ABSTRACT

CHEN, JIANFENG. On the Value of Sampling and Pruning for Search-Based Software Engineering. (Under the direction of Timothy Menzies).

Software Engineering researchers use search-based software engineering (SBSE) optimization techniques to solve SE configuration problems with single or multiple objectives; e.g. (1) selecting partial features to develop to stratify the shareholders or (2) reducing development cost, or (3) minimizing testing suites while improving coverage, etc.

Most previous researchers in this area applied evolutionary algorithms or their variants, EVOL for short in this research, to those tasks. Due to the complexity of SE models, their algorithms are excessively CPU intensive.

This research explores an alternative approach – OSAP, short for over-sampling-and-pruning. OSAP performs the oversampling and prunes the configurations, without evaluating all candidates. OSAP has four generations, named OSAP1, OSAP2, and so on.

The first generation of OSAP, OSAP1, a.k.a. SWAY, assumed that a small region in decision space can lead to best objectives. Therefore, it separated the decision space by some specific separation metrics to prune poor regions. When applied to two widely-explored SE requirement engineering models – XOMO and POM3, SWAY generated new state-of-the-art results. SWAY also returned comparable configurations to another baseline method, which just selects the best set from large amount of randomly generated configurations.

Some models may not hold that assumption of the SWAY, such as the software product line (SPL) or next release planning (NRP). The SPL/NRP optimization requires to find the best set of features (to develop) under some constraints. The second generation of OSAP, OSAP2, a.k.a. SWAY^2, first divides the configurations into several groups, and then applies the SWAY in each group.

While successful in many domains, SWAY was proved to fail for more complex problems. For example, cloud computer configuration problems defeat SWAY since their best configurations do not gather in some regions in decision space. Based on an analysis of those failures, this research developed a novel variant, the OSAP3, or RIOT, that builds up a linear surrogate model to evaluate sampling and therefore leads to a faster pruning. On experimentation, RIOT outperforms not only SWAY but also methods proposed by numerous other researchers for cloud computing configuration.

The success of RIOT has mainly relied on the linear surrogate models. To get accurate linear surrogate models, the SE problems must have some linear relations between configurations and the objectives. Unfortunately, not all SE problems have such features. In view of this, this research proposes the fourth generation of OSAP, a.k.a. the WORTHY. WORTHY builds up the surrogate models to determine whether decision change(delta) is promising, without holding the aforementioned linear relations presumption. This research explored WORTHY in two case studies. The first case

study revisited the XOMO/POM3 models and the second study was the test suite generations for symbolic models (expressed as 3-SAT CNF). Experiments showed that the WORTHY outperformed the OSAP1, or had better results than state-of-the-art techniques.

OSAP provides software engineers an efficiency toolkit to solve the search-based software engineering problems, especially for the large scale tricky problems. Also, the success of OSAP offers a new research direction for exploring search-based software engineering models. In the meanwhile, it shows that the success of EVOL, or its variant may come from the evaluations of large number of candidates, instead of the better mutations.

On the Value of Sampling and Pruning for Search-Based Software Engineering

by
Jianfeng Chen

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2019

APPROVED BY:

Min Chi	Emerson Murphy-Hill
Xipeng Shen	Timothy Menzies Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

Jianfeng Chen was born in Dongguan City, Guangdong Province in 1992. In 2014, he obtained his a B.Eng. Degree in Software Engineering from Shandong University China, where he worked closely with Dr. Yijun Yang in Research Center of Human-Computer Interaction & Virtual Reality. He started his PhD program from August 2014 in department of computer science, North Carolina State University. He joined Dr. Tim Menzies's lab in 2015. During his PhD, he had an internship in Facebook (2018, Cambridge MA) and successfully participated in the Google Summer of Code program (2016), in the team of Java Pathfinder (JPF). He also had cooperation experiences with the Laboratory for Analytic Sciences (NCSU) as well as LexisNexis Legal & Professional (LNG-RDU).

ACKNOWLEDGEMENTS

Throughout my journey of my PhD program, I have received a great deal of support and assistant – this is *not* an individual work.

I would first like to think my advisor, Dr. Tim Menzies, whose expertise was invaluable in pointing out my research directions in particular. Dr. Tim Menzies is a nice and experienced academic advisor. He could always give me some pressure in a correct time and in the suitable way, although sometimes his words contained some metaphors. I especially appreciated Dr. Menzies's efforts in preparing my Oral Prelim Exam. At that time, he spent a lot of time in revising my lecture as well as the slides. Through that exam, my presentation skill was significantly improved. (Apparently, of course,) I still need to strengthen my communication skills. As the last point, I would like to say thank you for Dr. Menzies's research fundings, which came from the NSF, LAS or LexisNexis etc. I am also grateful to my PhD committee members, Dr. Min Chi, Dr. Emerson Murphy-Hill and Dr. Xipeng Shen, for all the time spent, all the help and advice on my research and career.

I would like to thank every (current or former) members in RAISE lab. They are Dr. Wei Fu, Dr. Vivek Nair, Di Chen, Rahul Krishna, George Mathew, Zhe Yu, Amrit Agrawal, Patrick Xia, Huy Tu, Rui Shu, Xueqi Yang, Shrikanth NC, Joy Chakraborty, Fahmid Fahid and Dr. Junjie Wang, our distinguished visiting scholar from Chinese Academy of Science. I had very insightful discussions with Dr. Vivek Nair, Zhe Yu etc. Rahul and George answered many of my silly programming questions. Patrick saved me a lot of money in travelling and entertainments... Some of them were also my coauthors – Vivek, Rahul, Patrick, Junjie, et al. We worked and shared our opinions around the Room 3240 at EBII. Every one in the lab helped me a lot in different ways. So thank you all very much!

I am grateful to my new friends in NC State University. Here is a partial list of them: Shengpei Zhang, Rui Zhi, Wengran Wang, Hao Pang, Zexi Chen, Boxuan Zhong, Rong Huang, Jerry Lee, Bowen Li, Gina R. Bai, Weijie Zhou, Justin Smith, Zhewei Hu, Shuai Yang, Pei Deng, Xin Pan, Chin-Jung Hsu, Answesha Das, Lingnan Gao, Akond Rahman, Ye Mao, Yiqiao Xu, Linting Xue, and so on. Thanks for all kinds of helps or supports in life or academic.

I also want to thank the partners during my intern or co-op programs. They are Neha Rungta – principal engineering at Amazon AWS, Aaron Roth – former Facebook engineer, Vlad Bychkovsky and Jenny Ramseyer – Facebook researchers, Philip Clark and Kevin Haverlock – LexisNexis engineers, Bojan Cukic – Chair of CS department in UNCC, etc. They brought me the insights from industrial world. In the meanwhile, I miss the friendship with colleagues during my Facebook Intern. They are Yi-Hsiu Chen, Allison Tielking, Macros Banchik, Stephanie Angulo, Anthony Simpson, Karen Guo, etc. Thanks all for shring the funs in summer 2018.

I extend heartfelt thanks to the distinguished faculties and staffs at the department of computer science, including George Rouskas, Kathy Luca, Carol Allen, Todd Gardner, etc. Thanks for the helps

in these years!

Here I am grateful to Dr. Yijun Yang, a professor in School of Computing, Shandong University. He advised me a lot during my undergraduate. Thanks for encouraging me to apply for the PhD in North American.

Finally, last but not least, I am very thankful to my family and friends, who have been beside me, supporting me and helping me at all times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Software engineers often need to answer questions that explore trade-offs between competing goals. This is particularly true when stakeholders propose multiple goals or requirements and software developers need to find good choices that most reflect and balance rival objectives such as:

1. What smallest set of test cases that cover all program branches?
2. What is the set of requirements that balances software development cost and customer satisfaction?
3. What cheapest resources achieve most functionality?
4. What sequence of refactoring steps take least effort while most decreasing the future maintenance costs of a system?

As modern software grows increasingly complex, it becomes difficult (or impossible) to manually find these good choices. Hence, in recent years, there has been an increasing interest in search-based software engineering, or SBSE (details see §3.2). SBSE often uses multi-objective evolutionary algorithms (EVOL) [Har12; Say13a] that explore generations of mutations to a population of candidate solutions. Examples of this kind of analysis include:

- *Software product line optimization*: Sayyad et. al. [Say13b] compared several EVOL, including SPEA2, NSGA-II, IBEA, etc, and found that IBEA algorithm performed best in generating valid products from product line descriptions (for details see §6.3.2).

- *Project planning*: Ferrucci et al. [Fer13] modified the crossover operator in the NSGA-II algorithm and found that their approach (called NSGA-II$_v$) was useful for planning how to make best use of project overtime.

- *Test suite minimization*: Wang et al.[Wan13a] showed that their "weighted-based" genetic algorithm significantly outperformed other methods using industrial case study for Cisco Systems.

- *Improving defect prediction*: Tantithamthavorn et at. [Tan16] report that software quality predictors learned from data miners can be improved if an evolutionary algorithm first adjusts the tuning parameters of the learner.

- *Software clone detectors*: Wang, Harman et al. [Wan13b] report that the arduous task of configuring complex analysis tools like software clone detectors can be automated via multi-objective evolutionary algorithms.

A drawback with standard EVOL is that they can be very computational expensive (see §4.1). This can make them problematic to apply, especially for complex problems. For example, in the above list, the last two studies required 22 days and 15 years of CPU time respectively.

One way to address this CPU cost is to use cloud-based CPU farms. The advantage of this approach is that it is simple to implement (just buy the cloud-based CPU time). But cloud-based resources have some disadvantages: (a) cloud computing environments are extensively monetized so the total financial cost of tuning can be prohibitive; (b) that CPU time is wasted time if there is a faster and more effective way.

This research explores a faster and more effective way to solve problems traditionally addressed by EVOL. Our new approach offers a novel prospective on SBSE research. It is *not* supposed to explore *specific* algorithms or techniques for some SE model. *Instead*, it explores the decision space *(the space formed by all possible configurations in a SE problem)* and the objective space *(space of the corresponding objectives)* in general software engineering models. This new method is called a over-sampling-and-pruning methods, or OSAP for short. OSAP assumes that by (a) generating many candidates; but (b) only evaluating a few of them, it is possible to retrieve good configurations for the SBSE model in a very short time. this research explores four generations in total.

1.1 Statement of Thesis

For the optimization of search based software engineering problems:

- Given a proper configurations selector or comparator, built upon decision space,

- Over-sampling and pruning is better than a standard mutation based evolutionary approach;

- Where "better" is measured in terms of runtimes, number of evaluations and value of final results.

1.2 Generations of OSAP

This research developed four generations of OSAP, labelled as OSAP1, OSAP2, OSAP3, OSAP4 in current dissertation.

1.2.1 First Generation of OSAP, a.k.a. SWAY

OSAP1 is the initial version of OSAP. The configuration selector of OSAP1 is based on a simple (but maybe rough) assumption: the best configuration for a software engineering problem only appears in a small region of the decision space. Imposing this assumption, the OSAP1 further develops a (decision) separator that bi-clusters the configurations. OSAP1 pruned one cluster among them based on the evaluations of two representatives, one in each cluster. The pruning process is performed recursively under a small region is found. The method is also named as SWAY in the published paper.

1.2.2 Second Generation of OSAP, a.k.a. SWAY^2

OSAP2 is an improved version to OSAP1. It first requires the expert knowledge to divided the decision space. For example, in the requirement engineering problems, such division can be the size/scale of proposed configuration, measured by the number of features to be developed, etc. After dividing the decision space from expert knowledge, the SWAY is executed in each sub decision space. The optimal/desired configuration is the union set in each sub decision space. The OSAP2 is also named as SWAY^2 in the publications.

1.2.3 Third Generation of OSAP, a.k.a. RIOT

The SWAY series algorithms could not precisely select the optimal configurations, since the aforementioned assumption may not be true. OSAP3 adopts another configuration comparator. To faster compare and select the samples, this work creates a surrogate model. Please note that all surrogate model mentioned in this dissertation are built separately for every objective *(if the problem is multi-objective problem)*. The surrogate model in OSAP3 first set up some random anchors, which are required to evaluate by the original SE model, and then it estimates the objective of all other samples via the *linear* interpolation of the nearest anchor to furthest anchor. With the help of surrogate model, OSAP3 can quickly estimate any configurations, therefore build up a comparator/selector to pick up the optimal configurations. OSAP3 is also known as RIOT in the publications.

OSAP OUTLINES

OSAP1 (SWAY)

- *What was achieved:* SWAY first found that sampling-and-pruning method can get similar results to mutation based evolutionary algorithm, in a much faster way.

- *What was learned:* The evolution in EVOL may not be helpful. Given appropriate pruner, a small region in decision space could contain the optimal configurations.

- *Limiting assumptions:* Only valid for simple models which have the "golden" (decision space) region.

OSAP2 (SWAY^2)

- *What was achieved:* Fixed OSAP1 via doing the decision space partition first, using the domain / expert knowledge.

- *What was learned:* Utilizing the domain / expert knowledge could be helpful.

- *Limiting assumptions:* domain / expert knowledge is required; not flexible enough. To some extent, the "golden" region assumption still exists.

OSAP3 (RIOT)

- *What was achieved:* Created a surrogate model that directly estimate the objectives of the configurations, leading to effective sampling and pruning.

- *What was learned:* Building a surrogate model to abridge the decision space and objective space is possible. Such surrogate model could be used as an alternative to replace the complex SE model.

- *Limiting assumptions:* The surrogate model was based on the linearity of the model.

OSAP4 (WORTHY)

- *What was achieved:* Created another more flexible surrogate model without the dependence of linearity of the model.

- *What was learned:* Decision + configuration Δ oriented surrogate models better explore more complex SE models.

- *Limiting assumptions:* The model contain no or less randomness.

1.2.4 Fourth Generation of OSAP, a.k.a. WORTHY

The effectiveness of OSAP3 is highly depending on the the "linear" interpolation surrogate models. To remove the "linearity" assumption, OSAP4 directed built up the a surrogate model learning the decision space δ-vector of every pair of some evaluated configurations. The k-nearest neighbor(KNN) can be used as the surrogate model. The OSAP4 is also named as WORTHY in the papers.

1.3 Study Cases Overview

The motivation for this work is not to solve a particular case study; instead, it explores an alternative framework to the slow evolutionary algorithms(EVOL). Therefore, OSAP should be tested in diverse software engineering problems. By saying the "diverse", it meant the following perspective:

- Problems appears from different stages in software development process;
- Covering the problems with no constraints, slight constraints, and highly constraints;
- The decision space has various types, such as numeric, enumerations, discrete, binary etc.

It is impossible to explore all search-based software engineering problems in a single research. The following study cases were explored and discussed in this dissertation.

> Note: Statements on how the following study cases meets the diversity perspectives will be given at the end of this dissertation (§9.3).

1.3.1 XOMO and POM3

XOMO, introduced by Menzies et al. [MR05], is a general framework for Monte Carlo simulations that combines four COCOMO-like software process models from Boehm's group at the University of Southern California.

POM3 model is a tool for exploring the thorny management challenge in agile development[BT03b; Por08; BT03a]– balancing *idle rates*, *completion rates* and *overall cost*.

1.3.2 Software Product Line (SPL)

The software product line (SPL) model was also explored. SPL is a collection of related software products, which share some core functionality [Har14a]. From one product line, many products can be generated. For example, Apel et al. [Sie12] model the compilation configuration parameters of databases as a product line. By adjusting those configurations, a suite of different database solutions can be generated.

1.3.3 Next Release Problem (NRP)

An other study case is the next release problem (NRP) come across in requirement engineering. NRP concerns with defining which requirements should be implemented for the next version of the systems, according to customer satisfaction, budget constraints as well as precedence constraints between various requirements. Durillo et al. [Dur09], treated the next release problem as a multi-objective problem, since higher customer satisfaction and less development time or cost are conflicting objectives, we call this formulation as multi-objective NRP (MONRP). MONRP in this paper considers (maximizing) combination of importance and risk, (minimizing) cost and (maximizing) satisfaction.

1.3.4 Workflow Deployment in Cloud Environment

Scientific workflows (a.k.a data-intensive workflow) have been widely applied in scientific research, data mining and business intelligence analysis [Vöc11]. One fundamental problem in workflow research is *workflow scheduling*, i.e. associating the appropriate computer resource to each task in the workflow.

1.3.5 Test Suite Generation

Theory proving is one of basic techniques in software testing. Given a program, modern symbolic execution [Bal18] and/or dynamic symbolic execution techniques [Chr16] are able to generate the constrained SAT models, expressed as CNF (conjunctive normal form) or DIMACS [BB17] form. Each valid solution is corresponding to a unique feasible test case in a program. Consequently, searching for solutions to the constraint model constructs a test suite to the corresponding program.

1.4 Publications from this Thesis

Conferences

- [Under Review] **Jianfeng Chen** and Tim Menzies. "On the Benefits of Restrained Mutation: Faster Generation of Smaller Test Suites" Submitted to IEEE/ACM International Conference on Automated Software Engineering (ASE 2019).
- [CM18] **Jianfeng Chen**, and Tim Menzies. "RIOT: A Stochastic-Based Method for Workflow Scheduling in the Cloud." 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018.
- [Nai18] Vivek Nair, Amrit Agrawal, **Jianfeng Chen**, Wei Fu, George Mathew, Tim Menzies, Leandro Minku, Markus Wagner, and Zhe Yu. "Data-Driven Search-based Software Engineering." The Mining Software Repositories (MSR) 2018.

- [Nai16] Vivek Nair, Tim Menzies, and **Jianfeng Chen**. "An (accidental) exploration of alternatives to evolutionary algorithms for sbse." In International Symposium on Search Based Software Engineering, pp. 96-111. Springer, Cham, 2016.

Journals

- [Che18] **Jianfeng Chen**, Vivek Nair, Rahul Krishna, and Tim Menzies. "" Sampling" as a Baseline Optimizer for Search-based Software Engineering." IEEE Transactions on Software Engineering (2018).
- [Xia18](Minor revision) Tianpei Xia, Rahul Krishna, **Jianfeng Chen**, George Mathew, Xipeng Shen, and Tim Menzies. "Hyperparameter optimization for effort estimation." Empirical Software Engineering (EMSE), 2018
- [Che17] **Jianfeng Chen**, Vivek Nair, and Tim Menzies. "Beyond evolutionary algorithms for search-based software engineering." Information and Software Technology (2017).

1.5 Structure of this Dissertation

The rest of this dissertation is structured as follows.

- Chapter 2 motives the whole research, answering a frequently asked question – "why we need to replace the EVOL?"
- Chapter 3 formulates the search-based software engineering, including problem definition, evaluations of solvers and statistical ranking for various solvers.
- Chapter 4 is the overview of two strategies in solving SBSE: evolutionary algorithm and oversampling-and-pruning.
- Chapter 5 to Chapter 8 introduce four generations of OSAP. Each of them is tested by one or more study cases.
- Chapter 9 concludes the dissertation.

CHAPTER

2

WHY SBSE NEED FASTER, SIMPLER METHODS

2.1 What is the Value of Seeking Simplicity?

At the end of this dissertation, we will see that OSAP does not necessarily produce optimal optimizations. We advocate its use since it is very simple and very fast. But *what is the value of reporting simple and faster ways to achieve results that are currently achievable by slower and more complex methods?*

In terms of core science, we argue that the better can we understand something, the better we can match tools to SE. Tools which are poorly matched to task are usually complex and/or slow to execute. OSAP seems a better match for the tasks explored in this paper since it is neither complex nor slow. Hence, we argue that OSAP is interesting in terms of its core scientific contribution to SE optimization research.

Seeking simpler and/or faster solutions is not just theoretically interesting. It is also an approach currently in vogue in contemporary software engineering. Calero and Pattini [CP15] comment that "redesign for greater simplicity" also motivates much contemporary industrial work. In their survey of modern SE companies, they find that many current organizational redesigns are motivated (at least in part) by arguments based on "sustainability" (i.e. using fewer resources to achieve results). According to Calero and Pattini, sustainability is now a new source of innovation. Managers used sustainability-based redesigns to explore cost-cutting opportunities. In fact, they say, sustainability is now viewed by many companies as a mechanism for gaining complete advantage over their competitors. Hence, a manager might ask a programmer to assess methods like OSAP explored in

current work as a technique to generate new and more interesting products.

2.2 Why not just Use More of the Cloud?

OSAP reduces the number of evaluations required to optimize a model (and hence the CPU cost of this kind of analysis by one to two orders of magnitude). Given the ready availability of cloud-based CPU, we are sometimes asked *about the benefits of merely making something run 100 times faster when we can just buy more CPU on the cloud?*

In reply, we say that CPUs are not an unlimited resource that should be applied unquestionably to computationally expensive problems.

- We can no longer rely on Moore's Law [Moo98] to double our computational power very 18 months. Power consumption and heat dissipation issues effectively block further exponential increases to CPU clock frequencies [Kum03].

- Even if we could build those faster CPUs, we would still need to power them. CPU power requirements (and the pollution associated with generating that power [Thi14]) is now a significant issue. Data centers consume 1.5% of globally electrical output and this value is predicted to grow dramatically in the very near future [Bro08] (data centers in the USA used 91 billion kilowatt-hours of electrical energy in 2013, and they will be using 139 billion kilowatt-hours by 2020 (a 53% increase) [Thi14]).

- Even if (a) we could build faster CPUs and even if (b) we had the energy to power them and even if (c) we could dispose of the pollution associated with generating that energy, then all that CPU+energy+pollution offset would be a service that must be paid for. Fisher et al. [Fis12] comment that cloud computation is a heavily monetized environment that charges for all their services (storage, uploads, downloads, and CPU time). While each small part of that service is cheap, the total annual cost to an organization can be exorbitant. Google reports that a 1% reduction in CPU requirements saves them millions of dollars in power costs.

Hence we say that tools like OSAP, which use less CPU, are interesting because they let us achieve the same goals with fewer resources.

2.3 OSAP as a Baseline Optimizer

2.3.1 Why Researchers Need Baseline Algorithm

Experienced researchers endorse the use of baseline algorithms. For example, in his textbook on *Empirical Methods for AI*, Cohen [Coh95] strongly advocates comparing supposedly sophisticated systems against simpler alternatives. In the machine learning community, Holte [Hol93] uses the OneR baseline algorithm as a *scout* that runs ahead of a more complicated learner as a way to judge the complexity of up-coming tasks. In the software engineering community, Whigham et

al. [Whi15] recently proposed baseline methods for effort estimation (for other baseline methods in effort estimation, see Mittas et al. [MA13]). Shepperd and Macdonnel [SM12] argue convincingly that measurements are best viewed as ratios compared to measurements taken from some minimal baseline system. Work on cross versus within-company cost estimation has also recommended the use of some very simple baseline (they recommend regression as their default model) [Kit07].

2.3.2 Principles for Baseline Algorithms in SBSE

In their recent article on baselines in software engineering, Whigham et al. [Whi15] propose guidelines for designing a baseline implementation that include:

1. Be *simple* to describe and implement;
2. Be *applicable to a range of models*;
3. Be *publicly available* via a reference implementation and associated environment for execution;

To their criteria, we would add that for multi-objective optimization algorithms, such baselines should also:

4. Offer *comparable performance* to standard methods. While we do not expect a baseline method to out-perform all state-of-the-art methods, for a baseline to be insightful, it needs to offer a level of performance that often approaches the state-of-the-art.
5. *Not be resource expensive to apply* (measured in terms of required CPU or number of evaluations). The resources required to reach a decision are not a major concern for Whigham's cost estimation work. Before a community adopts SBSE baseline methods, we must first ensure that baseline executes very quickly. Some search-based software engineering methods can require days to years of CPU-time to terminate [Wan13b]. Hence, unlike Whigham et al., we take care *not* to select baseline methods that are impractically slow.

2.3.3 Can OSAP be a Baseline Optimizer?

OSAP satisfies all the above criteria. The method is straightforward:

- Generate a very large population of random candidates;
- Evaluate a small number of representative candidates;
- Cull any candidates that are near the poorly performing representatives.

Note that this uses much less machinery than a standard genetic algorithm; i.e., there are no complex selection, mutation or crossover operators. Nor does OSAP employ multi-generational reasoning. As a result, it is a simple matter to code OSAP (see pseudocode in Algorithm 1).

As to being *applicable to a wide range of models*, in this paper we apply OSAP to models with boolean and continuous decision variables:

- Our models with continuous decision variables are XOMO and POM3. XOMO [Men07; Men09b; Men09a] is an SE model where the optimization task is to reduce the defects, risk, development months, and the total number of staff members associated with a software project. POM3 [BT03b; Por08; BT03a] is an SE model of agile development towards negotiating what tasks to do next within a team.

- Our model with boolean decision variables is software product lines [Say13a; Say13c] for which the optimization task is to extract (a) valid products that (b) have more features and use (c) the most familiar features that (d) costs the least to implement and which (e) has the fewest known bugs.

As to *public availability*, a full implementation of OSAP including all the case studies presented here (including working implementations of other multi-objective evolutionary algorithms and our evaluation models in [Che17; Che18]) is available online.

In terms of *comparative performance*, for each model, we compared OSAP's performance against the established state-of-the-art method as reported in the literature. In those comparative results, OSAP was usually as good, and sometimes even a little better, than the state-of-the-art.

OSAP is also *not resource expensive to apply*. The algorithm does not evaluate all of its random candidates. Instead, OSAP employs a top-down bi-cluster procedure that finds two distant points X, Y, then labels all points according to which of X, Y they are closest to. The points are then evaluated, and all points near the worst one are culled. Hence, OSAP only evaluates $log(N)$ of N candidates, which makes it a relatively very fast algorithm.

Such reduction in runtime is an important feature of OSAP since some optimization studies can be very CPU intensive. For example, recent EVOLstudies in software engineering by Fu et al. [Fu16] and Wang et al. [Wan13b] required 22 days and 15 years of CPU time respectively. While, to some extent, this high CPU cost can be managed via the use of cloud computing services, those computing environments are extensively monetized so the total financial cost of tuning can be prohibitive. We note that all that money is a wasted resource if there is a more straightforward way (e.g., OSAP) to achieve similar results.

OSAP offers many benefits to practitioners and researchers:

1. Researchers can use results of OSAP as the "sanity checker". Experiments showed that results of OSAP are much better than random configurations and in most times, it is comparable to standard optimizers. Consequently, when designing new optimizers, researchers can compare their results to OSAP's to see whether their superiority is significant.

2. Practitioners can use OSAP to get quick feedback on their adjustments. For example, in agile development, managers can apply OSAP to POM3 to quickly get approximate changes of developing efforts or costs when they adjust release plans or team personnel, etc.

CHAPTER

3

PROBLEM FORMULATION

3.1 Software Configuration Optimization

Throughout the software engineering life cycle, from requirement engineering, project planning to software testing, maintenance and re-engineering, software engineers need to find a balance between different goals such as:

- *Software product line optimization*: find proper configurations for a software product line which can fulfill most features as well as with less known defects and less costs, etc.
- *Project planning*: find the best planning to reduce software project duration, overtime and the risk [Fer13].
- *Test suite minimization*: identify and eliminate redundant test cases from test suites, and reduce the total number of test cases to execute, thereby improving the efficiency of testing [Wan13a].

All of these problems can be viewed as *optimization problems*; i.e. tune the *configuration* parameters of a model such that, when that model runs, it generates "good" *objective* (i.e. output demonstrably better than other possible outputs). However, given the complexities of software engineering, SE models are often too complicated to prove that an output is optimal. For such models, the best we can do is run multiple optimizers and report the best output seen across all those optimizers.

In the past, due to the simplicity of software structure, developers/experts could make a decision based on their empirical knowledge. For such models of such simple knowledge, it may have been

possible to demand that those outputs are "optimal"; i.e. there exists no other configuration such that a better output can be generated. However, modern software is becoming increasingly complex. Finding the optimal solution to such kind of problems may be difficult/impossible. For example, in a project staffing problem, if there are 10 experts available and 10 activities to be accomplished, the total number of available combinations is 10 billion (10^{10}). For such large search spaces, exhaustively enumerating and assessing all possibilities is clearly impractical.

3.2 Search Based Software Engineering (SBSE)

In **search-based software engineering (SBSE)**, the software engineering problem is treated as a mathematical model: given the numeric (or boolean) configurations/decisions variables, the model should return one or more objectives. In a nutshell, the model can convert *decisions* "*d*" into *objective* scores "*o*", i.e.

$$o = model(d)$$

The direction of optimization for the objectives can be either to maximize or minimize their values. For example, in software engineering, we might want to maximize the delivered functionality while also minimizing the cost to make that delivery. If model delivers just one objective, then we call the this a *single-objective optimization problem*. On the other hand, when there are many objectives we call that a *multi-objective optimization problem*.

For the multi-objective optimization problem, often there is no "d" which can minimize (or maximize) all objectives. Rather, the "best" d offers a good trade-off between competing objectives. In such a space of competing goals, we cannot be optimal on all objectives, simultaneously. Rather, we must seek a *Pareto frontier* or solution of multiple solutions where no other solutions in the frontier "dominates" any other [ZT99].

There are two types of dominance– *binary dominance* and *continuous dominance*. Binary dominance is defined as follows: solution x is said to binary dominate the solution y if and only if the objectives in x is partially less (larger when the corresponding objective is to maximize) than the objectives in y, that is,

$$\forall o \in obj\ o_x \succeq o_y \text{ and } \exists o \in obj\ o_x \succ o_y$$

where *obj* are the objectives and $(\succeq, \succ)$ tests if an objective score in one individual is (no worse, better) than the other.

Continuous dominance, as defined by [ZK04], favors y over x if x "losses" least:

$$\begin{aligned} x \succ y &= loss(y,x) > loss(x,y) \\ loss(x,y) &= \textstyle\sum_j^n -e^{\Delta(j,x,y,n)}/n \\ \Delta(j,x,y,n) &= w_j(o_{j,x} - o_{j,y})/n \end{aligned} \tag{3.1}$$

Sayyad and Menzies argue that continuous dominance works better than binary domination when the number of objectives grows larger than three [Say13b].

