profile of a given product.

With this approximation technique and set of recommendations, Green Mining now has a fast technique for energy profile extraction while minimizing accuracy loss. If a researcher used a neighbour distance of 3 to generate their approximated energy profile they would save 75% of the original extraction time. If the researcher used a neighbour distance of 8 they would save 89% of the original extraction time. The proposed method allows a larger range of software revision sets to be studied because the time required by the testing process is reduced. This work also provides evidence that energy profiles can be quite stable in small regions of 1-4 revisions.

This work demonstrates how to test a new Green Mining technique with existing energy consumption measurements and how to harvest new energy consumption measurements. This work also demonstrates how to compare a technique's results using RMSD, and the time taken to produce an approximation with the approximation technique. The work also produces some insight for future optimizations, as no prior optimizations have been attempted on the Green Mining technique.

This advancement of Green Mining may lead to future energy conscious development tools. With energy consumption measurements that can be more easily collected and studied, researchers may be able to better identify software and energy consumption change relationships. The change relationships could then be applied during software development to aid developers in the development of more energy efficient software.

6.1 Future Work

Effectiveness of Machine Learning on approximating energy profiles If an energy profile for

a software product is unstable, with its energy consumption constantly increasing and decreasing with each additional commit, then an approximation by linear interpolation may not work well. Therefore, the question is raised, "can machine learning be applied to previously collected green mining energy profiles?" If so, can a trained learner be used on software changesets to predict how a software changeset will influence the energy consumption of a software system? If not a machine learning approach, are there other static analysis techniques that can be applied to a software code base, the code changesets introduced in a repository, and the energy consumption change measurements that are associated with the code changesets?

Machine learning would need a feature that is extractable from software changesets that has a relationship with energy consumption. The feature could be used to predict whether or not a changeset will influence energy consumption. Such an application could reveal important energy consumption altering revisions.

Energy debugging More investigation is needed into turning these results into a consistent and comprehensive software energy debugging process. Such a process would enable developers to help debug their energy bugs and save time. Helping developers discover and locate energy bugs would help them write more efficient applications.

The effect of churn Some potential features extractable from code changesets are code churn and software language models. These features would then need to be compared against energy consumption measurements to check for a relationship between the feature and energy change. More potential code changeset features can be investigated for whether or not they are useful with machine learning.

The effect of modularity The modularity of a system could affect the performance of a green mining approximation technique. Some modules might be more correlated with energy consumption than others. For instance, database modules might be relevant to the amount of IO performed. Furthermore, API change might be a predictor of a change in a green mining profile.

References

[1] S. Murugesan, "Harnessing green IT: Principles and practices," *IT professional*, vol. 10, no. 1, pp. 24–33, 2008.

[2] Alliance to Save Energy, "PC Energy Report 2007: United States," 2007.

[3] B. Steigerwald and A. Agrawal, "Developing Green Software," tech. rep., Technical report, Intel Corporation, 2011.

[4] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pp. 2–10, IEEE, 1999.

[5] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, pp. 29–42, ACM, 2012.

[6] A. Hindle, "Green mining: A methodology of relating software change to power consumption," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pp. 78–87, IEEE, 2012.

[7] A. Hindle, "Green mining: Investigating power consumption across versions," in *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 1301–1304, IEEE, 2012.

[8] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[9] C. Zhang, *The Impact of User Choice and Software Change on Energy Consumption*. PhD thesis, University of Alberta, 2013.

[10] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: a hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 12–21, ACM, 2014.

[11] M. Harman, "The current state and future of search based software engineering," in *2007 Future of Software Engineering*, pp. 342–357, IEEE Computer Society, 2007.