3.3 Measuring the Efficiency

To compare the efficiency of different SBSE optimizers, we used Runtime and Number of evaluations as the measures.
(M1) Runtime: The execution time from one algorithm starts to the termination of that algorithm.
(M2) Number of evaluations: Number of model evaluations during the whole problem optimization process.

The reason why we did not merely use M1 is that sometimes comparing the running time is not enough. Although all of our method were coded in the same language–python, the implementing details bring the bias. Note that the software engineering models are extremely large, and the evaluation process occupies significant part of the runtime.

3.4 Measuring the Effectiveness

3.4.1 Effectiveness Measuring in Single-objective Problem

For the single objective problem, literately there exist some optimal configuration that maximize/minimize the objective. Due to the complexity of software engineering models, it may be difficult to get the theoretical optimal configuration.

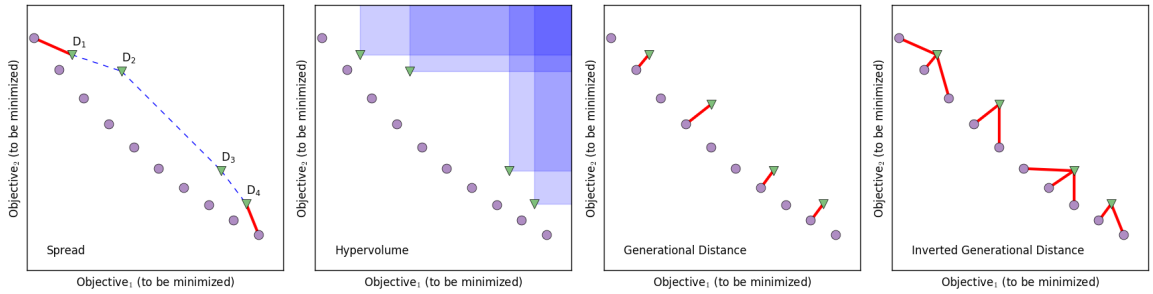
Therefore, in the real practice, when comparing the effectiveness over several algorithms, one can treat the optimal configuration found over all experiment repeats as the global optimal configuration. The difference between obtained configuration and optimal configuration should be reported as the *relative* diff values, instead of the *absolute* diff.

3.4.2 Standard Metrics for Multi-objective Problems

To compare the effectiveness for multi-objective problems, i.e. quality of obtained frontiers, of different optimizers, Wang et al. [Wan16] provided the following quality indicators. Figure 3.1 illustrates these quality indicators. To formulate them, here we first define PF_c and PF_0. PF_c is the Pareto front obtained by an algorithm while PF_0 is the optimal Pareto Front for a specific problem. Please note that in SE models, obtaining the Pareto Front is in-feasible in practice [Deb01]. Thus, we collected all solutions found by all algorithms and picked up all non-dominated solutions to form the PF_0. This strategy is widely applied in the area of previous EVOL applications.
(M3) Generated Spread (GS): GS [Zho06] is a diversity indicator. It is defined to extend the quality indicator *Spread* which only works for bi-objective problems. GS can be calculated by

$$GS = \frac{\sum_{k=1}^{m} d(e_k, PF_c) + \sum_{s \in PF_c} |d(s, PF_c) - \bar{d}|}{\sum_{k=1}^{m} d(e_k, PF_c) + |PF_c| * \bar{d}} \tag{3.2}$$

where $(e_1, e_2, \ldots, e_m)$ refers to m extreme solutions for each objective in PF_0; $d(*, \dagger)$ refers to the minimum Euclidean distance from solution $*$ to the set $\dagger$; $\bar{d}$ is the mean value of $d(s, PF_c)$ for all



(a) Generated Spread (GS)

(b) Hypervolume (HV)

(c) Generational Distance (GD)

(d) Inverted Generational Distance (IGD)

Figure 3.1 Schematic diagram of frontier quality measures can applied in this research.

solutions in PF_c. A lower value of GS shows that the results have a better distribution.

(M4) Pareto front size (PFS): PFS is another diversity indicator. It measures the number of solutions that are included in PF_c, i.e. $|PF_c|$. A higher PFS means that the users have more options to choose from, that is, a more diverse obtained Parto front.

(M5) Hypervolume (HV): HV is the combination of convergence and diversity indicator. As defined in [ZT99], HV measures the size of space the obtained frontier covered. Formally,

$$\text{Hypervolume} = \lambda\left(\bigcup_{s\in PF_c}\{s'|s \prec s' \prec s_{\text{ref}}\}\right) \tag{3.3}$$

where $\lambda(.)$ is the Lebesgue measure, the standard way measures the subset of n-dimensional Euclidean space. For example, Lebesgue measure is the length, area or volume when $n = 1,2,3$ respectively; $\prec$ is the binary domination comparator; s_{ref} denotes a reference point that should be dominated by all obtained solutions.

(M6) Generational Distance (GD): GD is a measure of convergence. It is the Edclidean distance between solutions in PF_c and nearest solutions in PF_0 [VL98]. It can be calculated by

$$GD = \frac{\sqrt{\sum_{i=1}^{|PF_c|} d(s_i, PF_0)^2}}{|PF_c|} \tag{3.4}$$

where $d(s_i, PF_0)$ refers to the minimum Euclidean distance from solution s_i in PF_c to PF_0. A lower GD indicates the result is closer to the Pareto frontier of a specific problem. A value of 0 means that all obtained solutions are optimal.

(M7) Inverted Generational Distance (IGD): IGD is another measure of convergence. It measures both convergence as well as the diversity of the solutions-measures the shortest distance from each solution in the Actual PF_0 to the closest solution in Predicted PF_c. Like Generational distance, the distance is measured in Euclidean space. In an ideal case, IGD is 0, which means the predicted PF_c is same as the actual PF_0.

3.4.3 Problem-Specified Measurements

For some models, researchers may define problem-specified measurements. Usually these measures are well defined to reflect the objective people care about. Introduction to such measurements will be introduced in the specified study case if they are applied.

3.5 Choice of Statistical Ranking Methods

Most of optimizers in SBSE are stochastic methods including EVOL and OSAP. To reduce the bias and report the robustness of one optimizer, we repeated all experiments for 30 times.

To rank various methods, all results were studied using non-parametric tests (the use non-parametric for SBSE was recently endorsed by Arcuri and Briand at ICSE'11 [MA13]). For testing statistical significance, we used non-parametric bootstrap test 95% confidence [ET93] followed by an A12 test to check that any observed differences were not trivially small effects; i.e. given two lists X and Y, count how often there are larger numbers in the former list (and there there are ties, add a half mark): $a = \forall x \in X, y \in Y \frac{\#(x>y)+0.5*\#(x=y)}{|X|*|Y|}$ (as per Vargha [VD00], we say that a "small" effect has $a < 0.6$). Lastly, to generate succinct reports, we use the Scott-Knott test to recursively divide our optimizers. This recursion used A12 and bootstrapping to group together subsets that are (a) not significantly different and are (b) not just a small effect different to each other. This use of Scott-Knott is endorsed by Mittas and Angelis in their 2013 TSE article [MA13] and by Hassan et al at ICSE'15 [Gho15].

CHAPTER

4

OVER-SAMPLING-AND-PRUNING

This chapter introduces the framework over-sampling-and-pruning proposed in this research. One core distinction made by this research is between common algorithms – evolutionary algorithms and over-sampling-and-pruning. Therefore, here we first introduce the evolutionary algorithms, and then address our new framework.

4.1 Evolutionary Algorithms

Evolutionary algorithms (EVOL) are widely applied in SBSE community. According to Zhang et al.'s survey [Zha12], up to 2017, 79% of research papers were built on evolutionary algorithms.

EVOL create the initial population first, and then execute the crossover and mutation repeatedly until "tired or happy"; i.e. until we have run out of CPU time limitation or until we have reached solutions that suffice for the purposes at hand.

Figure 4.1 shows the basic framework of EVOL. One simple way to understand EVOL is to compare them with Darwin's theory of evolution. To find good scores for the objectives, start from a group of individuals. As time goes by, the individuals inside the group crossover. The offsprings which have better fitness scores tend to survive (in the selection step). During the evolution, the mutation operation can increase the diversity of the group and avoid the evolution from getting trapped in the local optima.

Two important details within EVOL are the *initialization policy* in step (1) and the *evaluation policy* in step (3c). The standard initialization policy is to build members of the population by selecting, at random, across the range of all known decisions. For some models with constrained configuration space, this standard policy can be naive. One research interest in SBSE community is

1. Generate population $i = 0$ using some *initialization policy*.
2. Evaluate all individuals in population 0.
3. Repeat until tired or happy
   (a) *Cross-over:* combine elite items to make population $i + 1$;
   (b) *Mutation:* make small changes within population i;
   (c) *Evaluate:* individuals in population i;
   (d) *Selection:* choose some elite subset of population i.

Figure 4.1 Basic framework of EVOL

to find proper initial configurations. Some strategy can be inheriting from previous configurations, using professional domain knowledge, etc.

As to the *evaluation policy*, the standard policy is, for each decision, run the underlying model to generate objective scores for those decisions. For some models, such an evaluation policy is confusing, prohibitively expensive, or both:

- Verrappa and Letier warn that "..for industrial problems, these algorithms generate (many) solutions, which makes the tasks of understanding them and selecting one among them difficult and time consuming" [VL11].
- Zuluaga et al. comment on the cost of evaluating all decisions for their models of software/ hardware co-design: "synthesis of only one design can take hours or even days." [Zul13].
- Harman comments on the problems of evolving a test suite for software: if every candidate solution requires a time-consuming execution of the entire system: such test suite generation can take weeks of execution time.
- Krall & Menzies explored the optimization of complex NASA models of air traffic control. After discussing the simulation needs of NASA's research scientists, they concluded that those models would take three months to execute, even utilizing NASA's supercomputers [Kra16].

Another research interest is to find proper algorithm to the specific problem. In practice, EVOLs differ in the implement of selection, mutation, or crossover operations. Some widespread used EVOL are as follows:

- *GA = genetic algorithm*: GA models decisions as string of numbers (or binary symbols). To mate (crossover) two strings, just simply switch part of the strings inside two candidates [Ban98; Hol92].
- *IBEA = the indicator-based evolutionary algorithm*: IBEA is a GA that uses the continuous domination function to prune away worst candidates [ZK04];

- *NSGA-II = nondominated sorting genetic algorithm*: NSGA-II is a GA that uses a non- dominating sorting procedure to divide the solutions into *bands* where $band_i$ dominates all of the solutions in $band_{j>i}$. NSGA-II's elite sampling favors the least-crowded solutions in the better band [Deb00b].

- *SPEA2 = Strength Pareto Genetic Algorithm, version 2*: SPEA2 is a GA that favors individuals that dominate the most number of other solutions that are not nearby (and to break ties, it favors items in low density regions) [Zit02].

- And others such as MOEA/D [Zha07], differential evolution [SP97], particle swarm optimization [Neb09], and many more besides.

Depending on the selection strategy, most MOEAs can be classified into:

- **Pareto Dominance Based:** Pareto dominance based algorithms uses the binary domination to select solutions for the successive generations. These techniques are used in tandem with niching operators to preserve the diversity among the solutions.
  Examples: NSGA-II [Deb00a], PAES [KC99], SPEA2 [Zit01b]

- **Decomposition Based:** Decomposition based algorithms divide the problem into a set of sub-problems, which are solved simultaneously in a collaborative manner. Each sub-problem is usually an aggregation of the objectives with uniformly distributed weight vectors.
  Examples: MOGLS [Jas02], MOEA/D [ZL07]

- **Indicator Based:** Indicator based methods work by establishing a complete order among the solutions using a single scalar metric like hypervolume etc.
  Examples: HypE [BZ11], IBEA [ZK04]

All above algorithms typically evaluate thousands to millions of individuals as part of their execution. A fundamental challenge in engineering and other domains is that evaluation of a solution is very expensive:

- Zuluaga et al. comment on the cost of evaluating all decisions for their models of software/ hardware co-design: "synthesis of only one design can take hours or even days." [Zul13].

- Harman comments on the problems of evolving a test suite for software: if every candidate solution requires a time-consuming execution of the entire system: such test suite generation can take weeks of execution time [Har13].

- Krall et al. explored the optimization of complex NASA models of air traffic control. After discussing the simulation needs of NASA's research scientists, they concluded that those models would take three months to execute, even utilizing NASA's supercomputers [Kra16].

1. Randomly generates numerous **valid** configurations.
2. Assess and prune unpromising division of configurations.
3. (a) Terminate *or*
   (b) Go to step 2 if there are still too many unpromising configurations *or*
   (c) Go to step 1 to further explore configurations near prior outputs

Figure 4.2 Basic framework of OSAP

4.2 Over-sampling and Pruning

This research establishes a framework named over-sampling-and-pruning, or OSAP for short. The osap initially creates far more configurations (then EVOL) first, and then select the promising samples among them with the help of configuration selector or comparator.

Figure 4.2 shows the basic framework of OSAP. In step1, initial configurations is generated. All generated configurations will not be updated or mutated during the whole sampling process. Therefore, the generated configurations must be inside the configuration space, i.e. they should be valid for the model. Typically we generate numerous initial configurations; therefore, we called the framework "over-sampling".

Following "over-sampling", step 2 performs the "pruning". The OSAP adopts some configuration selector or comparator to remove/prune the inappropriate configurations. To serve the selector/-comparator, a small number of model evaluation is inevitable. Fortunately, such selectors/comparators are performed in the decision space. In other words, OSAP does NOT map every sample into the objective space – otherwise, it would be extremely slow. With the selectors/comparators, OSAP next quickly determine the unpromising configurations and drop them.

The geometric learner (including OSAP1 and OSAP2) splits configurations in two-fold. As a result, even we drop half of them, there are still too many awful configurations. In view of this, we need to go back to step 2 and recursively perform the "pruning" process (3b).

With random/diagonal anchors (introduced latter in OSAP3), we can quick assess every single configuration, as a result, drop all poor configurations. In this way, we can just terminate at step 3 (3a).

Finally with the δ-vector oriented surrogate models, the OSAP4 detects the optimal or near-optimal configurations among the neighbors of given config. For such version of OSAP, we can repeat the whole process (step1-3) so that OSAP4 has the opportunity to cover more spaces (step3c).

4.3 EVOL versus OSAP

When introducing OSAP I tell as troy about how nature can not plant the same tree in the same dirt; so nature uses time to explore options (evolution multiple variations). But software is not so

limited. We can run small variants of the same tree in the same space any number of times (magic of simulation). So evolutionary adaption across multiple generations is the natural choice for nature. But in software, we have other choices. e.g over-sampling from the initial generation.

The OSAP differs in EVOL as:

- *Candidate Size:* Size of candidates in OSAP is much larger than that of EVOL, since EVOL would generate new candidates to replace old ones during the evolution; while OSAP only prune/drop poor candidates, but not generate any new candidate.
- *Initialization Policy:* all configuration in OSAP must be valid. Some EVOL methods are equipped with configuration adjust operators to guarantee all delivered configurations are valid; while OSAPis not. To make all configuration in OSAP valid, some constrain solvers might be applied. For example, we use a SAT solver to generate valid configurations of a problem with feature model as configuration space.
- *Crossover and mutation:* NO these reproduce operations in OSAP.
- *Selection:* EVOL requires assessing EVERY configurations; while OSAP does not.

CHAPTER

5

SWAY, FIRST GENERATION OF OSAP

This chapter is based on the publications:

- [Nai16] "An (accidental) exploration of alternatives to evolutionary algorithms for SBSE." In International Symposium on Search Based Software Engineering, pp. 96-111. Springer, Cham, 2016.
- [Che18] "Sampling as a Baseline Optimizer for Search-based Software Engineering." IEEE Transactions on Software Engineering (2018).

Traditional SBSE optimizers (the evolutionary algorithms) starts from a set of initial populations, and creeps towards to the frontiers via evaluating numerous reproductive configurations. In this research, we propose a new perspective. Oversampling and then pruning large scale configurations is enough for optimizing SBSE problems. The prerequisite is that we should have the appropriate separation metrics which match the SE models.

First work to discuss and apply separation metrics was Joseph Krall's geometric active learner [Kra15a]. Krall created a tool called GALE which was an optimizer for SBSE problems. For the completeness of this research, we briefly introduce this algorithm. For more details and experiments on that, please see Krall's paper [Kra15a].

GALE is still following the basic framework of common MOEAs–starting from initial population and exploring further generations. However, inside each generation, GALE does not evaluate every candidates, instead, it has a geometric learner, or decision space separator named **where** to divide the candidates into leaves and then **mutate** solutions towards the better end. **where** first locates two extreme representatives among all candidates. Next, a simple projection is performed (Figure 2.

Line 2-11 at [Kra15a]) to split all candidates into two parts and only single part with better extreme representative[1] is preserved. The GALE recursively utilized the **where** to split large amount of candidates into small leaf. **mutate** is only performed in that leaf.

When transforming GALE from Java into Python, we accidentally disabled the **mutate**. To our surprise, we found that the quality of the results were similar to the GALE with **mutate** operator. That is, the evolution in GALE except the last generation can be omitted. Therefore, the configuration separator **where** in GALE is helpful. Since the splitting operator in GALE is very fast, we can enlarge the size of initial population and set up the number of generation as one. We called our new method as SWAY, the Sampling Way. Next, we introduce more details of SWAY.

5.1 Implementations of SWAY

In short, SWAY over-samples and prunes large amount of random configurations. It recursively clusters the candidates and chooses the superior cluster. Unlike the common MOEA algorithms whose candidates are improved by multiple generations of mutation, crossover and selection, the SWAY just picks up small superior candidates among a group of candidates. Consequently, the first step of SWAY is to generate large amount of candidates (we generated 10,000 candidates in our experiments). After generating the initial candidates, they won't be perturbed. SWAY just picks up the promising candidates through clustering and pruning.

If we cluster the candidates through their objectives, we need to evaluate all candidates (similar to common MOEA algorithms), SWAY would be very slow, since model evaluations in many SE problems are extremely time-consuming (see §3.2). Fortunately, the SWAY clusters the candidates by their decisions.

However, clustering through the configurations will lead to a challenge: this requires the model have high correlation between genotype (decision) - phenotype (objective) spaces – Actually, this is common in SE applications. For example, the cloud environment configuration meets such requirement– improve the number of VM/memories can get better QoS but also scarify the budget [Ard14]. The XOMO model, as well as software product line models, also meet such requirement (see results latter in this research).

Algorithm 1 on page 24 shows the general framework of SWAY. The candidates are split into two parts according to their decisions. Then SWAY prune half of then basing on the objectives of the corresponding representatives, where the limited number of model evaluations come from. The SPLIT function may differ for different types of decision representing and we will discuss the SPLIT function very soon. More details of Algorithm 1 on page 1:

- If the population size is smaller that some threshold, then we just return all candidates (line 1). Otherwise, SWAY splits the candidates into two parts, "west side" and the "east side"
- After that, lines 6 and 7 compares representatives of the sides. SWAY uses different methods to find those representatives.

[1] Two parts are preserved if two representatives can not dominate each other.

Algorithm 1: SWAY

Input : items – The candidates
Output : pruned results
Parameter : enough – The minimum cluster size
Require Func : SPLIT
BETTER

```
1 if numberOf(items) < enough then
2 |   return items
3 else
4 |   Δ1, Δ2 ← ∅, ∅
5 |   [west, east], [westItems, eastItems] ← SPLIT(items)
6 |   if ¬BETTER(west, east) then Δ1 ← SWAY(eastItems)
7 |   if ¬BETTER(east, west) then Δ2 ← SWAY(westItems)
8 |   return Δ1 + Δ2
```

- Prune the candidates based on a comparison of the representatives. If neither representative is better, then we SWAYon each part.

SWAY is a divide-and-conquer process. Let the number of candidates be N. It is not difficult to prove that the complexity of the candidate evaluation is $O(\log N)$ [Cor09].

The decision spaces in SE models have various types of representations, such as continuous/discrete arrays, graph/tree based structures, permutations, etc. The SPLIT function is designed according to different types. In this research, we introduce two SPLIT functions, one for continuous decision spaces, another for the binary. In the future, we will explore more SPLIT functions.

SPLIT clusters the candidate into parts then picks up representatives for each part. For models with continuous decisions, we use the Fastmap heuristic[Pla05; FL95] to quickly split the candidates. Platt [Pla05] shows that FastMap is a Nyström algorithm that finds approximations to eigenvectors.

Algorithm 2 lists the details of SPLIT function. To split the candidates into two parts according to the FastMap heuristic, first pick any random candidate (line 1) and then find the two extreme candidates based on the distances (line 2-3). The DISTANCE used in our case studies is the *Euclidean distance*. All other candidates are then projected onto the line joining the two extreme candidates(line 5-8). Finally, split the candidates into two parts based on their projection in the line.

BETTER function is for comparing the representatives for two halves of the candidates. SWAY use binary domination for individual comparisons.

The "enough" in Algorithm 1 on page 1 controls the size of final cluster. SWAY set it as $\sqrt{N}$, where N is number of total candidates, i.e. 10,000 in our experiments.

5.2 Case Study: XOMO and POM3

To explore and analyze the efficiency of SWAY, we compared SWAY with commonly used MOEA algorithms under XOMO and POM3 benchmarks. In this section, we will briefly introduce these

Algorithm 2: SPLITing decision space

Input : items – The candidates to split
Output : [west, east] – representatives;
[westItems, eastItems] – two parts
Require Func : DISTANCE

```
1  rand ← randomly selected item in candidates
2  east ← furthest item apart from rand // DISTANCE required
3  west ← furthest item apart from east // DISTANCE required
4  c ← DISTANCE(east, west)
5  foreach x ∈ items do
6      a ← DISTANCE(x, west)
7      b ← DISTANCE(x, east)
8      x.d ← (a^2 + c^2 − b^2)/(2c) // cosine rule
9  sort items by x.d
10 eastItems ← first half of items
11 westItems ← second half of items
12 return [west, east], [westItems, eastItems]
```

benchmarks, including the model definition and related research work for each of them, and then the exploration process to several research questions. Finally, we describe the performance measures we used in our experiments.

In selecting case studies to test SWAY, we reflected over the space of model types seen in the SBSE literature. The following is an approximate characterization of those models:

- Model size: large or small;
- Conflicting constraints: many or few;
- Decision types: discrete or continuous.

For our evaluation, we selected models that fall across the range of the above model types. For example:

- The XOMO model discussed below is a much smaller model with continuous-valued decisions and no constraints.
- The POM3 model (that used continuous-valued decisions) since prior work showed that POM3 is very slow to optimize [Kra15a].

Another reason to use the models described below is the existence of prior results from these models [Kra16; Kra15a; Kra14; Lek14; Say13a]. That is, by using the particular models described below, we can compare SWAYto the state-of-the-art.

Note that some consideration was given to using artificially generated models that could better span the space of models size, constraints, and decision types. In prior work, we used such models [Men02; Owe02b; Owe02a; MC97; MC00], but based on feedback from the SE community, we can

scale factors (exponentially decrease effort)	prec: have we done this before? flex: development flexibility resl: any risk resolution activities? team: team cohesion pmat: process maturity
upper (linearly decrease effort)	acap: analyst capability pcap: programmer capability pcon: programmer continuity aexp: analyst experience pexp: programmer experience ltex: language and tool experience tool: tool use site: multiple site development sced: length of schedule
lower (linearly increase effort)	rely: required reliability data: 2nd memory requirements cplx: program complexity ruse: software reuse docu: documentation requirements time: runtime pressure stor: main memory requirements pvol: platform volatility

Figure 5.1 Descriptions of the XOMO variables.

no longer endorse that approach. Specifically, the space of possible artificially generated models is so huge that it is difficult to show that results from any artificial model are relevant to any specific model.

	ranges			values	
project	feature	low	high	feature	setting
	rely	3	5	tool	2
FLIGHT:	data	2	3	sced	3
	cplx	3	6		
JPL's flight	time	3	4		
software	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		
	rely	1	4	tool	2
GROUND:	data	2	3	sced	3
	cplx	1	4		
JPL's ground	time	3	4		
software	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		
	prec	1	2	data	3
OSP:	flex	2	5	pvol	2
	resl	1	3	rely	5
Orbital space	team	2	3	pcap	3
plane nav&	pmat	1	4	plex	3
gudiance	stor	3	5	site	3
	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
	prec	3	5	flex	3
OSP2:	pmat	4	5	resl	4
	docu	3	4	team	3
OSP	ltex	2	5	time	3
version 2	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvol	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				site	6

Figure 5.2 Four project-specific XOMO case studies.

5.2.1 Benchmarks of XOMO

XOMO, introduced in [MR05], is a general framework for Monte Carlo simulations that combine four COCOMO-like software process models from Boehm's group at the University of Southern California. Figure 5.1 lists the description of XOMO input variables (All should be within $[1,6]$). The XOMO user begins by defining a set of *ranges* or a specific *value* of these variables to address his or her real situation in one software project. For example, if the project has (a) *relaxed schedule pressure*, they should set *sced* to its minimal value; (b) *reduced functionalists*, they should halve the value of *kloc* and minimize the size of the project database (by setting *data=2*); (c) *reduced quality* (for racing something to market), they might move to lowest reliability, minimize the documentation work and the complexity of the code being written, reduce the schedule pressure to some middle value– in the language of XOMO, this last change would be *rely=1, docu=1, time=3, cplx=1*.

XOMO computes four objective scores: (1) project *risk*; (2) development *effort*; (3) predicted *defects*; (4) total *months* of development. Effort and defects are predicted from mathematical models derived from data collected from hundreds of commercial and Defense Department projects [Boe00]. As to the *risk* model, this model contains rules that trigger when management decisions decrease the odds of completing a project: e.g., demanding *more* reliability (*rely*) while *decreasing* analyst capability (*acap*). Such a project is "risky" since it means the manager is demanding more reliability from less skilled analysts. XOMO measures *risk* as the percent of triggered rules.

The optimization goals for XOMO are to:

1. Reduce risk;
2. Reduce effort;
3. Reduce defects;
4. Reduce months.

Note that this is a non-trivial problem since the objectives listed above as non-separable and conflicting. For example, *increasing* software reliability *reduces* the number of added defects while *increasing* the software development effort. Also, *more* documentation can improve team communication and *decrease* the number of introduced defects. However, such increased documentation *increases* the development effort. Consequently, XOMO is multi-objective optimization problem. MOEA algorithms can handle this. [Kra15a] and [Lek14] pointed out that the NSGA-II [Deb00b] can solve this problem and return quite good results. In our experiments, we will compare SWAYwith the NSGA-II algorithm in solving XOMO cases.

In our case studies with XOMO, we use four scenarios taken from NASA's Jet Propulsion Laboratory [Men09b]. As shown in Figure 5.2, FLIGHT and GROUND are general descriptions of all JPL flight and ground software while OSP and OPS2 are two versions of the flight guidance system of the Orbital Space Plane.

From Figure 5.2, we can know that the FLIGHT model is more flexible than other cases, that is, the decision space for FLIGHT is larger than the GROUND or OSPs. Similarly, sorting the decision

Short name	Decision	Description
Cult	Culture	Number (%) of requirements that change.
Crit	Criticality	Requirements cost effect for safety critical systems.
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality.
Init. Kn	Initial Known	Number of (%) initially known requirements.
Inter-D	Inter-Dependency	Number of (%) requirements that have interdependencies to other teams.
Dyna	Dynamism	Rate of how often new requirements are made.
Size	Size	Number of base requirements in the project.
Plan	Plan	Prioritization Strategy: 0= Cost Ascending; 1= Cost Descending; 2= Value Ascending; 3= Value Descending; 4= $\frac{Cost}{Value}$ Ascending.
T.Size	Team Size	Number of personnel in each team

Figure 5.3 List of inputs to POM3. These inputs come from Turner & Boehm's analysis of factors that control how well organizers can react to agile development practices [BT03b].

	POM3a A broad space of projects.	POM3b Highly critical small projects	POM3c Highly dynamic large projects
Culture	[0.10, 0.90]	[0.10, 0.90]	[0.50, 0.90]
Criticality	[0.82, 1.26]	[0.82, 1.26]	[0.82, 1.26]
Criticality Modifier	[0.02, 0.10]	[0.80, 0.95]	[0.02, 0.08]
Initial Known	[0.40, 0.70]	[0.40, 0.70]	[0.20, 0.50]
Inter-Dependency	[0.0, 1.0]	[0.0, 1.0]	[0, 50]
Dynamism	[1, 50]	[1.0, 50.0]	[40, 50]
Size	[30, 100]	[3, 30]	[30, 300]
Team Size	[1, 44]	[1, 44]	[20, 44]
Plan	[0, 4]	[0, 4]	[0, 4]

Figure 5.4 Three specific POM3 scenarios.

space of the cases, we have

$$\text{OSP} \approx \text{OSP2} < \text{GROUND} < \text{FLIGHT} \tag{5.1}$$

5.2.2 Benchmarks of POM3, a Model of Agile Development

POM3 model is a tool for exploring the management challenge of agile development[BT03b; Por08; BT03a]– balancing *idle rates*, *completion rates* and *overall cost*. More specifically,

- In the agile world, projects terminate after achieving a *completion rate* of $X\%$ ($X < 100$) of its required tasks;
- Team members become *idle* if forced to wait for a yet-to-be-finished task from other teams;

- To *lower* the *idle rate* and *improve* the *completion rate*, management can hire staff–but this *increase* the *overall cost*.

The POM3 model simulates the Boehm and Turner model of agile programming [Boe00] where teams select tasks as they appear in the scrum backlog. Figure 5.3 lists the inputs of POM3 model. What users feel interested in is how to tune the decisions to:

- increase completion rates,
- reduce idle rates,
- reduce overall cost.

One way to understand POM3 is to consider a set intra-dependent requirements. A single requirement consists of a prioritization *value* and a *cost*, along with a list of child-requirements and dependencies. Before any requirement can be satisfied, its children and dependencies must first be satisfied. POM3 builds a requirements heap with prioritization values, containing 30 to 500 requirements, with costs from 1 to 100 (values chosen in consultation with Richard Turner [BT03b]). Since POM3 models agile projects, the *cost, value* figures are constantly changing (up until the point when the requirement is completed, after which they become fixed). Now imagine a mountain of requirements hiding below the surface of a lake; i.e., it is mostly invisible. As the project progresses, requirements (and their dependencies) becomes visible to the on-looking

Programmers are organized into teams. Every so often, the teams pause to plan their next sprint. At that time, the backlog of tasks comprises the visible requirements. For their next sprint, teams prioritize work for their next sprint using one of five prioritization methods: (1) cost ascending; (2) cost descending; (3) value ascending; (4) value descending; (5) $\frac{cost}{value}$ ascending. Note that prioritization might be sub-optimal due to the changing nature of the requirements *cost, value* as the unknown nature of the remaining requirements. POM3 has another wild card; it contains an *early cancellation probability* that can cancel a project after N sprints (the value directly proportional to number of sprints). Due to this wild-card, POM3's teams are always racing to deliver as much as possible before being re-tasked. The final total cost is a function of:

(a) Hours worked, taken from the *cost* of the requirements;

(b) The salary of the developers: less experienced developers get paid less;

(c) The criticality of the software: mission-critical software costs more since they are allocated more resources for software quality tasks.

In our study, we explore three scenarios proposed by Boehm personnel communication (Figure 5.4). Among them, POM3a covers a wide range of projects; POM3b represents small and highly critical projects and POM3c represent large projects that are highly dynamic, where cost and value can be altered over a large range. According to Lekkalapudi's report [Lek14], the POM3c is the most

complex model among them, or

$$\text{POM3a} < \text{POM3b} < \text{POM3c} \tag{5.2}$$

Similar to the XOMO benchmark, this is also a multi-objective optimization problem. From Lekkalapudi's report, NSGA-II can solve this problem and return quite good results. Consequently, same as the XOMO series, we will compare SWAY with the NSGA-II algorithm in solving POM3 study cases.

5.2.3 Experiment Design

To explore SWAY, we organized our exploration around the following research questions (RQ):

1. **(RQ1)** To what extent is SWAY faster than typical EVOL techniques?
2. **(RQ2)** Can SWAY find the solution with maximized (minimized) objective as other EVOL?

RQ1 questions how fast SWAY is while RQ2 questions whether the results from SWAY are comparable to other EVOL algorithms. Equation (5.1) and (5.2) indicates the order of problem size. In the following, all results will be presented in that size order.

There are many EVOL algorithms or modified EVOL to solve our benchmarks. In the following, we compare SWAY against some arguably state-of-the-art methods. Our reading of the literature is that:

Best prior results for XOMO and POM3 were reported using NSGA-II [Deb00b; Kra15a];

When applying NSGA-II to XOMO or POM3, prior reports [Kra15a] and [Lek14] did not state their control parameters. To adddress that issue:

- We ran a grid search [BB12] for the three parameters: *population size (MU)*, *cross-over probability (CXPB)* and *mutation probability (MUTPB)*.
- Our grid search space was defined as $\{\text{MU} = [100, 120, \ldots, 300]\} \times \{\text{CXPB} = [0.9, 0.8, \ldots, 0.6]\} \times \{\text{MUTPB} = [0.1, 0.15, \ldots, 0.25]\}$.
- We use hypervolume (see §3.4.2) as the quality indicator when grid searching. Here we used hypervolume since it is the combination of convergence and diversity indicator.
- Table 5.1 shows the tuned parameters.

Parameter tuning is necessary for SE exploration, especially in the area of search-based SE. For example, in a very recent FSE'17 paper, Fu et al. [FM17a] found that naive learner, e.g., SVM, with parameter tuning can even outperform complex deep learning techniques.

However, when discussing this work with other researchers and colleagues, one common question is "why grid search?". Our answer is that: "grid search" is simple enough. Even though researchers

Table 5.1 Parameters tuned by grid search for the NSGA-II algorithm in solving XOMO and POM3 cases.

Name	MU	CXPB	MUTPB
OSP	200	0.9	0.1
OSP2	100	0.8	0.2
GROUND	200	0.8	0.15
FLIGHT	300	0.9	0.15
POM3a	300	0.8	0.15
POM3b	160	0.9	0.1
POM3c	200	0.9	0.2

Table 5.2 Parameter configurations overview

Strategy	Algorithm	Pop Size	Crossover Rate	Mutation Rate	Repeat	Termination
EVOL	NSGA-II	See Table 5.1			30	Not improved in 5 gens
OSAP	SWAY	10,000	/	/	30	See Algorithm 1
Brute-Force	Groud-Truth	10,000	/	/	30	Generation 0

† archive size = pop size = 300
* Standard mutate: 0.001; Smart mutate: 0.98
POM3 and XOMO models are optimized by NSGA-II, SWAY and GroundTruth
SPL models are optimized by SATIBEA, SWAY and GroundTruth

created many parameter tuners, for example, Fu [FM17a] applied differential evolutionary optimizer, Arcuri [AF13] applied the central composite design to reduce the number of explored configurations, etc., grid search can cover most of the possible configurations.

In the following, whenever we mention NSGA-II, this will be that algorithm plus the parameters of Table 5.1.

All of our experiments were implemented using the *DEAP* [For12] EVOL Python framework.

The termination of SWAY is controlled by the minimum cluster size. The termination of NSGA-II and SATIBEA can be defined as maximum running time, a number of evolution generations or even manual termination, etc. In this paper, to enable a fair comparison with the state-of-the-art algorithm, we set the termination of existing MOEA algorithm as the point where solution quality does improve for consecutive five generations. Here, quality was measured by combing convergence and diversity using the hypervolume metric (see §3.4.2).

Finally, since there is no mutation or cross-over operations in SWAY, all results are from initial random-generated candidate sets. To address the tradeoff of SWAY, we also used the NSGA-II selection operator to find the Pareto frontier among the set of initial candidates. This calculation was time-consuming since it needs to evaluate all candidates (10,000 in our experiments) and sorted them. In the following, we call this method the GROUNDTRUTH. Table 5.2 concludes all parameters.

5.2.4 Is SWAY Faster than Typical EVOL?

We compared the speed of the algorithms through their runtime as well as the number of model evaluations. Table 5.3 shows the median runtime and evaluation numbers recorded in our experi-

Table 5.3 Median value of runtime and model evaluation numbers from 30 independent runs. Numbers rounded to the nearest percent (so "0%" means "less than 0.005")

	Runtime(seconds)			**# Evaluations**		
	SWAY (R_S)	MOEA (R_M)	$\frac{R_S}{R_S+R_M}$	SWAY (E_S)	MOEA (E_M)	$\frac{E_S}{E_S+E_M}$
osp	3.23	320	1%	18	4630	0%
osp2	3.31	112	3%	16	2432	1%
ground	3.29	388	1%	20	5321	0%
flight	5.62	663	1%	33	10980	1%
POM3a	4.69	450	1%	26	3836	1%
POM3b	5.01	583	1%	32	4532	1%
POM3b	6.33	990	1%	40	8302	0%

ments. As can be seen in all cases, SWAY is faster than the state-of-the-art evaluation algorithms, often by two orders of magnitude (especially for the smaller models at the top of the table). And even for the largest models, SWAY offered some speed up advantages.

Consequently, from Table 5.3, our answer to RQ1 is SWAY usually terminates orders of magnitude faster of the other algorithms used in this evaluation.

5.2.5 Are SWAY's Solutions as good as Other Optimizers?

Figure 5.5 shows boxplot of quality indicators among the 30 independent runs in our experiment (where each run used a different random number seed). In that figure:

- The RAND method is a "sanity check" for our technology. In this approach, N candidates were generated at random and then pruned to a final frontier by discarding any dominated points (domination computed from §4.1). Note that, to select N for this method, we used the strategy of [Oun17]: i.e., set N to the median size of the frontier generated by was used by EVOL.
- The GROUNDTRUTH method generates and evaluates many solutions, then reports the best parts of the Pareto frontier (found using the NSGA-II sorting strategy).
- The SWAY method randomly generates solutions, and then fetch some of them through the sampling strategies. Note that SWAY only evaluates a very small number of solutions.
- The MOEA method refers to the state-of-the-art method defined for each case study. Recall that this state-of-the-art is the grid-search-tuned NSGA-II .

In Figure 5.5, the colors denote a statistical comparison with the state of the art, where the statistical test is a non-parametric Wilcoxon comparisons at a 5% significance level:

- **GREEN**, **RED** denote results at are better, worse (respectively) than the state-of-the-art;
- Results that statistically insiginficantly different to the state-of-the-art are marked in **ORANGE**.

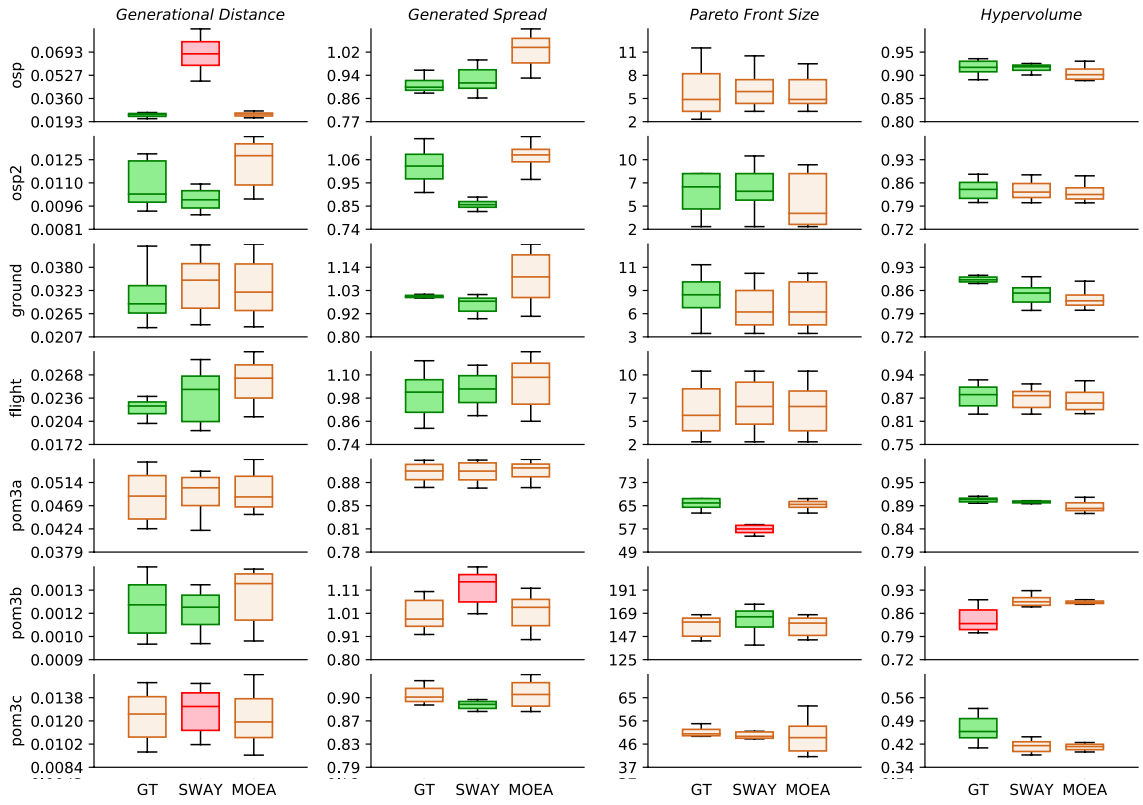


Figure 5.5 The box-plot of four quality indicators in each study cases (30 runs). In each box-plot: **Left:** the GROUNDTRUE method - The frontier of 10k random generated populations ; **Middle:** the SWAYalgorithm; **Right:** the state-of-the-art MOEA for that corresponding study case. Wilcoxon test was performed at a 5% significance level: The **orange box** marks the state-of-the-art method, or method which is no statistical significantly different from state-of-the-art method; **green box** marks the method which is significant better than the state-of-the-art while **red box** indicates that method is worse than the state-of-the-art method. Generational distances is the coverage indicator. Generated spread and pareto front size are the diversity indicators. Hypervolume is the combination of convergence and diversity. Lower values are preferred for generational distance, generated spread while higher values for pareto front size and hypervolume (see text for more details).

Figure 5.6 and Figure 5.7 summarize Figure 5.5 using the same color scheme. As shown in Figure 5.6, GROUNDTRUTH often found best results. At first glance, the results of Figure 5.6 seem to say that all work on heuristic multi-objective optimization should halt since merely making up lots of random solutions performs comparatively very well indeed. However, not shown in Figure 5.6 is the CPU time to achieve those results. Recall from Table 5.3 that SWAY required just under 2 minutes to optimize all the models of this work. By way of comparison, evaluating all the solutions in Figure 5.6 required 1 hours; i.e., that approach was 30 times slower. Figure 5.7 comments on the value of the solutions achieved via that very slow random GROUNDTRUTH method vs SWAY.

n	model	Generational Distance	Generated Spread	Pareto Front Size	Hyper-volume
1	osp	*0.5*	*1.95*	*0.46*	*0.6*
2	ops2	*1.30*	*0.82*	*0.72*	*0.65*
3	ground	*0.79*	*0.86*	*0.71*	*0.30*
4	flight	*1.96*	*0.86*	*0.42*	*0.52*
5	pom3a	*0.79*	*0.53*	*1.30*	*0.85*
6	pom3b	*0.75*	*0.43*	*0.56*	*0.85*
7	pom3c	*0.69*	*0.42*	*0.82*	*1.03*
	same + better	6/7	6/7	7/7	6/7

Figure 5.6 GROUNDTRUTH vs state-of-the-art: How often is ground truth worse, same, or better? Summarized from Figure 5.5. Color patterns are the same as Figure 5.5. Decimal in each cell is the effect size.

n	model	Generational Distance	Generated Spread	Pareto Front Size	Hyper-volume
1	osp	*1.97*	*0.45*	*0.36*	*0.63*
2	ops2	*0.99*	*0.56*	*0.46*	*0.49*
3	ground	*0.93*	*0.60*	*0.46*	*0.64*
4	flight	*0.72*	*0.63*	*0.49*	*0.59*
5	pom3a	*0.67*	*0.50*	*1.93*	*1.58*
6	pom3b	*0.67*	*0.84*	*0.69*	*0.66*
7	pom3c	*1.57*	*0.54*	*1.44*	*1.41*
	same + better	4/7	7/7	3/7	5/7

Figure 5.7 SWAY vs state-of-the-art: How often is SWAY worse, same, or better? Summarized from Figure 5.5.Color patterns are the same as Figure 5.5. Decimal in each cell is the effect size. Generating and evaluating all the models in this figure took the runtimes seen in Table 5.3.

Figure 5.7 summarizes the results achieved by SWAY. In the majority case, across all performance measures, SWAY performs the same or better as the state-of-the-art. Note that these results were achieved with the number of evaluations seem in Table 5.3; i.e. after merely dozens to a few hundred evaluations.

One quirk in Figure 5.5 is that sometimes, very simple *RAND* method achieved comparable generated spread values to EVOL or SWAY. This is due to the nature of solutions in multi-dimensional space. As noted by Domingos, random points in large dimensions space are often very distant [Dom12]. Hence, it is not surprising that a random selection does very well (as measured by spread). Note that achieving good spread scores via random methods says nothing about the *value* of the optimizations achieved via that method (merely the dispersion of those candidates). For a comment on the value of the optimization achieved, see the generational distance and hypervolume results of Figure 5.5 where, as we would expect, RAND performs much worse than other optimizers.

5.2.6 Threats to Validity

5.2.6.1 Optimizer bias

The goal of this case study was not to prove that SWAY is the best optimizer for all models. Rather, our goal was to say that, compared to current practice in the literature, SWAY offers competitive solutions at a small fraction of the evaluation costs of other methods. Hence, we propose SWAY as a reasonable first choice for benchmarking other approaches.

For that goal, it is not necessary to compare SWAY against all other optimizers. Rather, SWAY should be compared against known state-of-the-art in the literature.

5.2.6.2 Internal bias

Internal bias originates from the stochastic nature of multi-objective optimization algorithms. The evolutionary process required many random operations, same as the SWAY introduced in this work.

Table 5.4 Median value of runtime and model evaluation numbers from 30 independent runs. Numbers rounded to the nearest percent (so "0%" means "less than 0.005")

	Runtime(seconds)			**# Evaluations**		
	SWAY (R_S)	MOEA (R_M)	$\frac{R_S}{R_S+R_M}$	SWAY (E_S)	MOEA (E_M)	$\frac{E_S}{E_S+E_M}$
osp	3.23	320	1%	18	4630	0%
osp2	3.31	112	3%	16	2432	1%
ground	3.29	388	1%	20	5321	0%
flight	5.62	663	1%	33	10980	1%
POM3a	4.69	450	1%	26	3836	1%
POM3b	5.01	583	1%	32	4532	1%
POM3b	6.33	990	1%	40	8302	0%

To mitigate these threats, we repeated our experiments for 30 runs and reported the median/ boxplot of the indicators. We also employed statically tests to check the significance of the achieved results.

5.2.6.3 Sampling bias

This paper studied the performance of SWAY using two classes of models: XOMO and POM3. There are many other optimization problems in the area of software engineering, and it is possible that the results of this work will not apply to those models. Future research should explore more models to check the validity of our results.

5.3 Summary of OSAP1

As is indicated in Algorithm 1, the SWAY adopts some configuration clustering strategy to recursively group the candidates and then evaluates the cluster from one representative of each of them. The design of SWAY has following limitations.

- The optimal samples (specifically, the Pareto Frontier for multi-objective models) may not gather in some small region;
- Evaluating the cluster via one representative could be dangerous, i.e. select with prejudice;
- The effectiveness of SWAY is highly depending on the correctness of SPLITTING algorithm, which is difficult to design.

Some of above limitations will be addressed in SWAY^2, while more of them is left to the further version of OSAP.

Even though the SWAY has so many limitations, it is still valuable. The contributions of SWAY includes:

- SWAY offers a new research direction for search-based software engineering area. The success of GROUND method in above case study also showed that to some extent, the power of EVOL comes from large amount of candidate evaluations, instead of the mutation strategy;
- SWAY can be used as the *baseline* method. This will be discussed at the end of next chapter.

CHAPTER

6

SWAY^2, SECOND GENERATION OF OSAP

This chapter is based on the publications:

- [Che18] “Sampling as a Baseline Optimizer for Search-based Software Engineering.” IEEE Transactions on Software Engineering (2018).
- [Che17] "Beyond evolutionary algorithms for search-based software engineering." Information and Software Technology (2017).

When we applied SWAY to higher dimensional problems, such as the product line optimization, SWAY was defected. Therefore, it is necessary to generalize the SWAY. In this research, we called the generalized version of SWAY as SWAY^2. SWAY^2 makes full use of expert knowledge to do the initial splitting before applying the original version of SWAY. This chapter discusses the OSAP2, i.e. SWAY^2.

6.1 When is OSAP1 most Useful, Useless?

SWAY(OSAP1) is designed as a fast substitution of EVOL for solving SBSE problems. It can avoid large amount of model evaluations, which are very common in previous evolutionary algorithms. In view of this, SWAY is particularly useful in following two scenarios.

SWAY would be most useful if it is proposed to put humans-in-the-loop to help guide the evaluations (e.g. as done in [Ree07]). In this scenario, standard EVOL might have to ask a human for up to $O(N^2)$ opinions for G generations. On the other hand, SWAY would only trouble the user

$O(\log N)$ times

Also, SWAY was created to solve problems, where the practitioner is not able to evaluate thousands of individuals. For example Wang et al. [Wan13c] spent 15 years of CPU time to find software clone detectors or model explored by Krall et al. [Kra15b]) which take hours to perform a single evaluation.

However, as discussed later in this chapter, SWAY has two core assumptions. Firstly, it is applicable only when there is a mapping between "genotype" and "phenotype" space; i.e. between the settings to the model inputs and outputs of the model. Even though such mapping may not exist in every model, we find here that for SE models (that were written with the explicit goal of effecting outputs with input decisions), this assumption holds adequately, at least for the purposes of improving model output.

Secondly, SWAY techniques for dividing the data makes the *spectral learning assumption*; i.e. that within the raw dimensions of data seen in any domain, there exists a small set of *spectral* dimensions which can usefully approximate the larger set [Kam03]. While the universality of the spectral assumption has not been proven, it has seen to hold in many domains; e.g. see any data analysis method that uses principle components analysis [AC17; BD16; Shi17; Red16; Li16].

6.2 Initial Splitting with Expert Knowledge

Following shows the principles of SPLIT function to apply expert/domain knowledge.

From Algorithm 1, we note that the purpose of SPLIT is to divide candidates into two parts; a good part and a second poorer part, which is expected to be discarded. To demystify this splitting process, we list several principles of SPLIT function as follows:

1. Split must run over the information available *before* candidate evaluation; i.e. split must run over the decision space instead of objective space;
2. After SPLIT similar candidates should be clustered together.
3. Further, dissimilar, or opposite candidates should be separated into different clusters.
4. For each subspace of candidates' decision space, candidates are expected to be separated into two clusters, instated of gathering in one cluster.

Principle (I) is for overcoming the computing-complexity of model evaluations in SBSE problems. Principles (II) and (III) assume: there are some mappings between configuration and decision space (described in §3.2) in SBSE problems– candidates with similar decisions should have similar objectives. This assumption lets us avoid evaluating every candidate. Principle (IV) is to guarantee the diversity of SWAY outputs. For problems with multi-objectives, diversity of results is a main consideration.

To help researchers understand these principles intuitively, here we have a guide for creating SPLIT functions:

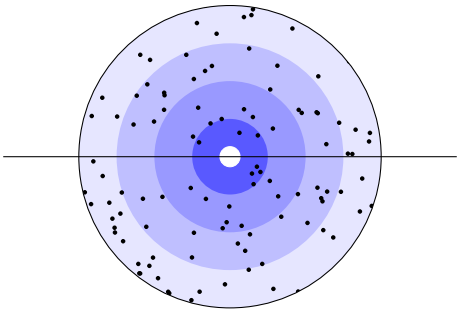


Figure 6.1 Map candidates into a circle by domain knowledge. The large white dot in the center is the "pivot", selected randomly among the candidates. All other candidates (the black dots) are located based on their radius and angular coordinates. The circle is divided into multiple equal-thickness annulus. The candidates with minimum angular coordinates form the *east* representatives. The candidates with maximum angular coordinates form the *west* representatives. Candidates whose angular coordinate is less than π(upper semicircle) form the *eastItems* and others (locates in lower semicircle) form the *westItems*.

1. To support principle II and III, for problems with continuous decisions or discrete decisions (with interval attribute [Ste46]), Minkowski distance, especially the Euclidean distance, is a good choice for measuring the similarity between candidates.

2. If clustering all candidates into two groups fails, then it is very likely that it did not fulfill principle (IV). To overcome this, we can exploit domain knowledge for the problem, manually divide candidates into several groups first. After that, apply the SPLIT function to each group.

3. Main goal of SPLIT function is to divide candidates into two clusters. To determine which cluster is good one, we must select representative(s). Two possible strategies for this selection – first is selecting random points; another is selecting extreme points, which can form the diameter of decision space.

6.3 Case Study I: Software Product Line optimization

6.3.1 An implementation of SWAY^2, Splitting the Discrete Space

Algorithm 2 performed well on models with numerical decisions. However, when applied to the problem with binary decisions, i.e. $D = \{0,1\}^n$, it was observed that SWAY performed far worse than standard MOEAs. On investigation, we found that reducing the decision space into a single line losses some information especially for the binary decisions. Specially, if the candidates are spaced using only the total number of 1-bits, then the distribution of instances was highly skewed towards the lower end.

Accordingly, inspired by the research in radial basis function kernel [Chu03], we invented a radial co-ordinate system as an implementation of SWAY^2. In a nutshell, this co-ordinate system divided the configuration space into several parts, basing on the "scale"(number of 1 bits in on configuration); and then applied the SWAY on each part. Specifically, the radial co-ordinate system can force

Algorithm 3: SPLIT for (binary) configuration spaces with domain knowledge

Input : items – The candidates to split
Output : Decision space is divided into multiple sub-space. Each subspace has eastItems, westItems and two corresponding representatives.
Parameter : totalGroup – the granularity
Require Func : DISTANCE

```
1  rand ← randomly selected item in candidates
2  foreach x ∈ items do
3      x.r ← |∀d_i ∈ x ∧ x == 1| // sum all the ''1'' values
4      x.d ← DISTANCE(x, rand)
5  normalize x.r into [0, 1]
6  R ← {i.r | i ∈ items} // all possible radius
7  foreach k ∈ R do
       // for each possible radius
       // equally distribute the candidates with k-radius into the concentric-circle
8      g ← {i | i.r = k}
9      sort g by x.d. g ← x_1, x_2, ..., x_|g|
10     for i ∈ [1, |g|] do
11         x_i.θ ← 2πi / |g|
   // split the candidates
12 thk ← max (R)/(totalGroup) // the thickness
13 foreach a ≤ totalGroup do
       // for the annulus with (a−1)thk≤ radius ≤ a thk
14     g ← {i | (a−1)thk≤ i.r ≤ a thk}
15     c_1 ← the item with minimum θ in g
16     add c_1 to east
17     c_2 ← the item with maximum θ in g
18     add c_2 to west
19     add items with θ ≤ π in g to eastItems
20     add items with θ > π in g to westItems
21     Create a sub-space infomation ← [c_1, c_2, eastItems, westItems]
22 return All sub-space informations created in Line 21
```

vectors of binary decisions away from outer edges into the inner volume of that space. Candidates representing similar-size individuals (share similar # of 1 bits) are grouped and comparisons only be performed inside the group. Algorithm 3 on page 41 implements such a radial co-ordinate system. This algorithms splits our binary decision using a randomly selected "pivot" point. After that, it maps the other candidates into a *circle*, rather than the *line* showed in Algorithm 2.

To map the candidates into this circle, first, for the candidate $x = (d_0, d_1, \ldots, d_n)$ ($d_i \in \{0, 1\}$), we assign $x.r$ as $\sum_{i=0}^{n} d_i$ and $x.d$ as the *Jaccard distance* between x and the "pivot" candidate(lines 2-4). The Jaccard distance is defined as

$$\sum |a_i - b_i| (A = (a_0, \ldots, a_n), B = (b_0, \ldots, b_n), a_i, b_i \in \{0, 1\})$$

Once mapped into a circle, we then uniformly spread all candidates with similar r values into a circumference whose radius is r, based on their d values– the one with minimum d values has the minimum angular coordinate; the one with second minimum d values has larger coordinate; and so on (lines 7-11).

This circle is then used to generate the partitions. Figure 6.1 shows how this circle is divided into several equal-thickness annulus (the number of annulus, i.e., the granularity of SPLIT is a configurable parameter). After the division:

- The candidates with minimum θ in each annulus area form the *east*;
- The candidates with maximum θ form the *west*.
- Candidates in the upper semicircle form the *eastItems* and others form the *westItems*.

6.3.2 Software Product Line Optimizations

As a study case, this research use software product line optimization to explore algorithm proposed in §6.3.1. A software product line (SPL) is a collection of related software products, which share some core functionality [Har14a]. From one product line, many products can be generated. For example, Apel et al. [Sie12] model the compilation configuration parameters of databases as a product line. By adjusting those configurations, a suite of different database solutions can be generated.

Figure 6.2 shows a feature model for a mobile phone product line. All features are organized as a tree. The relationship between two features might be "manadory", "optional", "althernative", or "or". Also, there exists some cross-tree constraints, which means the referred features are not in the same sub-tree. These cross-tree constraints complicate the process of exploring feature models[1]. In practice, all constraints, including the tree-structure constraints and the cross-tree constraints can be expressed by the CNF (conjunctive normal form). For example, Figure 6.2 indicates the following 19 CNFs.

[1]Without cross-tree constraints, one can explore the feature model through the top-down breath-first search.

$$
\begin{cases}
\neg\text{Mobile Phone} \vee \text{Calls} \\
\text{Mobile Phone} \vee \neg\text{Calls} \\
\neg\text{Mobile Phone} \vee \text{Screen} \\
\text{Mobile Phone} \vee \neg\text{Screen} \\
\text{Mobile Phone} \vee \neg\text{GPS} \\
\text{Mobile Phone} \vee \neg\text{Media} \\
\text{Media} \vee \neg\text{Camera} \\
\text{Media} \vee \neg\text{MP3} \\
\neg\text{Media} \vee \text{Camera} \vee \text{MP3} \\
\text{Screen} \vee \neg\text{Basic} \\
\text{Screen} \vee \neg\text{Color} \\
\text{Screen} \vee \neg\text{High resolution} \\
\neg\text{Screen} \vee \text{Basic} \vee \text{Color} \vee \text{High resolution} \\
\neg\text{Basic} \vee \neg\text{Color} \vee \neg\text{High resolution} \\
\text{Basic} \vee \neg\text{Color} \vee \neg\text{High resolution} \\
\neg\text{Basic} \vee \text{Color} \vee \neg\text{High resolution} \\
\neg\text{Basic} \vee \neg\text{Color} \vee \text{High resolution} \\
\neg\text{GPS} \vee \neg\text{Basic} \\
\neg\text{Camera} \vee \text{High resolution}
\end{cases}
$$

Harman et al. [Say13c; Say13a; Har14b; Hen15] defined following five practical objectives.

1. Find the valid products (products not violating any cross-tree constraint or tree structure) which have..
2. More features; and
3. Less known defeats; and
4. Less total cost; and
5. Most features used in prior applications.

In our experiment, we seek to optimize the above five goals too. Following the recommendation of Sayyad [Say13a], defect, cost, and knowledge of usage in prior applications is set stochastically.

A major problem with analyzing software product lines is that it can be very hard to find valid product since real-world software product lines can be far more complex than our example. Some software product line models comprise up to tens of thousands of features, with 100,000s of constraints (see Table 6.1). These networks of constraints can get so complex that random assignments of "use" or "do not use" to the features have very low probability of satisfying the constraints. For

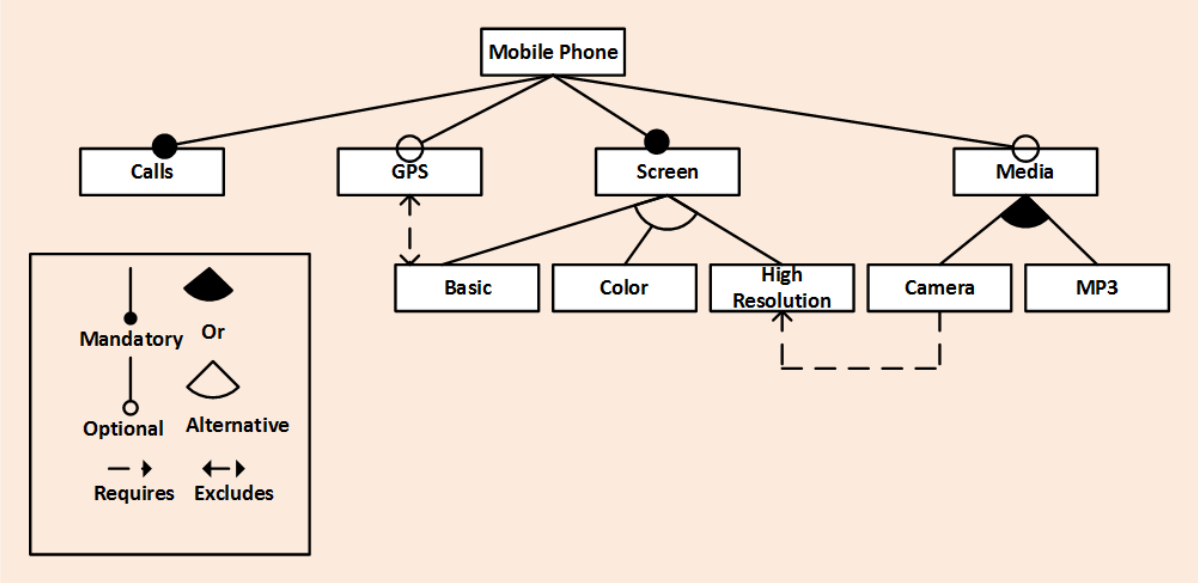


Figure 6.2 Feature model for mobile phone product line. To form a mobile phone, "Calls" and "Screen" are the mandatory features(shown as *solid* •), while the "GPS" and "Media" features are optimal(shown as *hollow* ◦). The "Screen" feature can be "Basic", "Color" or "High resolution" (the *alternative* relationship). The "Media" feature contains "camera", "MP3", or both (the *Or* relationship).

example, in one of our software product lines, the *linux* model, we generated 10,000 random sets of decisions for the features. Within that space, less than 4 decisions were valid.

Consequently, much of the research on optimizing the generation of products from software product lines has focused on how best to optimize within this heavily-constrained models:

- Sayyad et al. [Say13a] introduced the *IBEASEED* method– a five-goal optimization problem had its first generation of candidates initialized by a pre-processor that just sought out valid products (and one other goal).

- *SATIBEA* was introduced by Henard et al. [Hen15]. This makes full use of SAT solver technology to fix the invalid candidates every time the "mutate" or "crossover" operation is performed in the IBEA algorithm. Results showed that the SATIBEA algorithm can find the valid products for the extremely large feature models by tens of thousands evaluations, much better than other algorithms.

To the best of our knowledge, the SATIBEA is the best algorithms to find the optimal SPL which are represented in the form of CNFs. Consequently, we will compare SWAY with SATIBEA algorithm.

We used five product line models from SPLOT and LAVT repositories. Table 6.1 shows the basic information of our five cases.

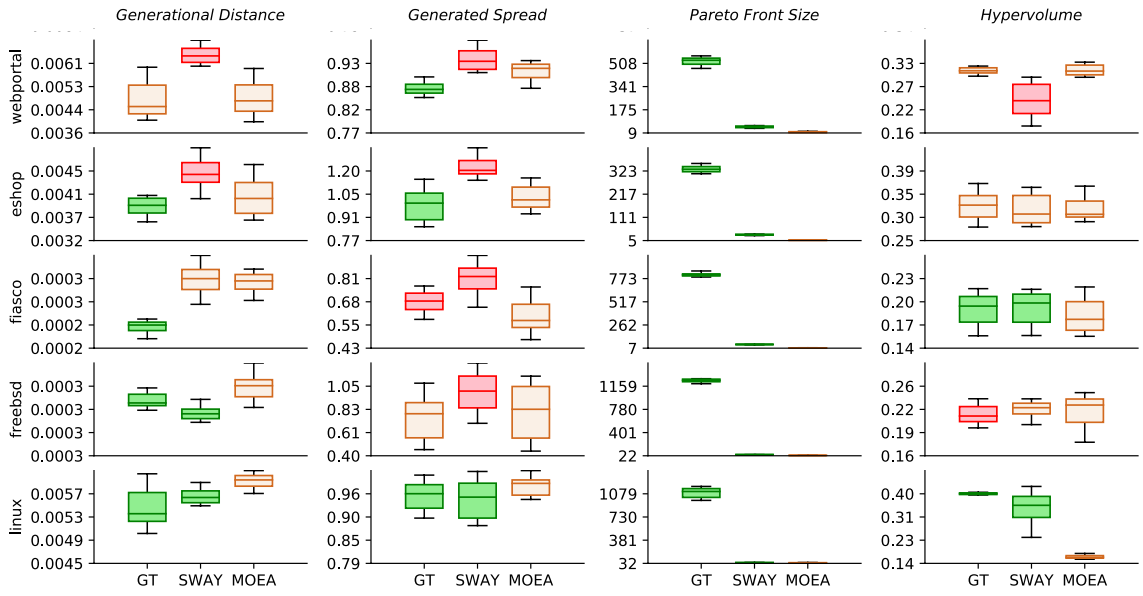


Figure 6.3 The box-plot of four quality indicators in each study cases (30 runs). In each box-plot: **Left:** the GROUNDTRUE method - The frontier of 10k random generated populations ; **Middle:** the SWAYalgorithm; **Right:** the state-of-the-art MOEA for that corresponding study case. Wilcoxon test was performed at a 5% significance level: The **orange box** marks the state-of-the-art method, or method which is no statistical significantly different from state-of-the-art method; **green box** marks the method which is significant better than the state-of-the-art while **red box** indicates that method is worse than the state-of-the-art method. Generational distances is the coverage indicator. Generated spread and pareto front size are the diversity indicators. Hypervolume is the combination of convergence and diversity. Lower values are preferred for generational distance, generated spread while higher values for pareto front size and hypervolume (see text for more details).

Table 6.1 Feature models used in this study, sorted by the number of constraints. The constraints includes the tree-structure and cross-tree constraints. SPLOT models can be found at `http://www.splot-research.org/` and LVAT models are at https://code.google.com/archive/p/ linux- variability-analysis- tools/

Name(Source)	Number of features	Number of constraints	Reference
webportal (SPLOT)	49	81	[Men08]
eshop (SPLOT)	330	506	[Lau06]
fiasco (LVAT)	1638	5,228	[Ber12]
freebsd (LVAT)	1396	62,138	[She11]
linux (LVAT)	6888	343,944	[She11]

Table 6.2 Median value of runtime and model evaluation numbers from 30 independent runs. SATIBEA is the state-of-the-art multi-objectives optimization strategy.

	Runtime(seconds)			**Evalutation#**		
	SWAY^2	SATIBEA	$\frac{\text{SWAY}^2}{\text{SATIBEA}}$	SWAY^2	SATIBEA	$\frac{\text{SWAY}^2}{\text{SATIBEA}}$
webportal,81	6	244	2%	134	5100	3%
eshop,506	18	321	6%	136	6732	2%
fiasco,5228	63	1065	6%	122	20102	1%
freebsd,62138	188	2058	9%	136	26146	1%
linux,343944	1103	3295	33%	168	8900	2%

6.3.3 Result I: Comparing the Efficiency

Table 6.2 is the median runtime and evaluation numbers recorded in our experiments. It is evident that, in all cases, SWAY^2 is faster than the state-of-the-art MOEA. Note that the SWAY^2 only requires less than 3% of model evaluations of the other MOEAs. Which means, SWAY^2 is useful when the complex SE models which model evaluation is computational expensive. With respect to the runtime, we can get the similar conclusion. For example, in the study case *linux*, the median runtime of SWAY^2 was 1103s (18min), while the runtime of SATIBEA algorithm was nearly an hour.

Consequently, from Table 5.3, we can conclude as: SWAY^2 terminates much faster of other evaluation algorithms, from several times to two orders of magnitude faster.

6.3.4 Result II: Comparing the Effectiveness

Figure 6.3 shows the distributions of all quality indicators among the 30 independent runs in our experiment. We plot three methods for each study case. The GROUNDTRUE method generates and evaluates 10,000 solutions, and then fetches the non-dominated solutions using the NSGA-II sorting strategy. The SWAY^2, on the other hand, randomly generates 10,000 solutions, and then fetches some of them through the sampling strategy, which requires much fewer evaluations. The third method is the state-of-the-art evolutionary algorithm, i.e. SATIBEA method.

The green boxes in SWAY columns indicate that SWAY^2 was significantly better than the state-of-

the-art MOEA (compared by the Wilcoxon test at a 5% significance level). It was found that in most cases, the SWAY^2 performs as well as, or better than the state-of-the-art algorithm. In only some cases with GD/GS as indicators did the MOEA methods outperform SWAY^2 (these are indicated by red boxes).

Another interesting result is that at most of the time, the GROUNDTRUE method performs better, or at least no significant worse than that of MOEA methods. That means, if someone found a better and faster sampling algorithm, it might be able to beat the evolutionary strategy. "Sampling" is enough for solving such kind of software engineering problems, as long as the sampling strategy is "smart" enough.

From the above, we can draw a conclusion: the SWAY^2 can get comparable results as the evolutionary algorithms.

6.4 Case Study II: Next Release Problem

6.4.1 Split Function for Discrete Models (NRP)

NRP problem *(discussed very soon)* has a discrete decision space. In our previous attempts, simply applying algorithm in Algorithm 2 did not work for this problem. From guide 2, we can exploit domain knowledge to divide candidates into several groups first and then use algorithm in Algorithm 2 in each group.

Let us explore how to exploit domain knowledge of the problem. In NRP, if more features are to release in early versions, then developing teams' workload tends to be intense in early stage (otherwise, the workload can shift to late stage, or spread equally). Exploiting this domain knowledge, we can divide candidates into groups according to their workload mode:

$$\mathbf{WL}(\mathbf{y}) = ||\{1 \le i \le N : y_i < \mathrm{P}/2\}|| \tag{6.1}$$

where $\mathbf{y} \in [1, P]^N$ represents a release plan for a project with N features and have P releases at maximum. Consequently, $\mathbf{WL}(\mathbf{y})$ represents how many features will be released in first half of release plan. We can divide candidates into groups. Note that, once an initial split of the space has completed using this principle, we can further split the resulting space using FASTMAP because after manual division, each group contains a subspace of the decision space (principle (IV)) which can be more easily handled by *split* functions described in Algorithm 2.

This is just one guidance for discrete models. For other problems, researchers can adjust the pre-handler according to problem description. In our experience, these pre-processors are simple to code, just like $\mathbf{y}$.

6.4.2 The NRP problem

In requirement engineering, next release problem (NRP) is one of problems with high complexity. NRP concerns with defining which requirements should be implemented for the next version of

Name	# Requirements	# Releases	# Clients	# Density	# Budget	Level of Constraints
NRP-50-4-5-0-110	50	4	5	0	110	1
NRP-50-4-5-0-90	50	4	5	0	90	2
NRP-50-4-5-4-110	50	4	5	4	110	3
NRP-50-4-5-4-90	50	4	5	4	90	4

Figure 6.4 Variants of NRP used in this study.

the systems, according to customer satisfaction, budget constraints as well as precedence constraints between various requirements. Durillo et al. [Dur09], treated the next release problem as a multi-objective problem, since higher customer satisfaction and less development time or cost are conflicting objectives, we call this formulation as multi-objective NRP (NRP). NRP in this paper considers (maximizing) combination of importance and risk, (minimizing) cost and (maximizing) satisfaction.

The problem can be mathematically described as follows. Given a software project with N requirements, find a vector $\vec{y} \in \{0,\ldots,P\}^N$ so that following objectives can be optimized.

$$\begin{cases} f_1 = \sum_{j=1}^{M} t_j \left(\sum_{i=1}^{N} (P+1-y_i) I_{ij} + r_i \right) \\ f_2 = \sum_{i=\{[1,N]|y_i>0\}} c_i \\ f_3 = \sum_{i=\{[1,N]|y_i>0\}} \sum_{1}^{M} I_{ij} \end{cases} \tag{6.2}$$

subject to

$$\sum_{i=1}^{N} c_i y_i \leq \mathrm{BR}_k, \forall k \in [1,P]$$

$$y_i \leq y_j, \forall e_{ij} \in E(G)$$

Where, P is total number of releases; $y_i = 0$ indicates abortion of requirement i; M is number of customers and t_j is the importance of t-th customer for developing company; I is a matrix which element I_{ij} indicators the business value of requirement i in view of customer j; r_i is the risk of requirement i;c_i is the economic cost of achieving requirement i; BR is an vector showing the budget of each release; G is a DAG indicating the release topology of different requirements.

First objectives f_1 indicates a combination of customer values as well as risk. The objective is to fulfill customers' requirements, as well as requirements with high importance first. While second objective f_2 sums up total economic cost for all developed requirements and f_3 sums up customers satisfaction at the end of all releases.

The first constraint indicating that the total cost should not exceed the budget allocated to each release. The second constraint is for topology of requirements.

This research explore 4 variants of NRP, ranging from the least constrained to the most con-

strained. Figure 6.4 lists the variants of the problem used in the paper. For example, problem variant NRP-50-4-5-4-110, describes the scenario where a software project has 50 requirements; among all requirements, 4% are dependent on others; also, the software is to develop for 5 clients within 110% of budgets. This means that the project is over-funded and hence making it less constraint wrt. its budget. The column *Level of Constraints* represents the level of constraints, 1 being the least constrained and 4 being the most constrained.

6.4.3 Case Study Research Questions

This work formulate our research questions in terms of the applicability of the techniques used in SWAY. As our approach promotes sampling instead of evolutionary techniques, it is a natural question, "how is this possible?". "Sampling instead of evolution" is very counter-intuitive to practitioners since, EVOL have been widely accepted and adopted by the SBSE community.

Also, if we are only sampling from 100/10000 solutions, it is possible to miss solutions, which are not present in the initial population. These are valid arguments while trying to find near-optimal solutions to problems with competing objectives. It may be argued that sampling is such a straight forward approach so it is critical to evaluate the effectiveness of such sampling method on variety of SE models both constrained and unconstrained, discrete as well as continuous. It is also interesting to see how the techniques of SWAY can be borrowed by the traditional MOEAs to improve the performance.

Therefore, to assess feasibility of our algorithm, we must consider:

- Performance scores generated from SWAY when compared to other MOEAs
- Can SWAY be used to super-charge other MOEAs such that the performance of super-charged MOEA is better than standard MOEA?

The above considerations lead to three research questions:
RQ1: Can SWAY perform "as good as" traditional EVOL *in NRP models?*
RQ2: Can SWAY be used to boost or super-charge the performance of other EVOL*?*
Since, SWAY uses very few evaluations, it can be potentially used as a preprocessing step for other MOEAs for cases where function evaluation is not expensive.

Please note that this work did not compare the efficiency of SWAY[2]. The efficiency of SWAY has been proved in many study cases discussed in former chapters. Theoretically, since the SWAY requests much less model evaluations then EVOL, it performs orders of magnitude faster.

6.4.4 Experimental Setup

In the following, we compare EVOL to SWAY[2] for 20 repeats[2]. All the optimizers use the population size recommended by their original authors; i.e. $n = 100$. But, to test the effects of increased sample, we run two versions of SWAY:

[2]Note: 30 repeats would be better. Limited by the computing resources during the experiment hours.

- SWAY100: builds an initial population of size $10^2 = 100$.
- SWAY10k: builds an initial population of size $10^4 = 10,000$.

One design choice in this experiment was the evaluation budget for each optimizer:

- If we increase number of iterations in EVOL to a very large number, that would bias the comparison towards EVOL since better optimizations might be found just by blind luck (albeit at infinite cost).
- Conversely, if we restrict EVOL to the number of evaluations made by (say) SWAY10k then that would unfairly bias the comparison towards SWAY since that would allow only a generation or two of EVOL.

Consequently, to compare SWAY^2 and EVOL, following the practice of [Kra15a; Kra16], we set maximum number of evaluations to 2000.

6.4.5 Results

As seen in Figure 6.5, EVOL performs worse that SWAY10k in term of spread. When comparing results from Hypervolume, EVOL performs better than SWAY10k in only one case (NRP-50-4-5-4-100). Please note that the NRP-50-4-5-4-110 is the least constrained problem (among the constrained problems) considered in this paper. Hence, we cannot recommend this EVOL as our preferred method, for the NRP family of problems. We observe in Figure 6.5 that:

- SWAY10k has the lowest spreads (lower the better) and highest hypervolume (higher the better) in most cases.
- The exception being NRP-50-4-5-4-100, which is the least constrained along the four variants. That said we see that SWAY is the top ranked optimizer in $\frac{4}{4}$ cases and $\frac{3}{4}$ with respect to Spread and Hypervolume respectively.
- Figure 6.6 compares the number of evaluations required by SWAY and other optimizers. Note that SWAY requires 20 times less evaluations compared to the standard optimizers, while performing reasonably well in other quality metrics.

In summary, the NRP results nearly always endorse the use of SWAY10k.

To answer the second research question, we compare EVOL which were super-charged with the results from SWAY^2. Super-charging a EVOL means seeding the initial population with the results obtained from SWAY^2. The supercharged NSGA-II and SPEA2 is called NSGA-IISC and SPEA2SC respectively. NSGA-IISC and SPEA2SC shared exactly the same configurations as in RQ1 except for the initial seeds.

The results from super-charging are shown in Figure 6.7. Results that endorse super-charging would have the following form:

Rank	Treatment	Median	IQR
50-4-5-0-110			
1	SWAY10k	105	14
2	SPEA2	112	23
3	NSGAII	130	19
50-4-5-0-090			
1	SWAY10k	98	8
2	SPEA2	137	45
3	NSGAII	167	34
50-4-5-4-110			
1	SWAY10k	97	15
2	SPEA2	113	11
3	NSGAII	125	28
50-4-5-4-090			
1	SWAY10k	102	12
2	SPEA2	119	21
2	NSGAII	133	21

Rank	Treatment	Median	IQR
50-4-5-0-110			
1	SPEA2	157	16
1	NSGAII	154	13
2	SWAY10k	111	6
50-4-5-0-090			
1	SWAY10k	103	8
1	SPEA2	105	20
1	NSGAII	104	19
50-4-5-4-110			
1	SWAY10k	145	5
2	SPEA2	68	3
2	NSGAII	66	5
50-4-5-4-090			
1	SWAY10k	73	3
2	SPEA2	63	4
2	NSGAII	64	5

Figure 6.5 Spread and hypervolumes seen in 20 repeats for NRP models. Upper: Spread (*less* is *better*). Bottom: Hypervolume (*more* is *better*).

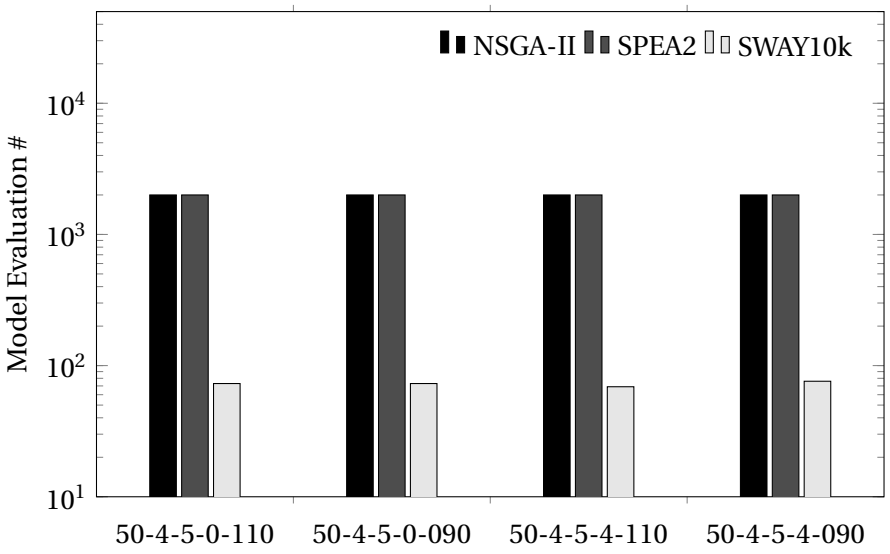


Figure 6.6 Medium number of model evaluations for NRP models.

- The super-charged version of the algorithm would have lower spreads or higher hypervolumes that otherwise;
- The results from the super-charged version have a different statistical rank (as listed in the first column of these results).

Overall, the results endorsing super-charging occur so rarely that we cannot recommend SWAY as a pre-processor to other optimizers. Even though this experiment showed that SWAY is not a good idea for super-charging other MOEAs, it does suggest another, potentially promising, avenue for future SBSE research. A common view for evolutionary algorithms is that they find out solutions through evolution, or improvements between iterations. Consequently, many researchers are focusing on how to customize operators between iterations so that EVOL's performance can be enhanced. However, this paper, we show that 1) SWAY, a sampling technique, can get similar results as standard EVOL; 2) standard EVOL cannot improve significantly from SWAY's output. These means evolution is not the only method that might be able to solve complex non-linear problems. Future researchers might also choose to focus on sampling methods.

6.5 Comments on the Simplicity of SWAY

Note: The SWAY in title of current section means the "sampling way" method, including the OSAP1 *and its variant* OSAP2.

One interesting feature of this work is our focus of simple, methods that are not so CPU-intensive as standard methods. In this regard, this work is somewhat unusual since our reading of the current literature is that most researchers in the optimization area are mostly trying to exploit more CPU.

Rank	Treatment	Median	IQR
50-4-5-0-110			
1	SPEA2	127	0.0
2	NSGAII	130	0.0
3	SPEA2SC	137	0.0
4	NSGAIISC	148	0.0
50-4-5-0-090			
1	SPEA2	140	23.18
2	SPEA2SC	158	30.77
3	NSGAII	167	30.74
3	NSGAIISC	183	37.95
50-4-5-4-110			
1	SPEA2	102	11.8
2	SPEA2SC	114	22.04
3	NSGAII	125	24.87
3	NSGAIISC	145	45.14
50-4-5-4-090			
1	SPEA2	117	29.07
1	SPEA2SC	122	20.85
2	NSGAII	133	32.41
2	NSGAIISC	149	39.47

Rank	Treatment	Median	IQR
50-4-5-0-110			
1	NSGAII	154	17.94
1	SPEA2	162	17.18
2	SPEA2SC	141	19.17
2	NSGAIISC	139	12.51
50-4-5-0-090			
1	SPEA2	103	22.4
1	NSGAII	104	15.67
1	SPEA2SC	102	15.23
1	NSGAIISC	101	21.28
50-4-5-4-110			
1	SPEA2	71	6.28
2	NSGAII	66	10.12
2	SPEA2SC	64	5.9
3	NSGAIISC	56	15.55
50-4-5-4-090			
1	NSGAII	64	14.78
1	SPEA2	66	12.72
1	SPEA2SC	70	9.87
1	NSGAIISC	64	19.3

Figure 6.7 SWAY results were used to seed the initial population of the traditional EVOL. Spread, hypervolumes and evaluation seen in 20 repeats. Upper: Spread (*less* is *better*). Bottom: Hypervolume (*more* is *better*).

Accordingly, this section comments on why we think that "less is more" and should be studied by researchers.

Wolpert's *no free lunch theorems* [WM97a] for optimizers and learners emphasized that **No learner/optimizer is always best**, which means we always need N methods on hand when we arrive at new data. While there always might be a better learner for this particular data set, the greater the performance gain desired, the fewer the learners exist that produce at least such a performance gain [Mon13]. That is, if we (say) twice find improvements over some initial baseline, the the odds of finding something even better become very small. So it does not doom us to an infinite search – just a search for something that is better than an initial result.

6.6 Summary of OSAP2

The OSAP2 is an advanced version of SWAY – it adds an initial division before the SWAY. Such first step requires the expert or domain knowledge. In this chapter, we had some policy suggestions for the first step. However, the OSAP2 is still built on the "golden" region assumption. That is a limitation.

CHAPTER

7

RIOT, THIRD GENERATION OF OSAP

This chapter is based on the publication

- [CM18] "RIOT: A Stochastic-Based Method for Workflow Scheduling in the Cloud." IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018.

The OSAP1 or OSAP2 discussed in previous chapters are based on one assumption: *The optimal configuration (i.e. pareto frontier) only exist in one (or several) small region(s) of decision spaces.* This assumption has been approved by widely-explored SE problems, such as the product line optimization problem discussed above. However, latter we found that some SE problems, such as the workflow configuration problem introduced very soon, defect this assumption. That is, the pareto frontier of the problem might *spread among many regions of the configuration space.* As a result, the path seeking for "golden" regions in decision space may be unpromising. A new configuration selector/comparator which can quickly determine whether any configuration is valuable should be introduced. The third generation of OSAP addresses this.

7.1 Linear Surrogate Model as the Selector

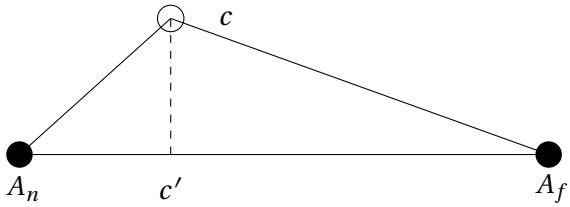
To avoid mistakenly removing valuable configurations, OSAP3 does not cluster or prune and group of initial candidates. Instead, it creates additional configurations, evaluate them and use that as **anchors** to guess the objectives of other candidates. These anchors should spread among the configuration space. There are three strategies to set the anchors:

- set the diagonal of the decision space as anchors, OR

Algorithm 4: Framework of OSAP3 – Linear Surrogate Model as the Selector

Input : items – The candidates
Output : pareto frontiers

1 $Anchors \leftarrow n$ evaluated items
2 $Randoms \leftarrow N \gg n$ un-evaluated items
3 **foreach** $c \in Randoms$ **do**
4 $A_n \leftarrow$ configurations in $Anchors$ that nearest to c
5 $A_f \leftarrow$ configurations in $Anchors$ that furthest to c
6 **foreach** $o \in \{o_1, o_2, \ldots\}$ **do**
7 $o_n, o_f \leftarrow o(A_n), o(A_f)$
8 $d_0, d_1 \leftarrow dist(A_n, c), dist(A_n, A_f)$
9 $\theta \leftarrow \arccos(||\overrightarrow{A_n c}||/||\overrightarrow{A_n A_f}||)$
10 $\widehat{o}_C \leftarrow \frac{d_0 cos\theta}{d_1}\left(o_f - o_n\right) + o_n$
11 Collect all items and return all frontiers



$$\frac{C_{A_n c'}}{C_{A_n A_f}} = \frac{O_{A_n c}}{O_{A_n A_f}}$$

Figure 7.1 Linear Surrogate model for quickly guessing the objectives. A_n, A_f are evaluated anchors. c is a candidate to be assessed. LHS in the equation calculates the ratio in configuration space, while RHS reflects the corresponding relations in objective space. This equation is consistent with line 7-10 in Algorithm 4.

- randomly create a few anchors, OR
- integrate two strategies.

With anchors set and evaluated, surrogate methods can be helpful to guess objectives of other candidates. Surrogate methods substitutes the origin complex model with a very simple surrogate model. Utilizing surrogate models, we can change the evaluation of complex SE model into very simple (but effective) mathematical function. When we are able to quickly assess the configurations, we can explore many more candidates among the decision space, therefore, find out the pareto frontiers of the problem which might spread among the whole decision space.

The proposed surrogate model in this research is based on following two conjectures in the SBSE problems:

- 1) configurations with similar decisions should share similar objectives;
- 2) within some interval in the decision space, the objective is monotonically increasing or decreasing. In other words, **they are linear models.**

With these conjectures, we proposed Algorithm 4. We randomly generate large amount of candidates, and then evaluate a small partial of them (the *Anchors*). For the rest of candidates, we create a surrogate model to estimate their objectives (as is shown in Figure 7.1). Next, we will apply this strategy to a SBSE problem–cloud computing configuration.

7.2 Case Study: Workflow Deployment Optimizations

7.2.1 Motivation

Scientific workflows (a.k.a data-intensive workflow) such as those shown in Figure 7.2 [DC08] have been widely applied in scientific research, data mining and business intelligence analysis [Vöc11]. One fundamental problem in workflow research is *workflow scheduling*, i.e. associating the appropriate computer resource to each task in the workflow.

For complex workflows, prior work used meta-heuristic optimizers (genetic algorithm, particle swarm optimization, ant colony optimization, etc. [Top02; RB17; CZ09; Dor06; RB14; Shi01; Tsa14; Zhu16]). Such meta-heuristics are often computationally expensive. Later in this case study, we applied a current state-of-the-art meta-heuristic workflow scheduling algorithm (EMSC [Zhu16]) to find optimized scheduling for 20 workflows. The the total runtime for optimization was more than 10 hours (on desktop computer with 2.0GHz, 8GB memory). As a comparison, the total expected runtime for these workflows was around 9 hours. In some specific workflows, the optimized time is 14x longer than its expected runtime in Amazon AWS clouds.

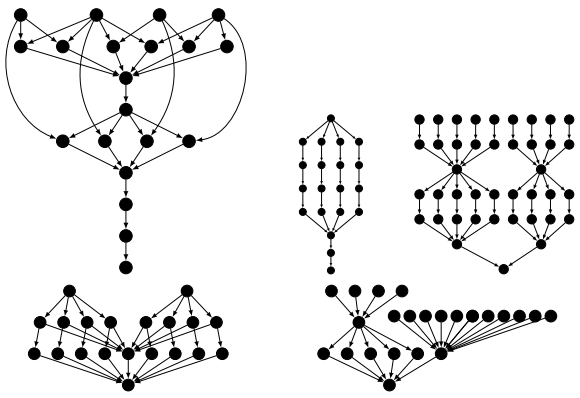


Figure 7.2 Examples of cloud computing workflows. Clockwise from top left: `Montage, Epigenomics, Inspiral, CyberShake, Sipht`. Each node is one "task" and each edge is a data flow from one task to another. Number of tasks can vary from dozens to thousands). The scheduling problem is to map these tasks to a smaller number of virtual machines, decide the ordering of tasks within one VM, and then decide what kind of machine should drive each VM.

Such long runtimes are problematic. Cloud computers execute in highly dynamic environments where scheduling tools need to be adaptive to changing conditions [RS10; Sch10; Ios11]. For example, Schad *et al.* [Sch10] found that the runtime of a widely used benchmarks suite can vary by up to 33% even when run on supposedly identical instances within the same cloud environment. Not only CPU, but also bandwidth can be highly variable within the cloud. Schad *et al.* report that network bandwidth between the same type of EC2 instances can vary from 410KB/s to 890KB/s. Hence, even after a workflow is planned and deployed, it is important to monitor instances and repeat the scheduling process during deployment when necessary. If repeated reschedulings are too slow, then it becomes impractical to use those algorithms.

7.2.2 Problem Formulation

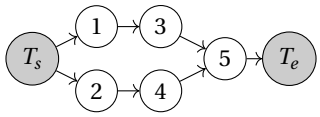
Scientific workflows, a.k.a. data-intensive workflows typically contain many computational tasks. These tasks are commonly interconnected via data or resource dependencies. They enable researchers to collaboratively design, manage, and obtain results that involve hundreds of steps, access big data and generate similar amount of intermediate and final data products [DC08]. One common way to represent the dependencies is through Directed Acyclic Graph (DAG) as shown in Figure 7.2. For a DAG $D = \langle V, E \rangle$, each task is represented as a vertex and every edge $e(i, j)$ indicates that task j must be executed after task i is finished. Mathematically, we denote

$$Pred(i) = \{j | (j, i) \in E\}$$
$$Succ(i) = \{j | (i, j) \in E\}$$

Task i can start only after all tasks of $Pred(i)$ are terminated. For convenience, among all tasks, we denote T_s as the *start* task which has no predecessors; and T_e, on the other hand, as the *exit* task without any successors. In this work, we assume that all workflows have single start task and exit task (this can be simply assured by adding dumb vertex as the head(tail) of all start(exit) tasks).

When deploying workflow into cloud environment, such as Amazon AWS services or Microsoft Azure Cloud, tasks can be executed in different virtual machines. Input/output files of tasks can be transferred via networking. Parallel executing in multiple virtual machines significantly reduced execution time of whole workflow, compared to trivial single PC execution. A deployment plan for the workflow under the specific cloud environment can be uniquely determined by following three components (see Figure 7.3 as an example) [Zhu16]:

- **task2VM mapping**: number of VMs should be used and what tasks should be deployed in the same VM.
- **VM types**: which type of computational resources should be assigned to the VMs.
- **secondary ordering**: ordering tasks within the same VM. Tasks inside one VM are polling in secondary ordering until one of them is ready to run.



task2VM mapping
$T_s, 1, 2 :: VM_a$
$3, 4 :: VM_b$
$5, T_e :: VM_c$

VM types
VM_a, VM_c :: t2.small
VM_b :: t2.large

secondary ordering
$T_s, 1, 2, 3, 4, 5, T_e$

Figure 7.3 A scheduling example that uniquely set the scheduling. In this workflow, Task $T_s, 1, 2$ are deployed in VM_a (an AWS EC2 t2.small instance). Task 3, 4 are deployed in VM_b (an AWS EC2 t2.large instance), etc. If both task 3 and 4 are ready to run at one moment, VM_a will run task 3 first, since task 3 has higher rank.

In this work, we treat workflow as a multi-objective problem. The goal of RIOT is to minimize execution time as well as the cost of hiring the virtual machines from cloud service provider.

For each task i, denote

$$ft(i) = st(i) + dur(i) \tag{7.1}$$

$$dur(i) = workloads(i) + filetime(i) \tag{7.2}$$

$$filetime(i) = \sum_{j \in Succ(i)}^{VM(i) \neq VM(j)} \frac{file(i,j)}{min[bw(i), bw(j)]} \tag{7.3}$$

where $ft(i), st(i), dur(i), workloads(i), filetime(i)$ are finish time, start time, duration, computational workload and I/O time of task i respectively. Duration of task i includes computing time of the workload as well as I/O time. $file(i,j)$ is the data flow between task i and task j. I/O speed is limited by bandwidth(bw) of VMs. Then, "execution time" is measured as *makespan*; i.e. $ft(T_e)$.

As to cost, in scientific workflows, we create a virtual machine when any task need it and terminate it only when no more future task needs it. Hence, the cost of a workflow is the sum of cost of every used virtual machine. Cost rate and charging policy may differ among providers so for this work, we followed AWS EC2 pricing (as done by previous works [Zhu16; DP14; RB14]). Note that the charging unit of VM by the hour.

A multi-objective optimizer should return is the *Pareto Frontier*. Mathematically, in this problem, we define one scheduling s_i dominates another scheduling s_j iff

$$makespan(s_i) \leq makespan(s_j) \text{and } cost(s_i) \leq cost(s_j) \tag{7.4}$$

All schedulings not dominated by any others form the Pareto Frontier. Engineers can inspect this frontier to find the solutions they find most useful.

7.2.3 Related Work

One of the earliest results in this area came from Topcuoglu *et al.* [Top02], who proposed HEFT (Heterogeneous-Earliest-Finish-Time). HEFT has two phases, task prioritizing phase and processor selection phase. In task prioritizing phase, tasks were ranked by their computation as well as communication cost. In processor selection phase, tasks were assigned to the processor which was first available.

While a significant initial result, subsequent research struggled to reduce the excessive time complexity of HEFT, which can be as be as high as $O(n^3)$ (where n is number of tasks). Also, experiment showed that HEFT's heuristics were easily trapped into local optimal [RB17].

As workflows become larger and larger, researchers turned to meta-heuristic methods. For example, Chen *et al.* [CZ09] used Ant Colony Optimization (ACO) [Dor06], whose pheromone function is a marker in decision space that attracts other candidate solutions. In that work, seven heuristics were applied to propose a pheromone function, such as reliability greedy, cost greedy, time/cost balance etc. In other work, Rodriguez *et al.* [RB14], found that particle swarm optimization (PSO) [Shi01] outperformed ACO as well as much other prior work. At the same time, 2014, Tsai *et al.* [Tsa14] proposed HHSA (hybrid heuristic-based scheduling algorithm) framework. HHSA was an ensemble method that ran separate ACO, PSO, and other evolutionary algorithms, then reported the best solution found by any method.

Figure 7.4 shows the general framework for evolutionary algorithms. For a problem with multi-objectives, one configuration c_1 *dominates* another configuration c_2 if it is better on at least one objective and worse on none; i.e.

$$\forall o \in obj\ o(c_1) \leq o(c_2) \text{ and } \exists o \in obj\ o(c_1) < o(c_2)$$

where *obj* are all objectives of the problem. All configurations which can not be dominated by another configuration form the *Pareto Frontier*. This frontier is used as the "parents" from which we

---

1. Generate population $i = 0$ using some *initialization policy*.
2. Evaluate all individuals in population 0.
3. Repeat until tired or happy
   (a) *Cross-over:* combine elite items to make population $i+1$;
   (b) *Mutation:* make small changes within population i;
   (c) *Evaluate:* individuals in population i;
   (d) *Selection:* choose some elite subset of population i.

Figure 7.4 Framework of Standard Multi-objective Evolutionary Algorithms such as NSGA-II, SPEA2, MOEA/D.

Table 7.1 Highly Cited Workflow Configuration Techniques from 2006 to present

	Author	Cited by	Optimization Strategy	Objectives
[YB06]	Yu, 2006	337	GA	makespan +cost
[Kim07]	Kim, 2007	114	SA	makespan
[CZ09]	Chen, 2009	293	ACO	makespan +reliability*
[Pan10]	Pandey, 2010	519	PSO	cost
[BM11]	Bittencourt, 2011	181	heuristic	cost
[Far12]	Fard, 2012	93	heuristic	makespan +cost
[Abr13]	Abrishami, 2013	264	heuristic	deadline +cost
[RB14]	Rodriguez, 2014	181	PSO	cost
[Tsa14]	Tsai, 2014	57	SA+GA +PSO+ACO	makespan
[DP14]	Durillo, 2014	45	heuristic	makespan +cost
[Mal15]	Malawski, 2015	230	heuristic	makespan +cost
[Zhu16]	Zhu, 2016	27	NSGA-II/SPEA2/ MOEAD	makespan +cost
This work			sampling-based	makespan +cost

* Objectives were combined to one by a weight vector.

build the next generation of candidates. Variants of MOEA differ in how they down-select from the general population to the frontier. SPEA2 [Zit01a] prefers solution which dominates more *number of* other solutions. While NSGA-II [Deb02] or other recently proposed algorithms sorts solutions by *dominance depth*, i.e. at which front is a solution located. MOEA/D [ZL07] decomposes the multi-objective into several single objective sub-problems and better solution can be reproduced from its neighboring sub-problem solutions. The selection policy we used later in RIOT is the NSGA-II non-dominated sorting. In non-dominated selection, we always pick up all configurations which can not dominated by any another candidates.

As to other work, in April 2017, we searched papers with the term “scientific workflow” and “scheduling” published in past decade in IEEE Xplore and ACM Digital Library. From all results, we selected “highly-cited” papers which have more than 10 citations per year since their publication. After skimming their abstracts and introductions, we found the 12 related papers in Table 7.1. Our reading of this literature is that the Zhu *et al.* [Zhu16] paper on EMSC is a comparative assessment of much of the previous work. EMSC explored the hypothesis that a little sequencing of the configuration task is very useful for configuring cloud environments. Specifically, EMSC encoded the cloud configuration by following 3-tuples scheme and proposed corresponding genetic operators (c.1 and c.2 in Figure 7.4).

- $\mathscr{I}$: Mapping tasks to instance);

- $\mathscr{O}$: Ordering tasks within the same instances;
- $\mathscr{T}$: Mapping instances to the available types of virtual machines.

Zhu *et al.* showed that off-the-shelf evolutionary algorithms such as NSGA-II [Deb02], SPEA2 [Zit01a], MOEA/D [ZL07] could be effectively figure out optimal configurations. Based on our literature reviews, we conclude that EMSC is the current state-of-the-art in cloud service configuration. The evaluation part of the Zhu *et al.* paper is every extensive and showed that EMSC achieved better configurations that a wide range of other approaches.

That said, as seen in our introduction, that algorithm suffers from a large optimization overhead: for workflows with many tasks, the optimization time. We attribute these long runtimes to the evolutionary algorithms used by Zhu *et al.* to assess options within $\mathscr{I}, \mathscr{O}, \mathscr{T}$. Hence, in the following, we adopt the $\mathscr{I}, \mathscr{O}, \mathscr{T}$ structure and look for faster ways.

7.2.4 How to Make a RIOT (Methodology)

RIOT uses the configuration scheme proposed by Zhu et al. [Zhu16] but find optimal of them in a different step.

Step1: Grouping Instances. Cluster the tasks onto virtual machine instances. This generates a vector v of size $V = |v|$ where each v_i can be one of eight types t_j shown in Table 7.2. We say that v is the *space of candidates* containing many candidates, each of which is one setting $\forall_{i \in v} v_i = t_j$. Typically, this first step results in a space of candidates of $V = 100$ so our search space has as many as 8^{100} candidates.

To map tasks into virtual machines, RIOT use TASKGROUP to cluster the tasks, and assign each cluster into one VM.

TASKGROUP defines the *critical tasks* as follows,

- It is the start task T_s, or
- It is a task whose data flow in-degree (number of edges incident to in workflow) is among the top third of all in-degress of all tasks

Next TASKGROUP assigns each task a probability p_i as

$$p_i = \begin{cases} 1.0 & task_i \text{is critical task} \\ \eta * \underset{j \in Pred(i)}{average}(p_j) & \text{otherwise} \end{cases}$$

where $\eta \in [0,1]$ is a control parameter.

With the probabilities, we can group tasks into the clusters. For each task, there is p probability to assign a new cluster. If one task does not map to new cluster, TASKGROUP maps it to any one of existing clusters (if it is a critical task) or any clusters of its predecessors (if it is not a critical task).

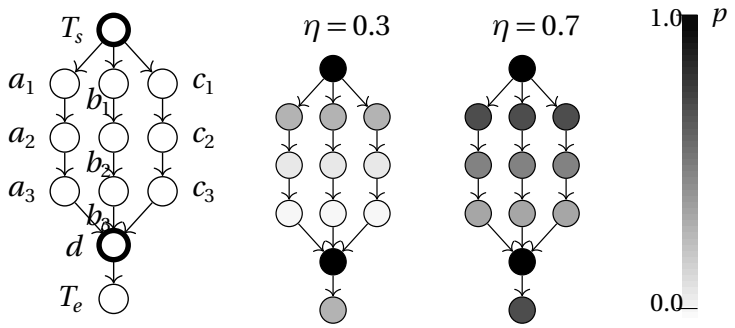


Figure 7.5 Left: a demonstrated workflow with critical tasks highlighted; Middle& Right: probability for each tasks to deploy into new VM with different η; $p = 0.0$ indicates that task always use existed VM, while $p = 1.0$ indicates that tasks always request new VMs.

Figure 7.5 demonstrates critical tasks and p assignment. As we can see, critical task separates the workflow into several "blocks". Tasks within one block are supposed to be executed in serial, therefore, in the same cluster. p value can control this– tasks closer to end of workflow have smaller p within a block, therefore, higher probability to assign to its predecessor's cluster.

From Figure 7.5, we can see that η is an parameter controlling number of clusters. Higher η implies more virtual machines might be applied in the deployment. To improve diversity (and explore more solutions), we set η as 0.05, 0.10, ... , 1.00 to generate 20 different task-instance mappings.

Step2: Finding Ordering. Determine which tasks to run first, second, etc on each instance.

RIOT used the B-Rank method of HEFT [Top02] to sort task execution priorities on the virtual machine. When sorting tasks, the B-rank metric is distance of the activity to the end of the workflow. Let T_e be the final task of a workflow). Then $rank(T_e) = 1$, and $rank(i) = 1 + \max_{j \in Succ(i)} rank(j)$

B-rank sorts tasks in decreasing order of $rank(i)$. Then, if two tasks i, j are ready to run on the same virtual machine, we select the task that is *lower* in this sort order (and ties are resolved randomly).

Step3: Determining Type. Determine the type of each instance using CLOUDSIM and the *surrogate sampling* method of Algorithm 4. CLOUDSIM is the standard simulator used in this area of research. As of August 2017, Google scholar reports that the original 2011 CLOUDSIM paper has 2492 citations. For our workflows, each run of CLOUDSIM takes under a second. This is a sub-routine called in the inner most loops of a cloud configuration algorithm so we must take care not to call this simulator too often. According, in Step3 we:

- Generate and evaluate only a few *anchor* candidates (say, 30).
- To this pool, we add a large number of, say, 500 *random*ly selected *unevaluated* candidates.

Table 7.2 Eight types of instances, sorted by price. Why do we explore just these eight types? These are nearly the same types explored in prior work [Zhu16], with some changes result from AWS EC2 service updates.

Type	Compute Unit	Bandwidth (MB/s)	Price($/hr)
m3.medium	3.75	85.2	0.067
m4.large	7.5	35.2	0.1
m3.large	7.5	85.2	0.133
m4.xlarge	15	68	0.2
m3.xlarge	15	131	0.266
m4.2xlarge	30	131	0.4
m3.2xlarge	40	131	0.532
m4.4xlarge	45	181	0.8

- To quickly guess the objective scores of this *random* pool (without incurring the simulation cost of CLOUDSIM) we use the surrogate sampling methods of Algorithm 4.
- Using these guesstimated objective scores, we apply the NSGA-II non-dominated sorting procedure to reduce the 500 candidates to just their Pareto frontier.
- CLOUDSIM then evaluates just the items in the Pareto frontier.

Final Step: Aggregation. Note that the first step *grouping step* is probabilistic so the above three steps are repeated 20 times (with different η).

Typically, the Pareto frontier returned from Step3 has (approx) 50 instances. RIOT reduces these 20*50=1000 candidates to a *penultimate set* of (approx) 20 candidates via a final call to the NSGA-II non-dominated sorting procedure.

Any one of the candidates in this penultimate set is a *final solution* we would recommend to the user. Since we do not know what user preferences may drive that final solution selection, in the experiments described below, we report a statistical analysis that compares the penultimate set to the Pareto frontiers generated by the other methods.

7.2.5 Evaluations

In this section we report numerical results of scheduling workflows in different structures as shown in Figure 7.2. Each structure scales 25 to 1000 tasks approximately (see `http://tiny.cc/wfeg` for more details). All workflows were supposed to deploy to Amazon AWS Cloud Services, with instances listed in Table 7.2.

In this work, we compared RIOT to two baseline schedulers. First is MOHEFT [DP14] (Multi-objective HEFT), a **heuristic algorithm** basing on the classic HEFT [Top02] method. Another is EMSC [Zhu16], a **meta-heuristic algorithm**. We ran EMSC basing on three popular MOEA, including NSGA-II, SPEA2 and MOEA/D.

Table 7.3 Median Runtime* among 30 Repeats in RIOT and Others

Model	MAKE SPAN*	RIOT (t)	MO-HEFT (t_1)	EMSC-NSGAII (t_2)	EMSC-SPEA2 (t_3)	EMSC-MOEA /D (t_4)	Speedup $\frac{\min(t_i)}{t}$
M.25	31	19	6	33	34	35	0
M.50	49	9	17	62	65	65	1
M.100	113	6	55	154	157	162	9
M.1000	257	250	1.6H	2.6H	2.9H	2.8H	22
E.24	0.4H	1	4	27	28	29	4
E.46	0.6H	2	14	58	59	61	7
E.100	3.6H	5	56	153	157	164	11
E.997	5.2H	211	1.6H	2.6H	2.7H	3.0H	27
I.30	655	1	6	34	35	36	6
I.50	954	2	15	62	64	65	7
I.100	700	7	63	175	182	199	9
I.1000	0.5H	189	1.5H	2.6H	2.7H	4.8H	29
C.30	124	1	7	34	35	36	7
C.50	198	3	16	63	63	65	5
C.100	241	7	61	154	154	162	8
C.1000	0.4H	273	1.7H	3.0H	2.6H	4.8H	23
S.30	1006	1	6	34	35	35	6
S.60	1152	3	23	76	77	79	7
S.100	1133	9	62	168	169	180	6
S.1000	1.2H	302	1.4H	2.0H	2.0H	2.2H	16

Runtime* is in seconds unless otherwise stated (H=hours).
Makespan* is the median makespan of all non-dominated scheduling found by any algorithm ran in the experiment.
Model M/E/I/C/S = Montage, Epigenomics, Inspiral, CyberShake, Sipht (see Figure 7.2)

We coded RIOT and two baseline tools in JAVA and ran them on the same machine (2.0GHz with 8GB memory, running in CentOS). For parameters of RIOT, by default, we set $N, n_0, n_T = \{500, 30, 8\}$ since we are using the eight types of Table 7.2. For other baselines, we strictly follow the setups defined in their associated publications.

To test performance robustness and reduce observational error, we repeated these all studies 30 times with different random seeds. To check the statistical significance of the differences between the algorithms, we performed a statistical test using Wilcoxon test at a 5% significance level.

Comparing via Runtime

Table 7.3 shows the runtimes of different treatments. J For convenience, we also report the *makespan* as well as speed-up of RIOT method.

From this table we observe:

- Measured in relative terms, except in a very small workflow (Montage25), we note that RIOT is 1-27x faster than other approaches.
- Measured in absolute values, the general trend is that other methods can take up to 4.8 hours while RIOT is never slower than 310 seconds. That is to say, RIOT terminates in just a few

Table 7.4 Median Measurements for all Experimented Workflows

	Hypervolume						IGD						Spread				
Model	RIOT	MH	EN	ES	EM	RAND	RIOT	MH	EN	ES	EM	RAND	RIOT	MH	EN	ES	EM
M 25	**79** *(1)*	38	82	80	70	47	**4** *(1)*	36	2	6	10	26	**83** *(1)*	91	92	55	71
M 50	**82** *(1)*	28	86	85	74	51	**4** *(1)*	42	6	4	10	22	**76** *(1)*	*n.a.*	105	62	77
M 100	**79** *(1)*	29	85	83	77	49	**3** *(1)*	48	2	8	22	30	**88** *(1)*	*n.a.*	123	81	81
M 1000	75 *(1)*	34	84	83	82	51	**3** *(1)*	50	0	2	10	24	**96** *(1)*	*n.a.*	104	55	93
E 24	**73** *(1)*	27	78	78	60	45	**8** *(1)*	45	2	5	14	28	**95** *(1)*	87	79	78	96
E 46	**67** *(1)*	0	68	68	62	28	**6** *(1)*	67	5	12	17	35	**79** *(1)*	*n.a.*	89	81	85
E 100	**75** *(1)*	6	70	69	64	35	**4** *(1)*	62	5	3	22	31	93 *(1)*	87	82	56	89
E 997	**80** *(1)*	0	68	67	65	31	7 *(1)*	156	3	7	7	32	**60** *(1)*	90	95	54	78
I 30	**72** *(1)*	29	79	75	63	46	**6** *(1)*	40	1	7	16	24	**95** *(1)*	92	104	70	90
I 50	**72** *(1)*	22	78	70	58	42	**4** *(1)*	39	1	5	16	22	**93** *(1)*	91	114	66	90
I 100	**69** *(1)*	0	73	71	68	35	**4** *(1)*	64	1	3	28	30	**80** *(1)*	*n.a.*	109	79	71
I 1000	**80** *(1)*	14	81	77	81	47	**4** *(1)*	49	0	2	12	32	**84** *(1)*	102	136	92	78
C 30	65 *(1)*	37	82	81	67	33	14 *(1)*	48	2	5	10	38	107 *(1)*	*n.a.*	91	42	83
C 50	**67** *(1)*	30	78	76	53	24	11 *(1)*	43	2	5	16	41	108 *(1)*	86	98	42	81
C 100	63 *(1)*	19	74	72	61	13	**9** *(1)*	55	3	5	16	55	110 *(1)*	95	91	51	95
C 1000	**79** *(1)*	39	81	81	80	49	**5** *(1)*	38	1	2	13	30	**86** *(1)*	*n.a.*	119	55	85
S 30	**72** *(1)*	28	74	74	23	10	**3** *(1)*	38	1	3	38	89	111 *(1)*	78	89	67	90
S 60	71 *(1)*	23	79	79	68	40	**7** *(1)*	49	3	6	13	32	**96** *(1)*	85	106	85	94
S 100	68 *(1)*	33	77	78	67	36	**5** *(1)*	41	2	5	14	33	**89** *(1)*	*n.a.*	103	85	88
S 1000	68 *(2)*	16	77	70	75	43	**5** *(2)*	48	1	5	8	32	**94** *(1)*	*n.a.*	127	79	71

Note: all values in the table are in $\mathbf{10^{-2}}$***, e.g., the median hypervolume of M 25 gained from RIOT is*** $\mathbf{79 * 10^{-2}}$***, i.e. 0.79***

RAND = Random search (Sanity check)

MH = MOHEFT; EN = EMSC-NSGA-II; ES = EMSC-SPEA2; EM = EMSC-MOEA/D

Hypervolume = higher are better; IGD = lower are better; Spread = lower are better

(values) next to RIOT are IQR (interquartile range, i.e. difference between 75th and 25th percentiles) of 30 repeats of RIOT

n.a. indicates no enough frontier points to calculate spread

Bold values indicate that RIOT performed as well as or better than any of MH/EN/ES/EM (under Wilcoxon Test).

seconds to minutes while other methods require minutes to hours.

- Comparing the expected makespan to optimizer runtimes, in some workflows with many tasks, such as Montage100, Montage1000, CyberShake1000, etc., previous methods were slower than the eventual runtimes (makespan); while RIOT requires just small ratio of makespans, making RIOT more suitable for re-scheduling in dynamic cloud environment.

To conclude, comparing via runtime, **RIOT finds schedulings much faster than the prior heuristic/ meta-heuristic methods.**

Comparing via Frontier Quality

In this work we treat workflow scheduling as a multi-objective problem. To compare the quality of returned frontier, three measures is widely applied [MA04] – Hypervolume, Inverted Generation Distance (IGD) and Spread. For our problem, we plot the objectives of frontiers in a 2D coordinate and have following definitions,

- *Hypervolume* is the area of space the obtained frontier dominated (top-right of the frontier

curve);

- *IGD* is average Euclidean distance of each point in obtained frontier to its nearest point in the true Pareto Frontier. It is almost impossible to find the true Pareto Frontier. Following Wang *et al.*'s guidance [Wan16], we collected non-dominated scheduling found by any algorithms in any repeats as the true frontier;

- *Spread* defines the average Euclidean distance of every pair of consecutive points in the obtained frontier. Lower spread implies better diversity.

Table 7.4 concludes the statistical measures for all workflows. In that table, **Bold** values indicate where RIOT performed as well as or better than any of MH/EN/ES/EM (under the Wilcoxon Test).

Within Table 7.4, we observe that:

- Variants of EMSC have similar performance; EMSC outperforms the heuristic method, MO-HEFT. This is consisted with EMSC's origin paper;

- Consider the IQR values[1]: the performance of RIOT is stable, even though it is a stochastic method;

- Measured by hypervolume, in 70% of experimented workflows, RIOT has significantly higher values than other baselines. That is, in most workflows, with monetary cost constraint, RIOT can find schedulings with less makespan, or within some deadline, RIOT can find schedulings required less cost;

- Measured by IGD, RIOT performs best in 85% workflow. In other words, RIOT's results are closer to true frontier;

- According to spread statistics, RIOT provides more diverse results in majority study cases;

- Summarizing all in Table 7.4, there is no any algorithm always performs the best (this is one of Wolpert's *No Free Lunch* results [WM97b]). However, RIOT performs best in majority of measurements and no bad in the remaining measures (compared to *RAND* or *MOHEFT* especially).

Summarizing above observations, **RIOT usually finds schedulings as good as anything else. This result is particularly remarkable for the large workflows**.

7.2.6 Threats to Validity

Sampling Bias While we tested RIOT on over two dozen workflows, it would be inappropriate to say that this sample covers the space of all possible workflows. As researchers, all we can do is to introduce our method, release the source code for our method and suggest that other researchers try a broader range of workflows.

[1] IQR = intra-quartile range = (75-25)th percentile.

Algorithm Bias In this work we compared our work to EMSC. We choose EMSC since it is the best method among all highly cited papers we have explored in this area. That said, there are many other ways to perform cloud configuration and one paper cannot assess them all.

To assist other researchers in exploring more configuration methods than those stated here, our reproduction package includes a full version of EMSC by implemented within the open source jMetal framework.

Evaluation Bias Following Wang's guidance [Wan16], we evaluated results by Hypervolume, Spread and IGD. There are many other performance measurements adopted in the community of software engineering. For example, some researchers take PFS (Pareto frontier Size) into consideration. PFS counts how many non-dominated solutions provided to the customers. Similar to spread, it is a diversity measure. Using various measures might lead to different conclusions. A comprehensive analysis using other measures is left to future work.

7.3 Summary of OSAP3

This section showed that the OSAP3 performed well in the study case workflow deployment. The kernel of OSAP3 is the linear surrogate model as is indicated in Figure 7.1. With this surrogate model, we can quickly evaluate, or specifically guess the objectives of the given configuration. However, the effectiveness of OSAP3 is also replied on this assumption. In other words, if the surrogate model loss the precision, then no words aforementioned is guaranteed.

The surrogate model requires the linearity of the SE model. For example, in the workflow deployment study case, we can check this assumption as follows: if someone increase the CPU/memory resource twice (multiple by 2 in decision space), then the workload terminating time could reduced, most probability by half (multiple by 1/2 in objective space). To some extent, this is the linearity of the model. The OSAP3 is highly replied on this assumption. Models without this assumption could fail in applying OSAP3.

CHAPTER

8

WORTHY, FOURTH GENERATION OF OSAP

This chapter is based on the following paper *(under review)* and some other empirical experiments.

- "On the Benefits of Restrained Mutation: Faster Generation of Smaller Test Suites" Submitted to IEEE/ACM International Conference on Automated Software Engineering (ASE 2019).

As is mentioned at the end of last chapter, the third generation of OSAP highly replies on the linearity of the model. However, no all SE models meet such assumption. As a simple example, in requirement engineering, the XOMO or POM3 model contains many features which has exponentially effect on the objective. As is shown in Figure 5.1, the `prec, flex, resl` etc. are the scale factors while some other factors are linearly decreasing efforts or linearly increasing efforts. A mix of these variables create a non-linear SE model.

To handle such kind of SE problems, OSAP4 introduces a novel surrogate model. We call this surrogate model is Δ-oriented.

8.1 Delta Oriented Surrogate Model

Role of the surrogate model in OSAP is the comparator or selector. In other words, given two or more configurations, the surrogate model should return which one is better, without frequently

Algorithm 5: Framework of OSAP4 – Delta Oriented Surrogate Model as the Selector

Input : model, assuming all objectives are to minimize
Output : pareto frontiers/optimal configuration

1 $Samples \leftarrow (n = 100)$ evaluated items
2 $PF \leftarrow$ pareto frontier in $Samples$
3 **foreach** $X \in PF$ **do**
4 $Neighbors \leftarrow$ Configurations near X in decision space
5 get all Δ_D^{pq} and $\Delta_{Oi}^{pq} (i = 1, 2, ...)$, where pq are pairs in $Neighbors$
6 train KNN model to predict Δ_{Oi} from Δ_D (i=1,2,...#of objs)
7 $Y \leftarrow$ random configuration
8 predict Δ_{Oi}^{XY} given Δ_D^{XY}
9 If exists i such that $(\Delta_{Oi}^{XY} \ll 0)$, evaluate Y using model
10 repeat Line 7-9, or Goto 3
11 Collect all new evaluated configurations, update $Samples$
12 Goto 2 or Terminate
13 Return all pareto frontiers achieved

calling the original complex SE model. If the new candidate is better than the old one, then we should pick that; otherwise, drop that.

To perform as the comparator or selector, apparently, the surrogate model can directly build up some mechanisms to estimate the objectives for the candidates. With the estimated objectives, it can definitely compare the configurations and therefore select the better one. This is exactly what the third generation OSAP does. However, one question is: is that really necessary to get the *value* of objectives for the candidates?

The answer to above question is *no*. Consider two configurations p and q,

- they have the decisions as $D(p)$ and $D(q) = D(p) + \Delta_D^{pq}$,
- where Δ_D^{pq} is their delta vector in decision space.
- There is *no need* to get estimation $\widehat{O_1}(p), \widehat{O_2}(p), \widehat{O_1}(q), \widehat{O_2}(q)$ etc. Here $\widehat{O_i}$ is the estimation of objective i.
- What we really need is the $sign(\widehat{\Delta}_{O1}^{pq})$, $sign(\widehat{\Delta}_{O2}^{pq})$, and so on.

Here further explains the last point. The Δ_{Oi}^{pq} is the diff of objective i between p and q, and similarly, the $\widehat{\Delta}_{O1}^{pq}$ is the estimation of that diff. It is not even necessary to get the estimation of $\widehat{\Delta}_{Oi}$. What is really important is the sign of that. By saying the "sign", OSAP4 adopts a generalized definition – there are three signs: significantly greater than 0, significantly less than 0, minor values.

The kernel of surrogate model for OSAP4 is to estimate the sign of Δ_O. It uses the Δ_D^{pq} as independent variables and the Δ_{Oi}^{pq} as dependent variable. Please note that as long as the sign is correct, the surrogate model is helpful.

In view of above, OSAP4 performs as Algorithm 5. In this algorithm, we first have a set of evaluated samples. Then for every configuration in pareto frontier X, we build a learning model for each objective. Such model's independent variables are the pairs of deltas in decision space, and the dependent variables are the deltas in objective space. Through the exploration, we found that the KNN can do a good job. To better explore the "landscape" near X, OSAP4 limits the inputting pair be the neighbors of X. Please note that the learning model is built for each pareto froniter, and for each objective.

OSAP4 is also known as WORTHY since it is seeking for the worthy delta to push forward the configurations within the decision space.

8.2 Case Study I: Revisit XOMO&POM3

This section revisits the XOMO and POM3 SE models. For details of these models, please see the former chapter.

8.2.1 Can the KNN Model Successfully Get the Sign of Δ_{Oi}?

The first research question to be explored is whether the surrogate model, specifically the KNN (k-nearest neighbors) can successfully predict the *sign* of Δ_O. To verify this research question, we performed the following experiment:

- step 1: randomly generated a set of samples
- step 2: evaluate all these samples
- step 3: select partial of the samples;
- step 4: for each pair of sampled configurations p and q, get the Δ_D^{pq} and Δ_{O1}^{pq}
- step 5: after getting all training pair info, train a KNN model which can predict Δ_{O1} given Δ_D
- step 6: get pairs of configurations not selected in step4, applying the KNN model to predict $\widehat{\Delta}_{O1}$
- step 7: plot the $\widehat{\Delta}_{O1}$ vs Δ_{O1}
- REPEAT 4 - 7 for all other objectives.

Figure 8.1 and Figure 8.2 reveal the relative error between $\widehat{\Delta}_{Oi}$ and Δ_{Oi1}. Please note that it is not necessary to get precise prediction – what really matters whether the sign of $\widehat{\Delta}_{Oi}$ is identical to the sign of Δ_{Oi}. The area within $-1 < \widehat{\Delta}_{Oi}, \Delta_{Oi} < 1$ is not so interested. The important observation is that the area in the second and forth quadrants are almost *empty*. In other words, the KNN predictor can return the correct signs of Δ_{Oi}. Therefore, the answer to this fundamental research question is that

The KNN surrogate model successfully got the sign of Δ_{Oi}.

Next we will see the effectiveness and efficiency of the OSAP4.

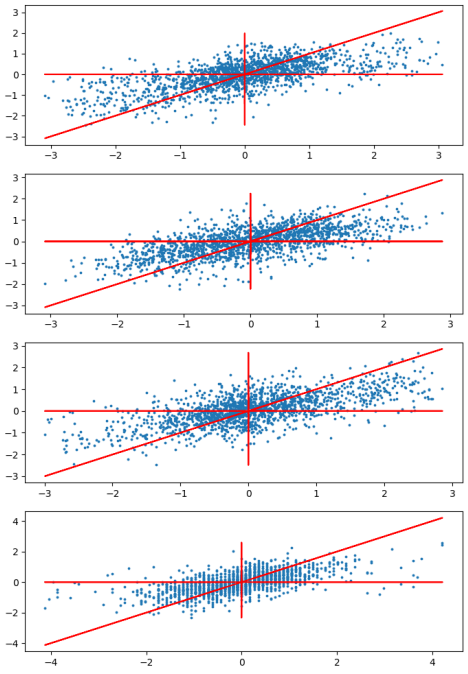
8.2.2 Comparing Effectiveness on OSAP1 and OSAP4

We adopt the same experiment design as was introduced in §5.2.3. This subsection is comparing the effectiveness of the results. We used the same evaluation metrics as former generation.

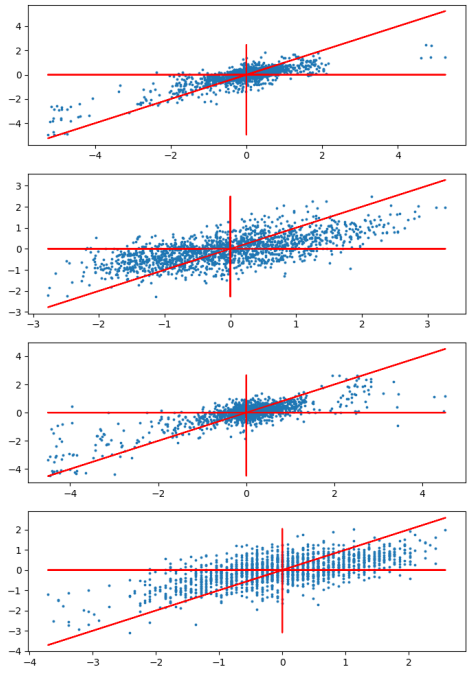
It is worthy to mention that here we fixed a small issue on the illustration of the results in the previous publication. As is introduced in §3.4.2, we need to do the normalization inside the objective space, before getting the diversity or convergence metrics. The previous publication did the normalization on all candidates, while the updated normalization here introduced in this chapter was performed under the pareto frontier only. At a first glance, it was not an issue in previous method. However, the range of values in objectives space inside the pareto frontier set would be smaller than that among all candidates, since the latter situation yields larger maximum values. In view of the actual application, we did the normalization based on the range formed by pareto frontier only. This might enlarge the differences between the methodology, putting SWAY in a disadvantage place.

Figure 8.3 and Figure 8.4 shows all metrics for the XOMO and POM3 models respectively. From them, we have the following observations:

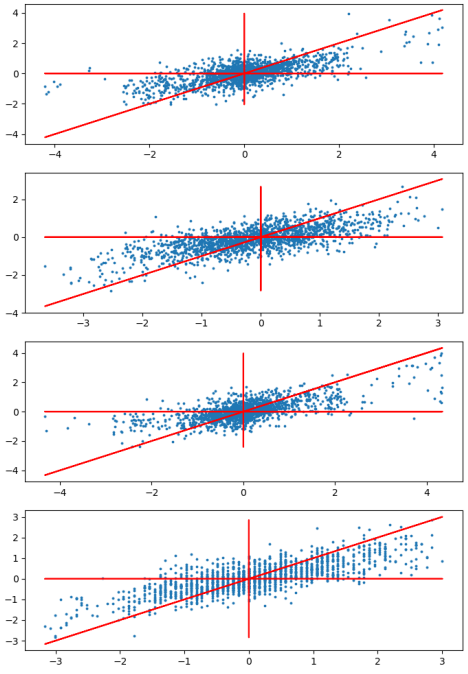
- SWAY is *not* stable. The hypervolume in all models shows this.
- Compared to other baselines, the WORTHY performed very stable. On average, the IQR over 20 WORTHY repeats is only 32% of that of SWAY repeats. GROUNDTRUE(GT) method is the most stable one. The WORTHY is 18% worse than that, measured by the IQRs. However, the WORTHY metrics are still significantly stable than SWAY.
- In the XOMO models, WORTHY performed as well as the GT; and it was even better than the MOEA sometimes (e.g. the gd@flight, etc.)
- In the POM3 models, WORTHY has similar performance as the GT, both of which were slightly worse than the MOEA.



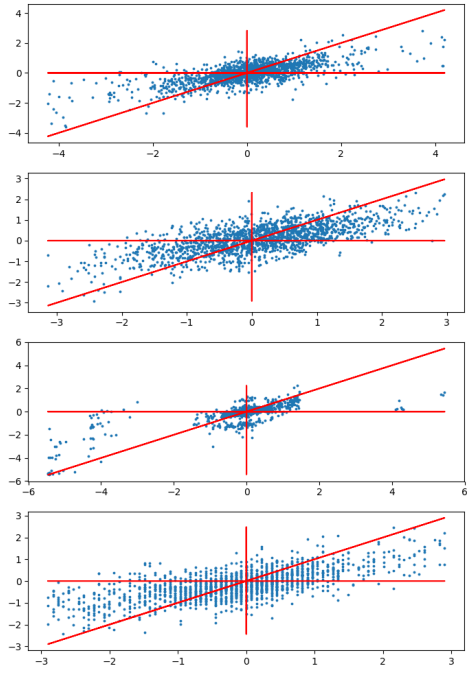
(a) OSP



(b) OSP2

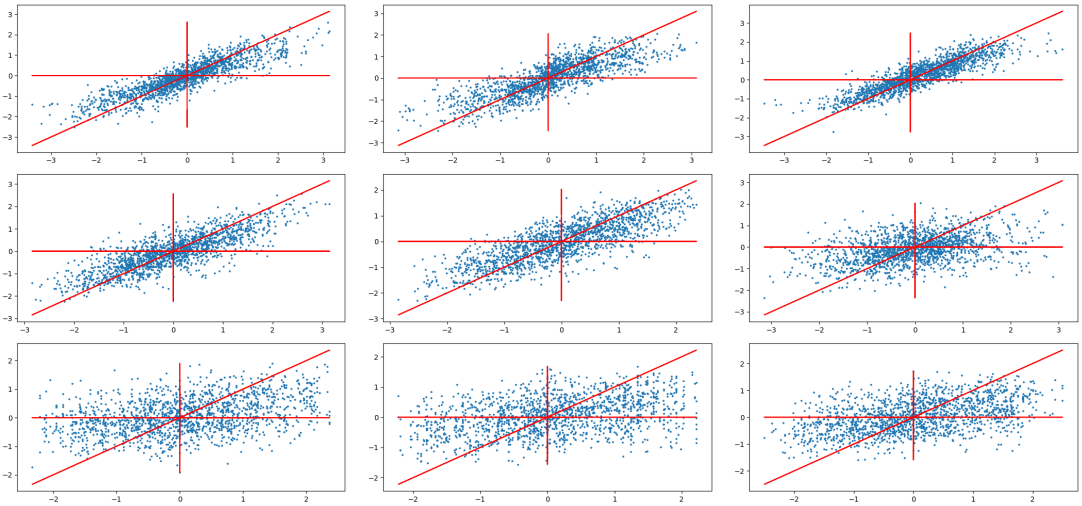


(c) Ground



(d) Flight

Figure 8.1 Predict $\Delta_O(y$ axis) vs. actual $\Delta_O(x$ axis) in XOMO models. All values are normalized by the standard deviation. The red line from lower-left to upper-right is the function $y = x$, i.e. the perfect predictions. Each subplot is an Δ_{Oi} prediction-actual values.



(a) Pom3a **(b)** Pom3b **(c)** Pom3c

Figure 8.2 Predict Δ_O(y axis) vs. actual Δ_O(x axis) in POM3 models. All values are normalized by the standard deviation. The red line from lower-left to upper-right is the function $y = x$, i.e. the perfect predictions. Each subplot is an Δ_{Oi} prediction-actual values.

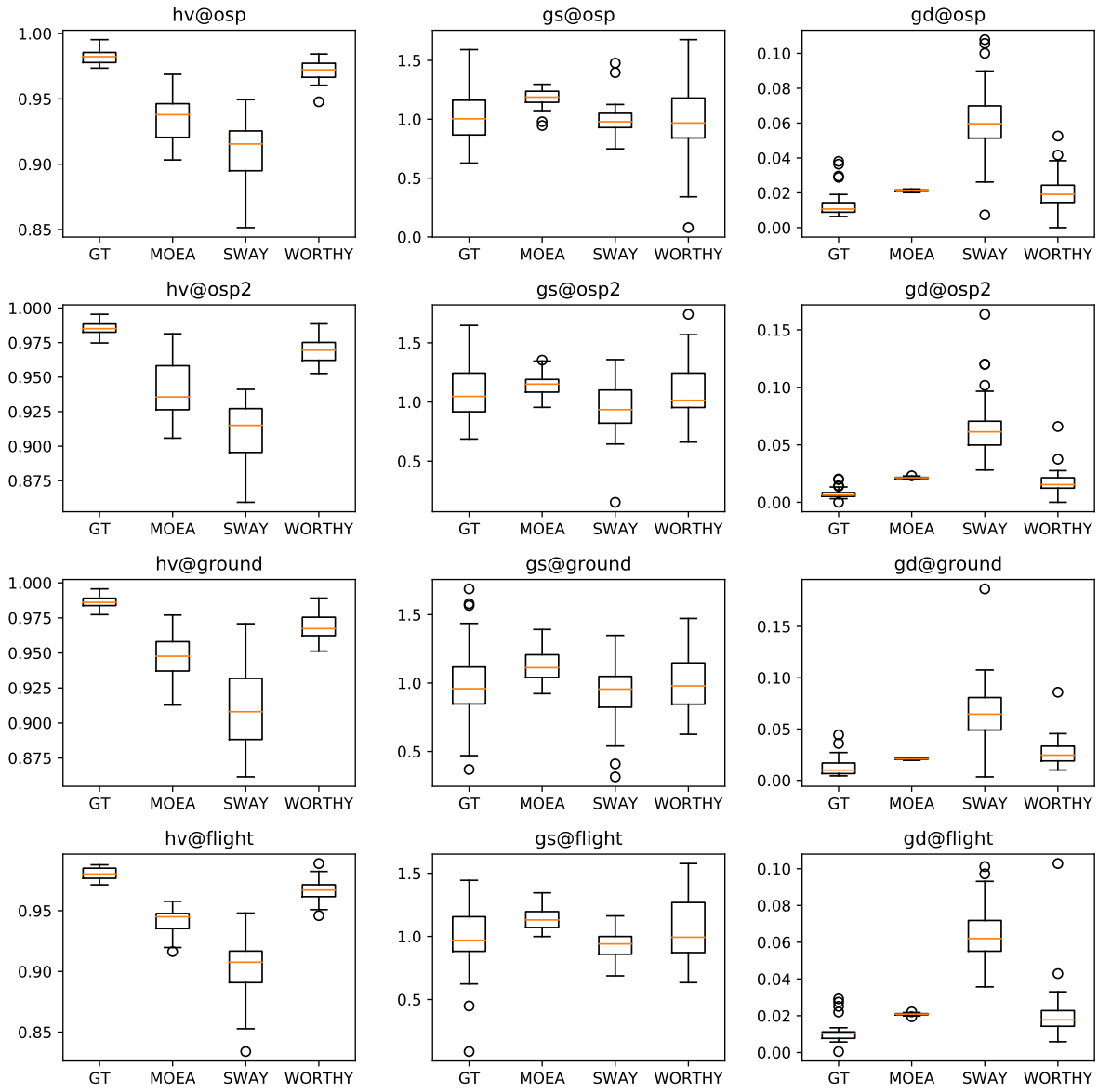


Figure 8.3 Comparisons on effectiveness for the XOMO models. hv=Hypervolume, higher the better. gs=General Spread, lower the better. gd=General Distance, lower the better. (see section 3 for details) The boxplot represents the metrics over 20 repeats.

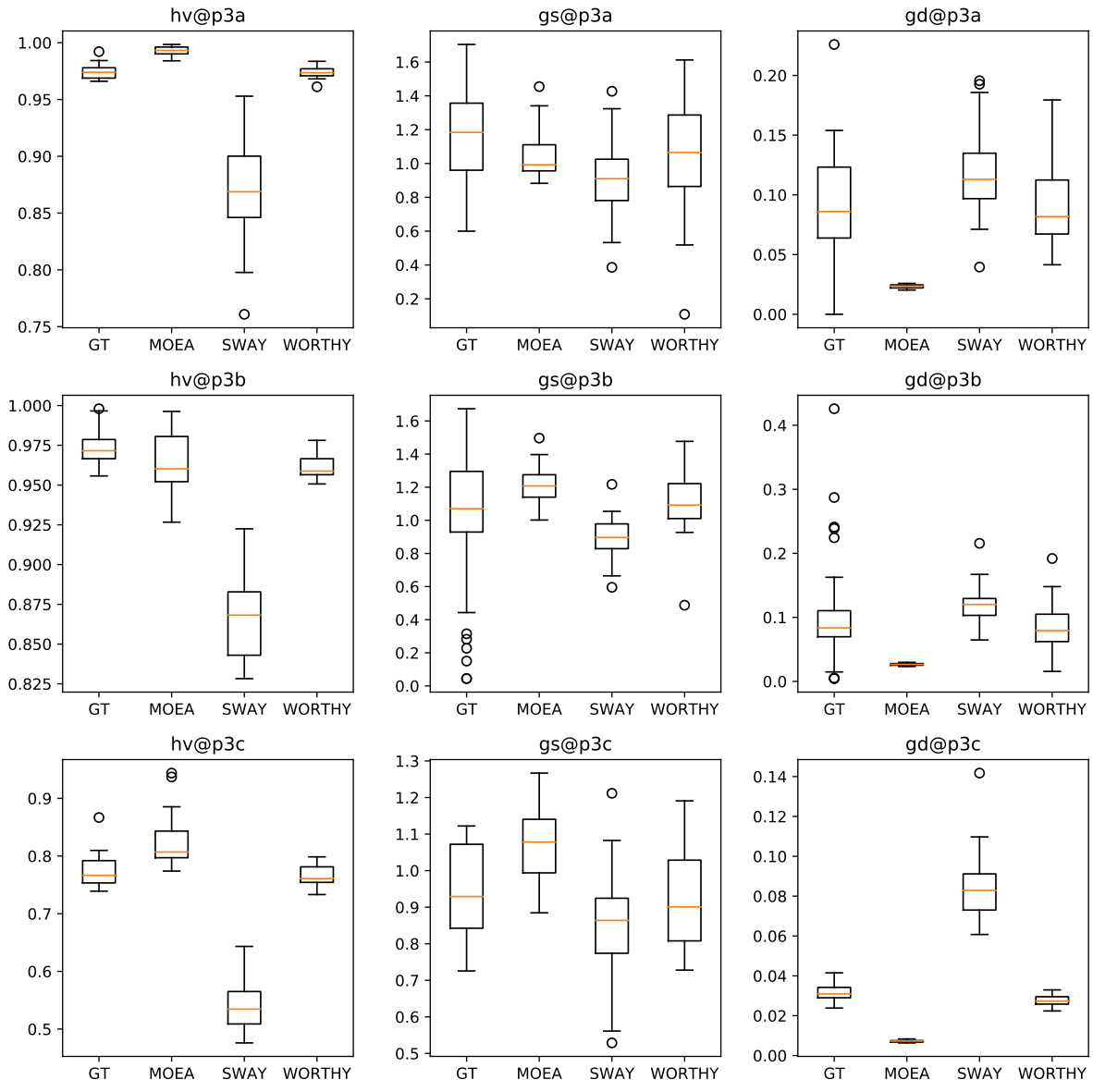


Figure 8.4 Comparisons on effectiveness on SWAY(OSAP1) and WORTHY(OSAP4) for the POM3 models. Formats are the same as Figure 8.3.

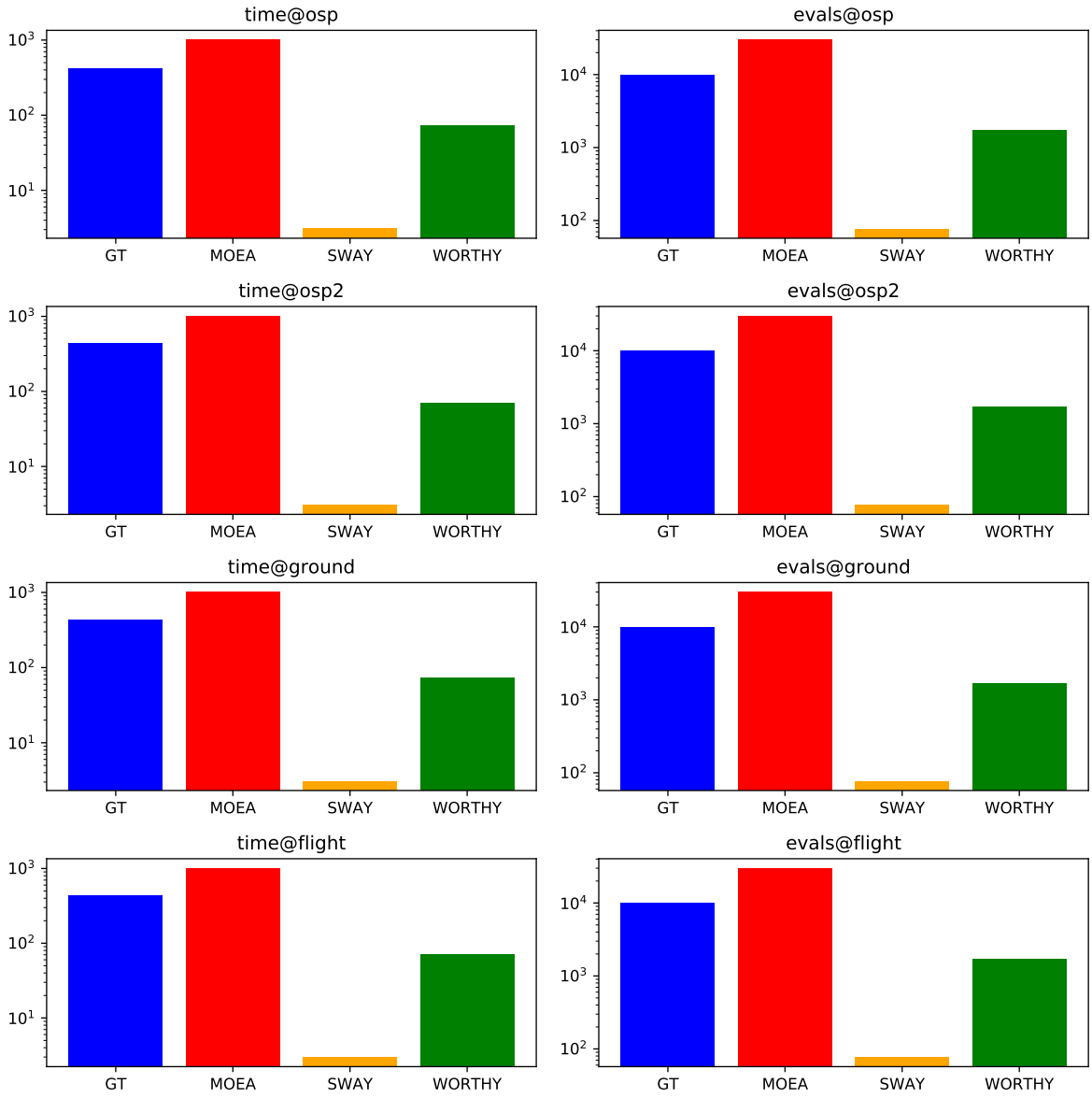


Figure 8.5 Average runtime and number of model evaluations over 20 repeats for the XOMO models.

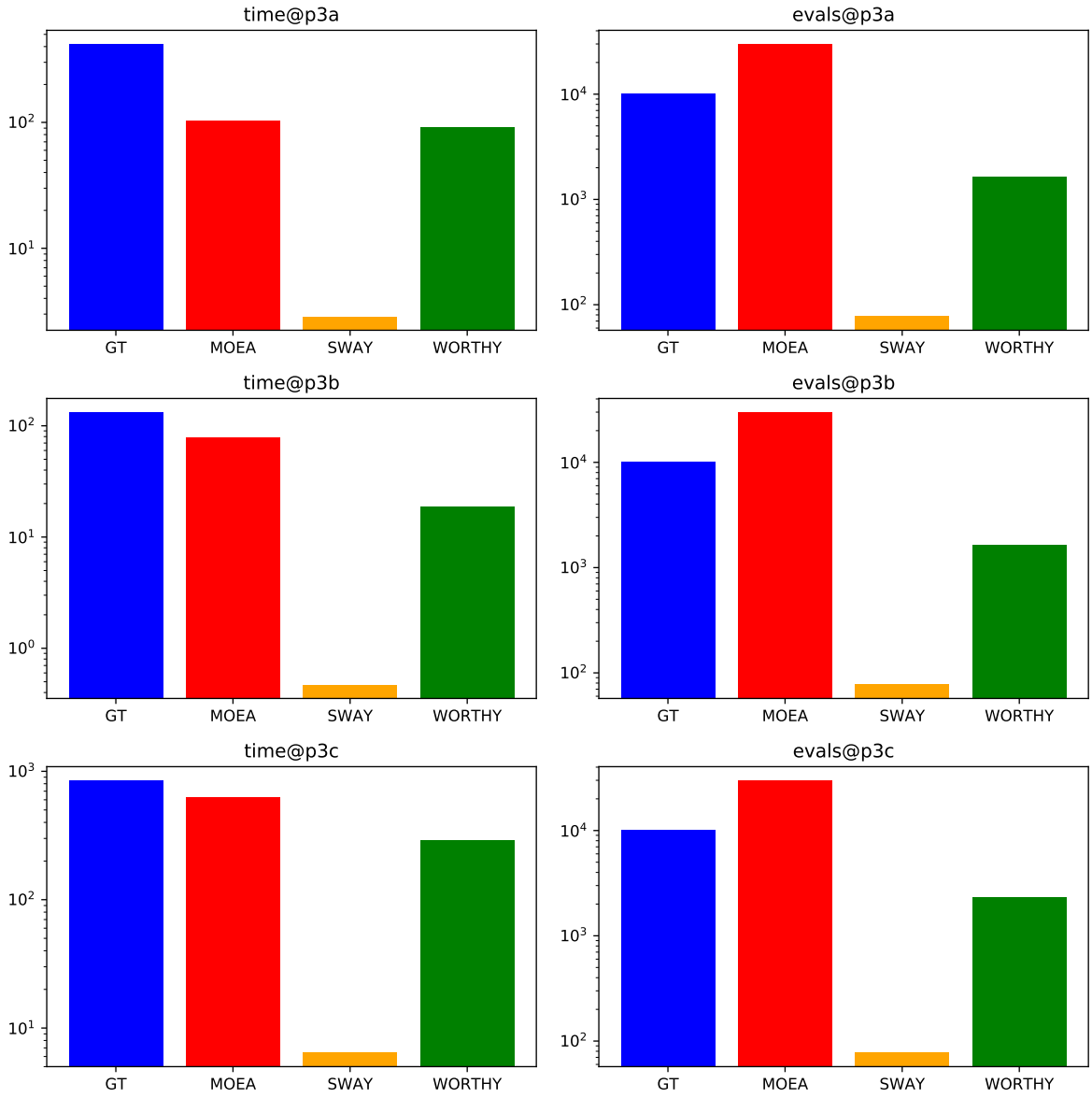


Figure 8.6 Average runtime and number of model evaluations over 20 repeats for the POM3 models.

According the these observations, we can say that: 1) WORTHY is stable and 2) results of WORTHY is comparable to the state-of-the-art MOEA methods.

8.2.3 Comparing the Efficiency

Figure 8.5 and Figure 8.5 illustrates the algorithm efficiency for XOMO and POM3 models respectively. Please note that all y-axis are in log scale.

The first observation is that the ratio of running time of any two methods was similar to the ratio of number of evaluations. In other words, the majority cost of all algorithms were in the model evaluations. One wired thing was the runtime of GROUNDTRUTH is larger than that of MOEA. That's because the process getting the pareto frontier took some time.

Secondly, the SWAY is always is the fastest one, because it just quires less than 20 model evaluations. On average, the WORTHY requires 31% (24%) of the runtime (model evaluation) of the GROUNDTRUE's. Measured by the number of model evaluations, the WORTHY saved as least 60% of efforts, compared to the state-of-the-art MOEA.

Combining the observations of efficiency as well as the effectiveness discussed just now, we suggest to use first generation OSAP(SWAY) only if the model is hard to evaluate, otherwise use the fourth generation (WORTHY).

8.3 Case Study II: Test Suite Generation

8.3.1 Problem Formulation

Target of this case study is to generate sufficient valid and diverse solutions to the constraint models returned by symbolic execution tools. To illustrate this, consider the code script in Figure 8.7.

In function `int mid(int, int, int)`, the symbolic execution tools track workflows of the program and tell us that the behavior of `mid` is determined by three integers, i.e. `x`, `y` and `z`. Specifically, `mid(int, int, int)` explores six paths:

```
path 1: [C1: x < y < z] L2->L3
```

```
1 int mid(int x, int y, int z) {
2  if (x < y) {
3    if (y < z) return y;
4    else if (x < z) return z;
5    else return x;
6  } else if (x < z) return x;
7  else if (y < z) return z;
8  else return y;
9 }
```

Figure 8.7 A simple C++ code script calculating the medium of three integers. [SK12]

```
p cnf 11511 41411
...
-11507 11510 0
-11510 11504 11507 11502 0
...
```

Figure 8.8 Script of a CNF benchmark(*LoginService2.sk_23_36*) explored in this work. See text for introduction to the format.

```
path 2: [C2: x < z < y] L2->L3->L4
path 3: [C3: z < x < y] L2->L3->L4->L5
path 4: [C4: y < x < z] L2->L6
path 5: [C5: y < z < x] L2->L6->L7
path 6: [C6: z < y < x] L2->L6->L7->L8
```

Given the relations of x, y and z expressed as $C_i (i = 1 \dots 6)$, the input of them should be $\vee C_i$; in other words, any of C_i is feasible. Conventionally, $\vee C_i$ can always be equivalently transformed into its dual form $\wedge C'_j$, or SMT(Satisfiability Modulo Theories) [BT18]. Furthermore, as x, y and z are integers, we can use 7 (less or more, depending on the tools) bits representing them and then transfer all variables into the boolean bits with the SMT conversion tools [Fin15].[1]

Output of aforementioned process is the CNF(conjunctive normal form). In Boolean logic, a CNF form has two or more boolean literals, or **variables** in text; and disjunction of some variables contributes to one **clause**. The CNF is a conjunction of one or more clauses. Figure 8.8 shows partial of a benchmark explored by the experiment. Line 1 indicates that that CNF benchmark has 11511 variables, with which filling 41411 clauses. Remaining lines shows details of all 41411 clauses, with zeros at the end as end-of-clause signal. For example, line 2-5 should be parsed as

$$\dots \wedge (\neg 11504 \vee 11510) \wedge (\neg 11510 \vee 11504 \vee 11507 \vee 11502) \wedge \dots$$

With CNF built up, a valid assignment, or **solution** in the text, to all variables can be revealed to the input of a program, i.e. a test case. Searching for sufficient valid solutions provide the software engineers a set of test suite. Feldt *et al.* [Fel16] showed that the diversity of solutions is proportional to the code coverage. Consequently, we need to find diverse solutions to constraint models.

8.3.2 Related Work

The 3-SAT problem is NP-complete [Tov84]. Solving constraint problems in theory proving is therefore NP-hard[2]. This problem has been explored for decades. This session reviews all related research works in this area.

[1] In this work, we discussed the constraints with boolean variables. Searching for SMT directly is left to the future work.
[2] Unless the model is 2-SAT, which is rare.

Table 8.1 Related work for solving theory proving constraints via sampling over recent decades.○/●: the absence / presence of corresponding item. ◑: only partial benchmarks *(the small benchmarks)* were reported.

Reference	Year	Citation	Sampling methodology	Benchmark size	Verifying samples	Distribution/ diverse reported
[Yua99]	1999	105	Binary Decision Diagram	≈1.3K	○	○
[Iye03]	2003	50	Interval-propagation-based	200	○	○
[Yua04]	2004	54	Binary Decision Diagram	< 1K	○	○
[Wei04]	2004	141	Random Walk + WalkSat	*No experiment conducted*		
[GD11]	2011	88	Sampling via determinism	6k	○	○
[Erm12]	2012	25	MaxSat + Search Tree	*Experiment details not reported*		
[Cha14]	2014	29	Hashing based	400K	○	◑
[Cha15]	2015	28	Hashing based (paralleling)	400K	○	◑
[Mee16]	2016	29	Universal hashing	400K	○	◑
[Dut18]	2018	5	$\mathbb{Z}3$ + flipping mutation	400K	○	◑

One category of methodology to solve the theory proving problem is to simplify or decompose the CNF forms via theoretical proving. A recent example was the *GreenTire* proposed by Jia *et al.* [Jia15]. *GreenTire* supports constraint reuse based on the logical implication relation among constraints. Advantage of such kind of methods is their efficiency guarantees. Similar to the analytical methods in linear programming, they are always applied to specific class of problem. However, even with the improved theory proving/solver, such methods may be difficult to be adopted in large models. *GreenTire* was tested in 7 benchmarks. Each benchmark was corresponding to a small code script with tens lines of code, e.g. the `BinTree` in [Vis06]. For the larger models, sampling is another research direction.

The other category of methodology – sampling techniques, were based on constraint solvers, e.g. $\mathbb{Z}3$, together with the extra sampling heuristics. These algorithms took all assignments of the problem as searching space and sampled partial of them for the program testing or verification purposes. Here concludes some sampling tools over the pass 20 years.

The earliest sampling tools were based on binary decision diagrams (BDDs) [Ake78]. Yuan *et al.* [Yua99; Yua04] build a BDD from the input constraint model and then weight the branches of the vertices in the tree so that walks from root to the terminal vertex are able to generate samples with desired distribution.

Iyer proposed a technique named *RACE* which has been applied in multiple industrial solutions [Iye03]. *RACE* builds a high-level netlist model to represent the constraints and implements a branch-and-bound algorithm for sampling diverse solutions. The advantage of *RACE* is its implementation simplicity. However, the *RACE*, as well as the BDD-based approached introduced above, return highly biased samples, that is, highly non-uniform samples.

Given the SAT solver WALKSAT [Sel93], Wei *et al.* [Wei04] proposed *SampleSAT*. *SampleSAT* combines random walk steps with greedy steps from WALKSAT. This method works well in small constraint models. However, due to the greedy nature of WALKSAT, the performance of *SampleSAT* is highly skewed as size of the constraint model increases. In 2011, Gogate *et al.* move forward the frontier via developing effective importance sampling algorithms for mixed probabilistic and deterministic graphical models.

For seeking diverse samples, universal hashing [Man93] techniques were used. These algorithms were designed for strong guarantees of uniformity. Meel *et al.* [Mee16] provided an overview of key ingredients of integration of universal hashing and SAT solvers. With universal hashing, we can generate uniform solutions to a constraint model. Also, these algorithms can be applied to the extreme large models (with near 0.5M variables). Limitation for these algorithms is their computational complexity.

More recently, several improved hashing-based techniques have been purposed to balance the scalability of the algorithm as well as diversity (i.e. uniform distribution) requirements. For example, Chakraborty *et al.* proposed an algorithm named *UniGen* [Cha14], following by the *Unigen2* [Cha15]. *UniGen* provides strong theoretical guarantees on the uniformity of generated solutions and has applied to constraint models with hundreds of thousands of variables. However, the *UniGen* still suffers from the large computation resource requirement. Later the *Unigen2* parallels *Unigen* and achieved near linear speedup of the number of CPU cores.

To the best of our knowledge, the state-of-the-art technique is the *QuickSampler* [Dut18] in solving this puzzle. *QuickSampler* was evaluated on large real-world benchmarks, some of which has more than 400K variables. *QuickSampler* outperforms aforementioned Unigen as well as another similar technique named *SearchTreeSampler* [Erm12].

QuickSampler starts from a set of valid solutions generated by $\mathbb{Z}3$. And then figure out the diffs between the solutions. The diffs of two solution σ_a and σ_b is defined as $\delta = \sigma_a \oplus \sigma_b$ where $\oplus$ is the XOR in bit-vector. Given the diffs, *QuickSampler* applies the diff into another solutions, generating new bit-vector. Such samples are not guaranteed to be valid. But according to Dutra *et al.*'s experiment, in majority benchmarks, the valid rate for the samples can be higher than 70%. Consequently, *QuickSampler* is an agile tool to generate larger amount of valid samples to the constrain models.

8.3.3 Motivation of this Work

Latest techniques aforementioned [Erm12; Dut18] are able to generate thousands, or even millions of solutions within hours. Such kind of test suite indeed guarantees abundant diverse. However, researchers are also interested in another open problem – test case prioritization, or the simpler test case selection. The test case prioritization/selection techniques schedule/choose test cases for execution in order to increase their effectiveness at meeting some performance goal, e.g. locating bugs/defects. Myers *et al.* pointed out that software testing utilizes approximately 40%-50% of total (human and CPU) resources, 50%-60% of the total cost of software development [Mye11].

Therefore, test case prioritization/selection techniques contributes a lot in reducing the software cost. Constructing the test suite with fewer cases inside is an exciting task.

New tools can handle introduced in §8.3.2 larger and larger benchmarks, making the theory proving executable in practice. Recent tools also emphasized the distribution of the solutions [Cha14],[Cha15] ,[Mee16],[Dut18]. However, there still exist three limitations in the state-of-the-art tool – QuickSampler. Purpose of this work is to overcome the corresponding limitations.

For fist limitation – the independent support requirements, our work should avoid that. On one hand, the independent support significant reduces the complexity of sampling space. According to Dutra *et al.*, number of variables in the benchmarks ranges from 100 to 400K, approximately; while in majority cases, the size of independent support are less than 100, with 481 at maximum. However, on the other hand, getting the independent support is not trivial. It is still an open question. The QuickSampler assumes that the independent support is known. For the software engineers, getting the independent support needs hours. And the independent support changes in every regression testing. Therefore, the independent support known assumption prohibits the engineers in real-world practice. Removing this assumption is the first task.

Second limitation is the time of verifying all solutions. As new samples are not generated from SAT solver or other process with theoretical proven, not all samples are valid. Even though the valid rate of them was quite high, engineer still care only about the valid ones. In Dutra *et al.*'s experiment, evaluating the validness of all samples took much longer time of the QuickSampler itself. Therefore, improving the efficiency of sample verification is another task.

Finally as mentioned above, software engineers want to get minimum test suites with abundant diverse; because it is not practical to run all tens of millions of samples returned by QuickSampler (or other tools). Consequently, the new method should achieve considerable diverse with less test cases.

8.3.4 Apply OSAP4 to Test Suite Generation

The test suite generation problem is suitable for OSAP4, since it is very difficult to get the large amount of initial samples. But we still need to make some change so that this problem can fit into Algorithm 5. Note that Algorithm 5 is designed for the search-based software engineering problems with multi-objectives. In the test suite generation, we do not need to evaluate the configurations. What we need to do is to figure out enough (diverse) valid test cases in a short time. Therefore, we have to adjust the Algorithm 5 so that it can be applied to test suite generation problem.

First of all, we don't have the pareto frontier (PF) in line 2. Line 2 is changed to use the center of clusters of the *Samples*. To get the clusters, we can use k-means algorithm.

Second, to increase the diversity, we were not just using the *Neighbors*, instead, we use the whole samples set.

Third, in line 5, we did not have the Δ_{Oi}^{pq}; as a result, we cannot, or need not, to train the KNN model. What we should do is just get all Δ_{D}^{pq} and their corresponding frequency. We just select the Δ_D with high frequency first.

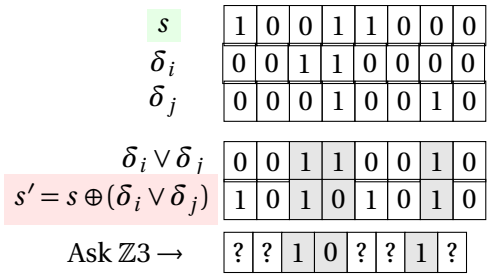


Figure 8.9 A demonstration of $\mathbb{Z}3$ fixing procedure. Given s, δ_i and δ_j, the candidate s' is generated. Assuming s' is invalid after the verification, WORTHY first utilizes $\delta_i \vee \delta_j$ as the mask to retain some bits, i.e. the hightlighted bits, then ask $\mathbb{Z}3$ to get assignment of all other bits.

To sum up, we have the following steps designed for the test suite generation

- step1: get *Samples* $\leftarrow (n = 100)$ valid test cases
- step2: get all Δ_D^{pq}, where pq are paris in *Samples*. And the distance is defined by exclusive-or (common for binary variables).
- step3: calculate the frequency for each Δ_D
- step4: run k-means in *Samples* to get the PF (Analogy to pareto frontier in multi-objective problems)
- step5: for each sample in PF, apply (use XOR) (one or two) deltas to the old sample. Note that deltas with high frequency should be applied first.
- step6: for each new samples generated in step5, verify that. If not true, fix that via Figure 8.9.

8.3.5 Experiment Design

Our work is implemented in C++, with the $\mathbb{Z}3$ v4.8.4 (the latest release when the experiment was conducted). The k-means component was provided by free edition of ALGLIB [BB13], a numerical analysis and data processing library delivered for free under GPL or Personal/Academic license.

We compared this work to state-of-the-art technique QuickSampler, purposed by Dutra *et al.* in 2018. To get fair comparison, we updated the $\mathbb{Z}3$ solver in QuickSampler into the latest version. QuickSampler does not integrate the samples verification into the workflow. In the experiment, we adjusted the workflow of QuickSampler so that all samples are verified before termination. Also, the original outputs of QuickSampler were the assignments of independent support. However, as a software engineer, he or she wants to get the full assignment of all variables. Consequently, for valid samples, we extended the QuickSampler to get full assignment from independent support's assignment via propagation.

To reduce the observation error and test the performance robustness, we repeated the experiment 20 times with 20 different random seeds. To simulate real practice, such random seeds were

used in $\mathbb{Z}3$ solver (for initial solution generation), ALGLIB (for the deltas clustering) and other components. Due the space limitation, we cannot report results for all 20 repeats. Correspondingly, we report the medium of all repeats under each metrics.

All experiments were run in the machines with Xeon-E5@2GHz and 4GB memory, running CentOS.

The benchmarks in this experiment inherit from experiments in QuickSampler. We choose these benchmarks since

- we want to compare our method to QuickSampler over same benchmarks;
- the benchmarks are online available;
- these benchmarks were adopted in multiple works, including [Cha14; Cha15; Mee16; Dut18] etc.

The benchmarks include bit-blasted versions of SMTLib benchmarks, ISCAS89 circuits augmented with parity conditions on randomly chosen subsets of outputs and next-state variables, problems arising from automated program synthesis and constraints arising in bounded theory proving. In a nutshell, the benchmarks are representative of scenarios engineers met in software testing or circuit testing in embedded system design. For more introduction of the benchmarks, please see [Cha15; Dut18].

It is worth mentioning that we omitted the benchmark *diagStencilClean.sk_41_36* in the experiment, since purpose of this work is to sample a set of valid solutions to meet the diversity requirement; while there are only 13 valid solutions from this model. The QuickSampler spent 20 minutes (in average) to search for one solution.

We will not report partial results in three *blasted* benchmarks marked with a star(*) in Table 8.2. Based on the experiment, we found that even though the QuickSampler generates tens of millions of samples for them, all samples were the assignment to the independent support. We could not figure our other bits beyond the independent support with $\mathbb{Z}3$. Reasons for that may come from the validity of independent support to these benchmarks. Therefore, we will not report the comparison of WORTHY vs. QuickSampler, in these benchmarks. In fact, solving or sampling on these benchmarks is not difficult; since they are all tiny, as compared to other larger benchmarks.

Table 8.2 concludes the attributes of all 30 benchmarks. We can see that number of variables ranges from hundreds to more than 486K. The large benchmarks have more than 50K clauses, which are extremely huge.

For clear analysis, we divide the benchmarks into three groups – the small benchmarks with vars $< 6K$; the medium benchmarks with $6K <$ vars $< 12K$ and the large benchmarks with vars $> 12K$. They can be distinguished via row colors in Table 8.2.

8.3.6 Are the Deltas Transferable?

As we can see in §8.3.4, WORTHY is based on the following assumption. *(In fact, the* OSAP4 *framework is also based on the* Δ *learning trick.)*

Table 8.2 Benchmarks overview. Benchmarks are sorted by number of variables. Medium benchmarks are highlighted with blue rows while the large ones are in orange rows. Three benchmarks (marked with *) are not included in some further reports. See text for details.

Benchmarks	Vars	Clauses
blasted_case47	118	328
blasted_case110	287	1263
s820a_7_4	616	1703
s820a_15_7	685	1987
s1238a_3_2	685	1850
s1196a_3_2	689	1805
s832a_15_7	693	2017
blasted_case_1_b12_2*	827	2725
blasted_squaring16*	1627	5835
blasted_squaring7*	1628	5837
70.sk_3_40	4669	15864
ProcessBean.sk_8_64	4767	14458
56.sk_6_38	4836	17828
35.sk_3_52	4894	10547
80.sk_2_48	4963	17060
7.sk_4_50	6674	24816
doublyLinkedList.sk_8_37	6889	26918
19.sk_3_48	6984	23867
29.sk_3_45	8857	31557
isolateRightmost.sk_7_481	10024	35275
17.sk_3_45	10081	27056
81.sk_5_51	10764	38006
LoginService2.sk_23_36	11510	41411
sort.sk_8_52	12124	49611
parity.sk_11_11	13115	47506
77.sk_3_44	14524	27573
20.sk_1_51	15465	60994
enqueueSeqSK.sk_10_42	16465	58515
karatsuba.sk_7_41	19593	82417
tutorial3.sk_4_31	486193	2598178

The delta δ of two valid solution a, b can be transferred (or grafted) into another valid solution c.

That is, if we have two valid solutions a and b, then $c \oplus (a \oplus b)$ will likely be valid, where c is another valid solution. In other words, given a set of valid solutions, most of pairwise delta are identical. The basic research question in our exploration is to verify this assumption.

To verify this, for each benchmark, we randomly created 100 valid solutions, $s_1, s_2, \ldots, s_{100}$ from $\mathbb{Z}3$ solver. And then calculate $\delta_{ij} = s_i \oplus s_j$. Given that 100^2 pairs of delta, we report the size of identical delta set.

Figure 8.10 reveals the number of identical deltas among 100^2 pair of deltas between every two

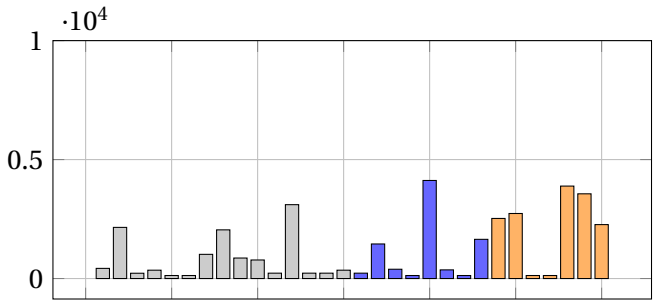


Figure 8.10 Number of identical deltas among 100*100 pair of valid solution deltas for all benchmarks. Color schemes are the same as Table 8.2.

valid solutions. Among all benchmarks, we did not get any identical delta set with size larger than 5000. In 25 benchmarks, we have size of identical delta set less than 2500. That is, in all benchmarks, the majority of deltas were duplicated among the 10000 pairs. The result is exciting, because it approves our assumption, applying the delta of two valid solutions to another valid solution is likely generating a new valid one. This is the fundamental of WORTHY.

In short, we can conclude as follows –

Most likely, the delta of two valid solutions can be transferred to another valid solution.

8.3.7 Can WORTHY Find Test Suite with Enough Diverse Faster?

As a theory proving technique, a basic research focus is about its algorithm efficiency. To illustrate the efficiency of WORTHY, here we report and compare the diversity of output samples over time, instead of the number of them over time, since what software engineers care about is the diversity, instead of the counting of samples.

This experiment adopts the diversity metrics – normalized compression distance (NCD) [Fel16]. Normalized compression distance (NCD) is the diversity metric which is based on information theory. Kolmogorov complexity [LV13] of a binary string x is the length of the shortest program that outputs x. NCD is based on the observation that the degree to which a string can be compressed by real-world compression programs, such as `gzip` or `bzip2`. Therefore, this work chose `gzip` to estimate the Kolmogorov complexity of a binary string. Let $C(x)$ be the length for the compression of x and $C(X)$ be the compression length of binary string set X's concatenation. NCD of X is defined as

$$\mathrm{NCD}(X) = \frac{C(X) - \min_{x \in X}\{C(x)\}}{\mathtt{max}_{x \in X}\{C(X \backslash \{x\})\}} \tag{8.1}$$

Feldt *et al.* showed that test suite with high NCD implies the higher code coverage during the testing. NCD is engineering and practice oriented.

We did not adopt the diversity metric (distribution of samples displayed as histogram) in [Cha15; Dut18] for three reasons:

- the histogram shows the occurrence of some solution patterns, which is not suitable in RQ2

analysis, since the goal of WORTHY is not regarding counting. In fact, WORTHY did not generate millions, or even tens of millions samples as QuickSampler did.

- the histogram did not apply to all benchmarks, especially the large benchmarks. In [Dut18], they only illustrated the results in the benchmarks for which number of samples generated by Unigen2 was at least five time of total number of solutions. Among 30 benchmarks, only five of them were reported.

- the calculation of that metric is time-consuming. It would be approximately 20 CPU days.

In the experiment, the sampling process were terminated whenever the improvement of NCD was less than 5% within 10 minutes. We believe this can be treated as the converge signal of the algorithm. Whether or not they would get better NCD given extension time is left to the future work.

Figure 8.11 plots the NCD comparisons when the WORTHY and QuickSampler got terminated. It shows that in most cases, both of WORTHY and QuickSampler can achieve NCD> 0.9, except for the largest benchmark *tutorial3.sk_4_31*. Comparing the NCD of two algorithms, we can see that in majority cases, measured by NCD, the WORTHY achieved more than 95% of QuickSampler, except for two benchmarks – *s1238_a_3_2* and *parity.sk_11_11*.

Given the observation that WORTHY can get test suite with considerable diverse, the next question is regarding to the sampling time. Figure 8.12 shows the sampling time in the experiment. Please note that we did not report the WORTHY sampling time for *s1238_a_3_2* and *parity.sk_11_11*, since they got much worse results in measuring the diversity. Furthermore, time for building up the initial sets using the $\mathbb{Z}3$ was *not* counted in Figure 8.12 too. The last column in Table 8.2 provides an estimation of initial set construction costs. It is a tradeoff – spend minutes finding initial set before sampling, while avoid hours to figure out the independent support required by the QuickSampler. To get the fair comparison, this work just compares the sampling time, ignoring the cost searching for initial set (in WORTHY) or independent support (in QuickSampler).

Figure 8.12 shows that in majority cases, WORTHY terminated much faster than the QuickSampler algorithm. In some benchmarks, such as the *ProcessBean.sk_8_64.cnf* , *20.sk_1_51.cnf* or *sort.sk_8_52.cnf*, WORTHY is magnitude faster than QuickSampler, given the requirements of diversity. WORTHY were slower than QuickSampler in partial benchmarks, all of which are small or medium benchmarks. In such benchmarks, WORTHY could still terminate within dozens of minutes. However, in the large benchmarks, WORTHY terminated faster, saving the software engineers lots of time.

Based on the aforementioned analysis, we can answer the research question as –
In majority cases, WORTHY can find test suite with similar diverse as QuickSampler, AND in a much shorter (affordable) time.

8.3.8 Can WORTHY Reduce the Size of Test Suite?

Another question software engineers interested in is how large the test suites are. The test suite should be smaller, under the premise that the test suite should provide enough diversity. The most

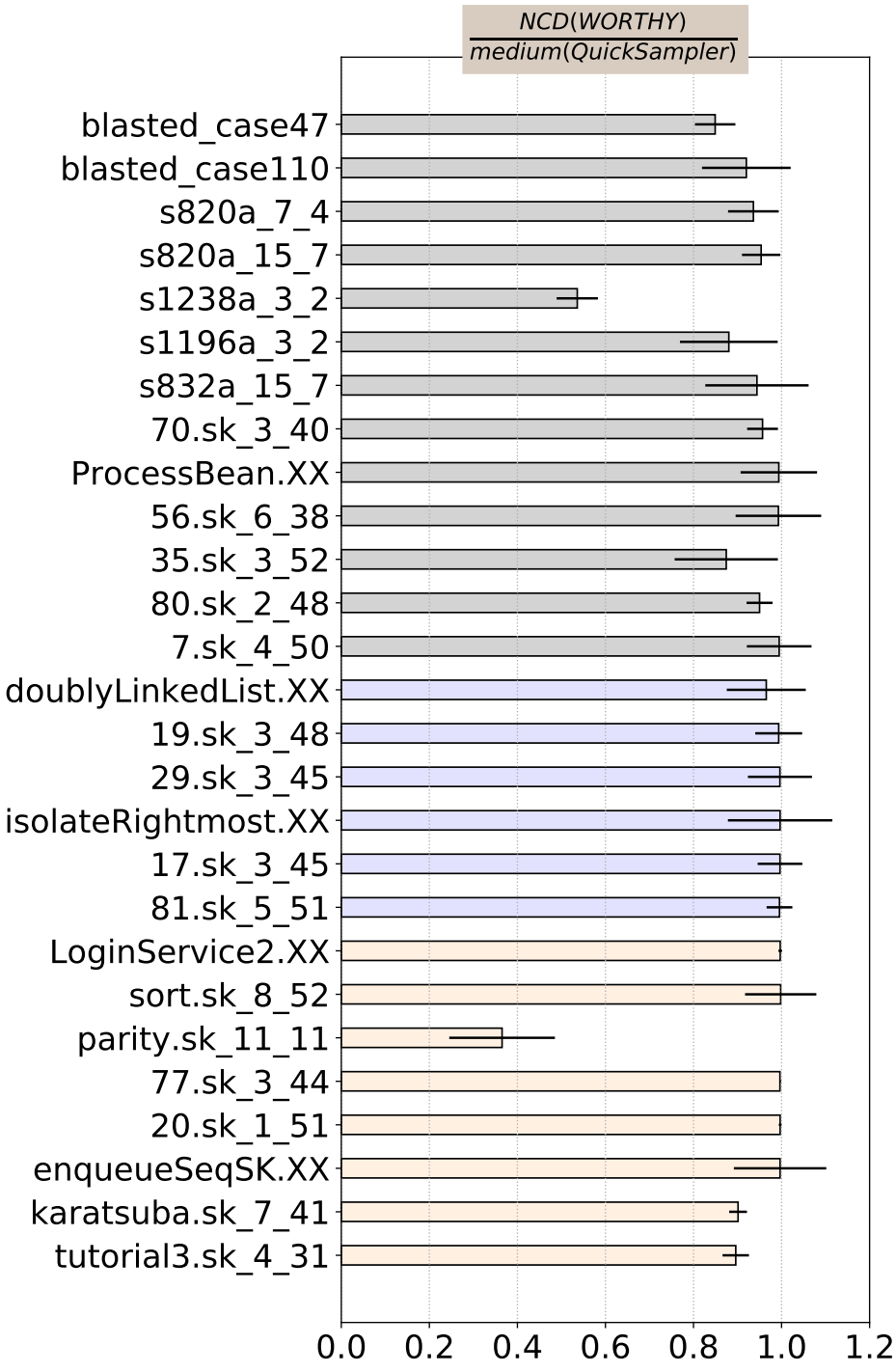


Figure 8.11 Normalized compression distance (NCD) got when the experiment terminated. NCD is a diversity metric. The larger NCD the better. WORTHY and QuickSampler were terminated whenever the NCD got improved by less than 5% within 10 minutes window. The number above the grouped bars are $\frac{\text{NCD(WORTHY)}}{\text{NCD(QuickSampler)}}$ (in percentage). In majority cases, WORTHY terminated at the NCD almost the same as QuickSampler.

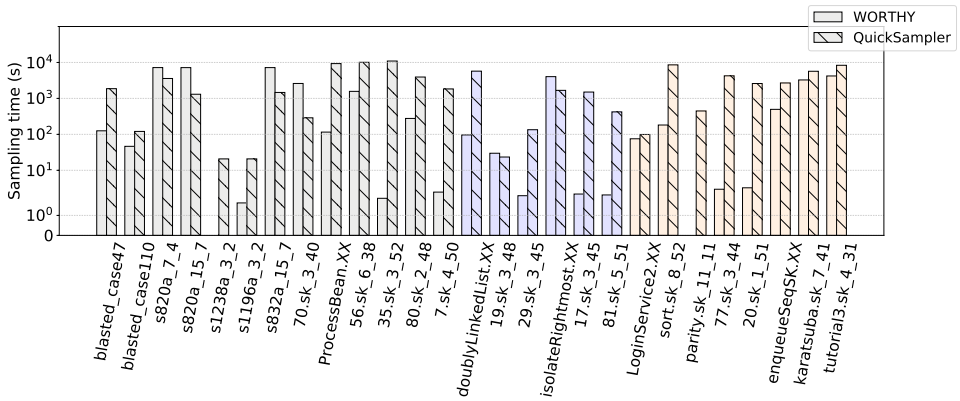


Figure 8.12 Sampling time required before execution got terminated. The y-axis is in log scale. The WORTHY sampling time for *s1238_a_3_2* and *parity.sk_11_11* are not reported since their achieved NCD were much worse than QuickSampler's (see Figure 8.11). The *speedup* > 1 implies WORTHY terminates earlier than QuickSampler.

important motivation for WORTHY is to search for a test suite which is small and devised enough.

Table 8.3 illustrates the number of unique cases in the test suites when the algorithms terminated. All benchmarks are sorted and grouped by their complexity (number of variables), same as Table 8.2. The second column, or S_W, are the size of test suite given by WORTHY, while the numbers in third column, i.e. S_Q, are given by QuickSampler. The final column is the ratio between S_Q and S_W. Benchmarks *s1238_a_3_2* and *parity.sk_11_11* are not reported due to the poor performance of WORTHY in these benchmarks.

From Table 8.3, we have

- S_Q is 91x (in average), 14x (in medium) of S_W. That is, sharing the similar diverse, compared to QuickSampler's, running the test suites from WORTHY can save > 90% CPU times assuming the testing time is proportional to test suite size.
- In the largest models, the test suite size from WORTHY is always magnitude smaller.
- Except the extremely simple benchmark, there are two more benchmarks having $S_Q/S_W < 1$. However, S_Q and S_W were in the same magnitude among them; that is, result of WORTHY is slightly worse than QuickSampler when it is defeated.

Therefore, we can summarize the observations as

WORTHY searches for the test suite with much less cases than QuickSampler, reducing the loads in testing significantly.

8.3.9 Threats to Validity

First threat to validity is the *baseline bias*. Indeed, there are many other sampling techniques, or solvers, to the theory proving problem that WORTHY can compare to. However, goal of WORTHY is not searching for as many test cases as possible. Conversely, WORTHY need to search for the

Table 8.3 Number of unique cases in the test suite. Benchmarks are sorted by number of variables. Medium benchmarks are highlighted with blue rows while the large ones are in orange rows.

Benchmarks	WORTHY S_W	QuickSampler S_Q	$\frac{S_Q}{S_W}$
blasted_case47	2799	71	0.00
blasted_case110	174	2386	13.71
s820a_7_4	37363	124457	3.30
s820a_15_7	28965	70099	2.40
s1196a_3_2	125	1890	15.12
s832a_15_7	27440	96764	3.50
70.sk_3_40	2950	4270	1.40
ProcessBean.sk_8_64	1093	75392	68.97
56.sk_6_38	1727	149031	86.29
35.sk_3_52	158	193920	1227.34
80.sk_2_48	553	54440	98.44
7.sk_4_50	158	18090	114.49
doublyLinkedList.sk_8_37	178	12042	67.65
19.sk_3_48	104	200	1.90
29.sk_3_45	125	660	5.28
isolateRightmost.sk_7_481	15380	7510	0.49
17.sk_3_45	128	12780	99.84
81.sk_5_51	127	2814	22.18
LoginService2.sk_23_36	304	210	0.69
sort.sk_8_52	712	10184	14.30
77.sk_3_44	145	33858	233.50
20.sk_1_51	139	10039	72.22
enqueueSeqSK.sk_10_42	238	2495	10.48
karatsuba.sk_7_41	39	4210	107.94
tutorial3.sk_4_31	236	2953	12.51

test suites with less cases. For that goal, it is not necessary to compare WORTHY against all other techniques. Rather, WORTHY should be compared to known state-of-the-art in the literature.

Second threat is the *internal bias*. The internal bias raises from the stochastic nature of sampling techniques. WORTHY requires many random operations. To mitigate the threats, we repeated the experiments for 20 times and reported the medium result of them.

Third threat is the *measurement bias*. To determine the diversity of a test suite, in the experiment, we use normalized compression distance (NCD). Effectiveness of NCD was approved [Fel16]. However, there exists many other diversity measurements for the theory proving problem. Changing the diversity measurement might lead to change the results. It is impossible to adopt all of them in one experiment. For the convenient of further exploration, we released the source code of WORTHY. One can test that under any other diversity metrics.

Another threat is the *hyperparameter bias*. The hyperparameter is the set of configurations for the algorithm. In WORTHY, we need to set the size of initial samples, mutation tree minimum leaves size as well as the termination criteria. Grid search is a common technique to find the best hyperparamter. Fu *et al.* proposed an agile hyperparamter tuning technique based on differential evolution [FM17b]. WORTHY may performance better via hyperparameter tuning. It is left for the future work.

8.4 Summary of OSAP4

The fourth generation of OSAP created a more flexible surrogate model without the dependence of linearity of the software engineering models. OSAP4 has the Δ-oriented surrogate model, which use the Δ_D and Δ_O as the independent and dependent variables. From the surrogate model, it is not the exact objective values that are evaluated, but the relatives of them. As long as we know whether configure p has better objective than configure q, we can do the selection and pruning. Therefore, this method does not require the linearity of the model.

Also, the number of initial sample size is smaller in this model. For some highly constraint models, this is extremely helpful, since it is not easy to get a large set of valid initial samples.

CHAPTER

9

CONCLUSIONS AND FUTURE WORK

9.1 Executive Summary

This thesis explored four generation of OSAP. This section gives some executive conclusion to enable researchers or engineers to use that. Please note that every generation of OSAP has its own advantage and disadvantage. It is not possible to find an algorithm to fit all problems.

First of all, OSAP1 should be the fastest method. We should always try that as the baseline methodology. Expert or domain knowledge is required in OSAP2. If we have such knowledge, OSAP2 is a good starter.

The latest version OSAP4 was designed to remove the limitations of the old versions. The OSAP4 is extremely helpful when it is difficult to get large set of initial samples, such as in the situation that the model is highly constrained. The OSAP4 requires more model evaluations than the previous versions. This is one drawback of that. However, fortunately, it is still faster than the standard EVOL.

9.2 Generations of OSAP Revisited

This work explored four version of OSAP. The OSAP, short for over-sampling-and-pruning, is the framework to optimize the search-based software engineering problems. In this framework, samples are not to be improved via crossover, mutation, etc. like the standard evolutionary algorithm (EVOL); instead, all configurations are generated and selected from a huge set of samples.

Thesis Statement: for the optimization of search based software engineering problems, given a proper configuration selector or comparator built upon decision space, OSAP is better than a standard mutation based evolutionary approach (EVOL); where "better" is measured in terms of

runtimes, number of evaluations and value of final results.

By saying "built upon decision space", it meant that all operators (sampling and pruning) are executed in the decision space. This was important – if that operators was built upon the objective space, then all samples have to be evaluated so that they can be mapped into the objective space. Also, please note that it was still necessary for the OSAP to acquire model evaluations. However, such model evaluations was not a common case and all evaluations were well-designed which can be utilized to "plot" the software engineering models.

The first generation OSAP1, also known as SWAY, is looking for a "golden" region in the decision space. It assumes that (all / most of) the optimal configurations (pareto frontier) are located only in such a small region. To get such region, SWAY uses a simple cluster named WHERE to group the samples into two and then prune one of them by evaluating and comparing the representative in each group. After several recursive clustering and pruning, there left a small region. Regardless of the effectiveness of WHERE, the "golden" region assumption may be not valid for the software engineer models.

The second generation OSAP2, also known as SWAY^2, is designed to fix aforementioned limitation in some extent. It looks for multiple "golden" regions. To support the searching process, SWAY^2 requires the expert or domain knowledge to do the "primary" splitting on the decision space. After the splitting, the decision space is divided into multiple sub-spaces. The SWAY is called to find the "golden" region for each sub-spaces. §6 have some guidance for the primary splitting. Two major limitations here: 1) it is not 100% automated – the first step requires the expert or domain knowledge; 2) the "golden" region assumption still needs to be true.

To release the "golden" region assumption, the OSAP has to find some way to directly compare any configurations ans then select the optimal ones. The surrogate model helps here. The surrogate model is an alternative model to the original SE problem which can quickly compare the configurations. In OSAP3, or the RIOT, the surrogate model is designed to get the objectives of a configuration. The surrogate model in RIOT is built upon the linearity of the SE model and we have shown the effectiveness in resource planning related problems, e.g. the workflow deployments in the cloud environment. The success of OSAP3 is highly relied on the linearity of the model.

For the model which does not the (strong-) linearity components, the fourth generation of OSAP proposes a novel surrogate model. OSAP4 points out that it is unnecessary to fetch the exactly value of the objectives. Instead, what really matters is the objective deltas. The OSAP4 introduces the Δ-oriented surrogate model. To build such model, we only need set of Δ_D and their corresponding Δ_O. Note that if we have N evaluated samples, we therefore have O(N^2) <Δ_D, Δ_O>. In other words, OSAP4 requires less initial sample sets. This is very helpful in the highly constrainted problems.

9.3 Study Cases Revisited

In this work we explored five study cases, including

- XOMO, POM3

- Software product line optimization (SPL)
- Next release planning problem (NRP)
- Workflow deployment in cloud environment (WORKFLOW)
- Test suite generation (TEST)

This section discuses the diversity of this studies. It is worthy to mention that it is impossible to explore all SE problems, or all types of SE models in a single work. More exploration is left to the future work. To support other researchers, all algorithms or models are open-sourced *(see corresponding papers for details)*. Anyone feels interested in the work can modify the OSAP or use that as the baseline.

Classified by the SE procedure, we have

- XOMO, POM3 is for the software management
- Software product line optimization and next release planning is for the requirement engineering
- Workflow deployment is for the deployment process
- Finally, the test suite generation is for the testing

Classified by the decision space, we have

- XOMO and POM3 has numeric decisions
- NRP and WORKFLOW has enumerate decisions
- SPL and TEST has the binary decisions.

Classified by the extra constraints, we have

- XOMO, POM3 or WORKFLOW are simple, i.e. no specific constraints enforced in the model
- NRP, WORKFLOW is some constraints (defined by the valid decision enumerations)
- SPL and TEST and highly constrainted model. They are NP-Complete problems.

9.4 Other Works Related to this Research

9.4.1 Better Evolutionary Algorithms for SBSE

Most of previous research in SBSE community is focusing on how to make evolutionary algorithms more adaptive to the specific SE problems or oracles. Such adaption includes

- Better problem encoding. For example,

  - Amarjeet et al. [Chh17] purposed a Harmony Search Based Remodularization Algorithm (HSBRA) to solve the software remodularization problem for object-oriented software systems. They made their problem encoding better adaptive to software modularization solution. In HSBRA, they presented a vector based encoding, where a modularization configuration with the same value for all the classes means that all classes are placed in the same module, while a solution with all possible values (from 1 to n) means that each module is composed of one class only. These effective problem encoding guarantees that they can represent all valid configurations for this problem.
  - Kessentini et al. [Kes17] applied the NSGA-II algorithm to detect the model level changes in software development. They took as input an exhaustive list of possible types of model refactoring operations, the initial model, and the revised model. All of these inputs were integrated into an unique encoding scheme.

- Combining multiple evolutionary algorithms. For example,

  - Idri et al. [Idr15] investigated the analogy-based software development effort estimation (ASEE) techniques. Basing on the finding of their study the use of techniques such as Fuzzy logic, genetic algorithm, etc. in combination with an ASSE method was promising to generate more accurate estimates.
  - Mahdavi et al. [Mah05] showed that combing the results from multiple hill climbs and/or genetic algorithms can improve on the results for simple HC and GA.

- Better objective(fitness) function. For example,

  - Huang et al. [Hua17] proposed differential evolution based on self-adaptive fitness function for automated test case generation. In their method, they address the self-adaptive fitness function so that it can be better reflecting one configuration limitations.

All these efforts are to find better evolutionary algorithms for the specific SE problem. In this research we proposed the general framework to optimize the SBSE problems.

9.4.2 Faster Evolutionary Algorithms

Researchers in other communities make efforts to make evolutionary algorithms terminate sooner. For example, Shahrzad et al. [Sha16] analyzed the advantage of age-layering in an evolutionary algorithm as well. In aged-layered evolutionary algorithms, a small sample of candidates are evaluated first; and if they seem promising, they are evaluated with more samples. The age-layering method effectively reduces the fitness evaluations and speedups the evolution process. However, at least for the aged-layered algorithm reported by Shahrzad et al. [Sha16], this approach still requires millions of model evaluations.

(1+1) EA is another strategy which can reduce the computing intensity of evolution algorithms [Dro02]. In (1+1) EA, the population size is set to one. The candidate is mutated in some

probability and then replaces the former one if better fitness is found. Compared to the common evolution algorithms which population size can up to hundreds or even thousands, the (1+1) EA can significantly reduce the fitness evaluations [Dro02]. But the drawback for standard (1+1) EA is that it did not naturally handle models with multi-objectives, or conflicting objectives, which are very common in SBSE.

Another strategy to speed up the evolutionary algorithms is the use of a surrogate model. Ong et al. [Ong03] presented a parallel evolutionary algorithm which leverages surrogate model for solving the computational expensive design problems. A surrogate model is a statistical model built to approximate the computationally expensive model. They created a surrogate model basing on the radieddal basis function. The computation of RBF is much cheaper than the original model. But the precise of surrogate model strongly depends on the evaluated candidates. To improve the precision of surrogate model for search-based software engineering problems, we have to enlarge the number of model evaluations [Sha16].

9.4.3 Solving Problems via Sampling

Sampling has been successively applied to the noisy real-word optimization problems by Cantu-Paz [CP04]. They introduced an adaptive sampling policy which they test on a 100-bit onemax function. While Cantu-Paz demonstrated that the adaptive sampling could find better solutions, from our perspective, the drawback with that work is that it requires far more computation time.

Zhang et al. [YJ16] proposed a novel hybrid sampling-based clustering methods. Their method was evaluated on a 2-D-synthetic data, collection of benchmarks and real-world facial recognition data sets. Experiments showed that their method can identify the complex cluster structure of the data set, including imbalanced clusters, manifold clusters and clusters with various cluster densities.

In area of Large scale genome-wide association studies, Niel et al. [Nie18] combined both stochastic strategy inspired from random forests and Bayesian Markov blanket-based methods. Their approach performed similar to previous benchmarks in that area, but runs faster.

All these research utilize the sampling to solve various problems. They approve that sampling is valuable. This research systematically introduced the sampling framework for search-based software engineering, proving again the value of sampling.

9.5 Future Work

9.5.1 Ensemble Learning

In machine learning, the ensemble learning use multiple learning algorithm to obtain better predictive performance (compared to single learning algorithm). In the latest version of OSAP, we have a KNN surrogate model. Is the KNN model the best choice for all problems? Can we apply the ensemble learning methods to get the better predictions for specific software engineering models? This is left for the future work.

9.5.2 Better Exploring the Constrained Model

In the software product line optimization and test suite generation study cases, we use the external constraint solver, i.e. the Z3 or PocaSAT, to find the samples that meet the constraints, or fix the invalid samples. However, the Z3 is still very slow, for those problems are NP-Complete problems. How to better explore the constraint model is left to the future work. Apparently we need to utilize such professional tools. But can we call these tools less? Also, for some constraints, the professional constraint solver is in lack. We need to find some way to reduce the problem complexity. For example, we can get some sampling strategy to divide the problem/constraints into several smaller problem and then conquer them.

9.5.3 Increasing Sampling for Modified Models

In many times we have to find the configuration for modified models. For example, in the cloud deployment study cases, the workflow task may be expanded; therefore, we have to adjust the deployment of the workflows. Regression testing is another example for the modified models. Up to now the OSAP does not explore such kind of modified models. How to handle such situations using the *increasing sampling* is left for the future work.

BIBLIOGRAPHY

[Abr13] Abrishami, S. et al. "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds". *Future Generation Computer Systems* **29**.1 (2013), pp. 158–169.

[Ake78] Akers, S. B. "Binary decision diagrams". *IEEE Transactions on computers* 6 (1978), pp. 509–516.

[AF13] Arcuri, A. & Fraser, G. "Parameter tuning or default values? An empirical investigation in search-based software engineering". *Empirical Software Engineering* **18**.3 (2013), pp. 594–623.

[Ard14] Ardagna, D. et al. "A multi-model optimization framework for the model driven design of cloud applications". *International Symposium on Search Based Software Engineering*. Springer. 2014, pp. 61–76.

[AC17] Arias-Castro, E. et al. "Spectral clustering based on local PCA". *Journal of Machine Learning Research* **18**.9 (2017), pp. 1–57.

[BZ11] Bader, J. & Zitzler, E. "HypE: An algorithm for fast hypervolume-based many-objective optimization". *Evolutionary computation* **19**.1 (2011), pp. 45–76.

[Bal18] Baldoni, R. et al. "A survey of symbolic execution techniques". *ACM Computing Surveys (CSUR)* **51**.3 (2018), p. 50.

[Ban98] Banzhaf, W. et al. *Genetic programming: an introduction*. Morgan Kaufmann Publishers San Francisco, 1998.

[BT18] Barrett, C. & Tinelli, C. "Satisfiability modulo theories". *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[BD16] Bellet, A. & Denis, P. "Spectral Graph-Based Methods for Learning Word Embeddings" (2016).

[Ber12] Berger, T. et al. "Variability modeling in the systems software domain". *Generative Software Development Laboratory, University of Waterloo, Technical Report* (2012).

[BB12] Bergstra, J. & Bengio, Y. "Random search for hyper-parameter optimization". *Journal of Machine Learning Research* **13**.Feb (2012), pp. 281–305.

[BB17] Bishop, A. B. & Bitanski, N. "DIMACS Workshop on Complexity of Cryptographic Primitives and Assumptions". *City* (2017).

[BM11] Bittencourt, L. F. & Madeira, E. R. M. "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds". *Journal of Internet Services and Applications* **2**.3 (2011), pp. 207–227.

[BB13] Bochkanov, S. & Bystritsky, V. "Alglib". *Available from: www. alglib. net* **59** (2013).

[BT03a] Boehm, B. & Turner, R. “Using risk to balance agile and plan-driven methods”. *Computer* **36**.6 (2003), pp. 57–66.

[BT03b] Boehm, B. & Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[Boe00] Boehm, B. et al. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[Bro08] Brown, R. E. et al. “Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431” (2008).

[CP15] Calero, C. & Piattini, M. *Green in Software Engineering*. 2015.

[CP04] Cantú-Paz, E. “Adaptive sampling for noisy problems”. *Genetic and Evolutionary Computation Conference*. Springer. 2004, pp. 947–958.

[Cha14] Chakraborty, S. et al. “Balancing scalability and uniformity in SAT witness generator”. *Proceedings of the 51st Annual Design Automation Conference*. ACM. 2014, pp. 1–6.

[Cha15] Chakraborty, S. et al. “On parallel scalable uniform SAT witness generation”. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2015, pp. 304–319.

[CM18] Chen, J. & Menzies, T. “RIOT: A Stochastic-Based Method for Workflow Scheduling in the Cloud”. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE. 2018, pp. 318–325.

[Che17] Chen, J. et al. “Beyond evolutionary algorithms for search-based software engineering”. *Information and Software Technology* (2017).

[Che18] Chen, J. et al. “" Sampling" as a Baseline Optimizer for Search-based Software Engineering”. *IEEE Transactions on Software Engineering* (2018).

[CZ09] Chen, W.-N. & Zhang, J. “An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements”. *IEEE Trans. on Systems, Man, and Cybernetics* **39**.1 (2009), pp. 29–43.

[Chh17] Chhabra, J. K. et al. “Harmony search based remodularization for object-oriented software systems”. *Computer Languages, Systems & Structures* **47** (2017), pp. 153–169.

[Chr16] Christakis, M. et al. “Guiding dynamic symbolic execution toward unverified program executions”. *Proceedings of the 38th International Conference on Software Engineering*. ACM. 2016, pp. 144–155.

[Chu03] Chung, K.-M. et al. “Radius margin bounds for support vector machines with the RBF kernel”. *Neural computation* **15**.11 (2003), pp. 2643–2681.

[Coh95] Cohen, P. R. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[Cor09] Cormen, T. H. *Introduction to algorithms*. MIT press, 2009.

[Deb00a] Deb, K. et al. “A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II”. *IEEE Transactions on Evolutionary Computation* (2000).

[Deb00b] Deb, K. et al. “A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II”. *International Conference on Parallel Problem Solving From Nature*. Springer. 2000, pp. 849–858.

[Deb01] Deb, K. et al. *Scalable Test Problems for Evolutionary Multi-Objective Optimization*. TIK Report 1990. Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2001, pp. 1–27.

[Deb02] Deb, K. et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. *IEEE Trans. on evolutionary computation* **6**.2 (2002), pp. 182–197.

[DC08] Deelman, E. & Chervenak, A. “Data management challenges of data-intensive scientific workflows”. *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE Intl. Sym. on*. IEEE. 2008, pp. 687–692.

[Dom12] Domingos, P. “A few useful things to know about machine learning”. *Communications of the ACM* **55**.10 (2012), pp. 78–87.

[Dor06] Dorigo, M. et al. “Ant colony optimization”. *IEEE computational intelligence magazine* **1**.4 (2006), pp. 28–39.

[Dro02] Droste, S. et al. “On the analysis of the (1+ 1) evolutionary algorithm”. *Theoretical Computer Science* **276**.1-2 (2002), pp. 51–81.

[DP14] Durillo, J. J. & Prodan, R. “Multi-objective workflow scheduling in Amazon EC2”. *Cluster computing* **17**.2 (2014), pp. 169–189.

[Dur09] Durillo, J. J. et al. “A study of the multi-objective next release problem”. *Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSEâĂŹ09)*. 2009, pp. 49–58.

[Dut18] Dutra, R. et al. “Efficient sampling of SAT solutions for testing”. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE. 2018, pp. 549–559.

[ET93] Efron, B. & Tibshirani, R. J. *An introduction to the bootstrap*. CRC, 1993.

[Erm12] Ermon, S. et al. “Uniform solution sampling using a constraint solver as an oracle”. *arXiv preprint arXiv:1210.4861* (2012).

[FL95] Faloutsos, C. & Lin, K.-I. “FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets”. *ACM SIGMOD international conference on Management of data*. San Jose, California, United States, 1995.

[Far12] Fard, H. M. et al. “A multi-objective approach for workflow scheduling in heterogeneous environments”. *Proceedings of the 2012 12th IEEE/ACM Intl. Sym. on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society. 2012, pp. 300–309.

[Fel16] Feldt, R. et al. “Test set diameter: Quantifying the diversity of sets of test cases”. *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2016, pp. 223–233.

[Fer13] Ferrucci, F. et al. “Not going to take this anymore: multi-objective overtime planning for software engineering projects”. *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 462–471.

[Fin15] Finke, M. “Equisatisfiable SAT Encodings of Arithmetical Operations”. *Online] http://www. martin-finke. de/documents/Masterarbeit_bitblast_ Finke. pdf* (2015).

[Fis12] Fisher, D. et al. “Interactions with Big Data Analytics”. *interactions* **19**.3 (2012), pp. 50–59.

[For12] Fortin, F.-A. et al. “DEAP: Evolutionary Algorithms Made Easy”. *Journal of Machine Learning Research* **13** (2012), pp. 2171–2175.

[FM17a] Fu, W. & Menzies, T. “Easy over Hard: A Case Study on Deep Learning”. *arXiv preprint arXiv:1703.00133* (2017).

[FM17b] Fu, W. & Menzies, T. “Revisiting unsupervised learning for defect prediction”. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM. 2017, pp. 72–83.

[Fu16] Fu, W. et al. “Tuning for software analytics: Is it really necessary?” *Information and Software Technology* **76** (2016), pp. 135–146.

[Gho15] Ghotra, B. et al. “Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models”. *37th IEEE International Conference on Software Engineering*. 2015.

[GD11] Gogate, V. & Dechter, R. “SampleSearch: Importance sampling in presence of determinism”. *Artificial Intelligence* **175**.2 (2011), pp. 694–729.

[Har13] Harman, M. *Personal communication*. 2013.

[Har12] Harman, M. et al. “Search Based Software Engineering: Techniques, Taxonomy, Tutorial”. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **7007** (2012), pp. 1–59.

[Har14a] Harman, M. et al. “Search based software engineering for software product line engineering: a survey and directions for future work”. *Proceedings of the 18th International Software Product Line Conference-Volume 1*. ACM. 2014, pp. 5–18.

[Har14b] Harman, M. et al. “Search based software engineering for software product line engineering: a survey and directions for future work”. *18th International Software Product Line Conference, SPLC ’14, Florence, Italy, September 15-19, 2014*. 2014, pp. 5–18.

[Hen15] Henard, C. et al. “Combining multi-objective search and constraint solving for configuring large software product lines”. *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*. Vol. 1. IEEE. 2015, pp. 517–528.

[Hol92] Holland, J. H. “Genetic Algorithms”. *Scientific American* **267**.1 (1992), pp. 66–72.

[Hol93] Holte, R. C. “Very Simple Classification Rules Perform Well on Most Commonly Used Datasets”. *Machine Learning* **11** (1993), p. 63.

[Hua17] Huang, H. et al. “Differential Evolution Based on Self-Adaptive Fitness Function for Automated Test Case Generation”. *IEEE Computational Intelligence Magazine* **12**.2 (2017), pp. 46–55.

[Idr15] Idri, A. et al. “Analogy-based software development effort estimation: A systematic mapping and review”. *Information and Software Technology* **58** (2015), pp. 206–230.

[Ios11] Iosup, A. et al. “On the performance variability of production cloud services”. *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM Intl. Sym. on*. IEEE. 2011, pp. 104–113.

[Iye03] Iyer, M. A. “RACE: A word-level ATPG-based constraints solver system for smart random simulation”. *null*. IEEE. 2003, p. 299.

[Jas02] Jaszkiewicz, A. “On the performance of multiple-objective genetic local search on the 0/1 knapsack problem-a comparative experiment”. *IEEE Transactions on Evolutionary Computation* **6**.4 (2002), pp. 402–412.

[Jia15] Jia, X. et al. “Enhancing reuse of constraint solutions to improve symbolic execution”. *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM. 2015, pp. 177–187.

[Kam03] Kamvar, S. D. et al. “Spectral Learning”. *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI’03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 561–566.

[Kes17] Kessentini, M. et al. “Search-based detection of model level changes”. *Empirical Software Engineering* **22**.2 (2017), pp. 670–715.

[Kim07] Kim, J.-K. et al. “Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment”. *Journal of parallel and distributed computing* **67**.2 (2007), pp. 154–169.

[Kit07] Kitchenham, B. a. et al. “Cross- vs. Within-Company Cost Estimation Studies: A Systematic Review”. *IEEE Transactions on Software Engineering* **33**.5 (2007), pp. 316–329.

[KC99] Knowles, J. & Corne, D. “The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation”. *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 1. IEEE. 1999.

[Kra15a] Krall, J. et al. “GALE: Geometric Active Learning for Search-Based Software Engineering”. *IEEE Transactions on Software Engineering* **41**.10 (2015), pp. 1001–1018.

[Kra16] Krall, J. et al. “Learning Mitigations for Pilot Issues When Landing Aircraft (via Multiobjective Optimization and Multiagent Simulations)”. *IEEE Transactions on Human-Machine Systems* **46**.2 (2016), pp. 221–230.

[Kra14] Krall, J. et al. “Learning the Task Management Space of an Aircraft Approach Model”. *Proceedings of the 2014 AAAI Conference*. AAAI’14. 2014.

[Kra15b] Krall, J. et al. “Gale: Geometric active learning for search-based software engineering”. *IEEE Trans. on Software Engineering* **41**.10 (2015), pp. 1001–1018.

[Kum03] Kumar, R. et al. “Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction”. *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 36. 2003, pp. 81–.

[Lau06] Lau, S. Q. *Domain analysis of e-commerce systems using feature-based model templates*. University of Waterloo Ontario, Canada, 2006.

[Lek14] Lekkalapudi, N. K. *Cross trees: Visualizing estimations using decision trees*. West Virginia University, 2014.

[Li16] Li, H. et al. “Randomized algorithms for distributed computation of principal component analysis and singular value decomposition”. *arXiv preprint arXiv:1612.08709* (2016).

[LV13] Li, M. & Vitányi, P. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2013.

[Mah05] Mahdavi, K. “A clustering genetic algorithm for software modularisation with a multiple hill climbing approach”. PhD thesis. Brunel University, 2005.

[Mal15] Malawski, M. et al. “Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds”. *Future Generation Computer Systems* **48** (2015), pp. 1–18.

[Man93] Mansour, Y. et al. “The computational complexity of universal hashing”. *Theoretical Computer Science* **107**.1 (1993), pp. 121–133.

[MA04] Marler, R. T. & Arora, J. S. “Survey of multi-objective optimization methods for engineering”. *Structural and multidisciplinary optimization* **26**.6 (2004), pp. 369–395.

[Mee16] Meel, K. S. et al. “Constrained sampling and counting: Universal hashing meets SAT solving”. *Workshops at the thirtieth AAAI conference on artificial intelligence*. 2016.

[Men08] Mendonça, M. et al. “Decision-making coordination in collaborative product configuration”. *Proceedings of the 2008 ACM symposium on Applied computing*. ACM. 2008, pp. 108–113.

[MC97] Menzies, T. & Compton, P. “Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models”. *Artificial intelligence in medicine* **10**.2 (1997), pp. 145–175.

[MC00] Menzies, T. & Cukic, B. “When to test less”. *IEEE Software* **17**.5 (2000), pp. 107–112.

[MR05] Menzies, T. & Richardson, J. “Xomo: Understanding development options for autonomy”. *COCOMO forum*. Vol. 2005. 2005.

[Men02] Menzies, T. et al. “Applications of abduction: Testing very long qualitative simulations”. *IEEE Transactions on Knowledge and Data Engineering* **14**.6 (2002), pp. 1362–1375.

[Men07] Menzies, T. et al. “The Business Case for Automated Software Engineerng”. *22nd IEEE/ACM International Conference on Automated Software Engineering*. Atlanta, Georgia, USA, 2007.

[Men09a] Menzies, T. et al. “Accurate Estimates Without Local Data?” *Software Process Improvement and Practice* (4 2009).

[Men09b] Menzies, T. et al. “How to Avoid Drastic Software Process Change (using Stochastic Stability)”. *31st International Conference on Software Engineering*. 2009.

[MA13] Mittas, N. & Angelis, L. “Ranking and clustering software cost estimation models through a multiple comparisons algorithm”. *IEEE Transactions on software engineering* **39**.4 (2013), pp. 537–551.

[Mon13] Montanez, G. D. “Bounding the number of favorable functions in stochastic search”. *2013 IEEE Congress on Evolutionary Computation*. IEEE. 2013, pp. 3019–3026.

[Moo98] Moore, G. E. “Cramming More Components Onto Integrated Circuits”. *Proceedings of the IEEE* **86**.1 (1998), pp. 82–85.

[Mye11] Myers, G. J. et al. *The art of software testing*. John Wiley & Sons, 2011.

[Nai16] Nair, V. et al. “An (accidental) exploration of alternatives to evolutionary algorithms for sbse”. *International Symposium on Search Based Software Engineering*. Springer. 2016, pp. 96–111.

[Nai18] Nair, V. et al. “Data-Driven Search-based Software Engineering”. *arXiv preprint arXiv: 1801.10241* (2018).

[Neb09] Nebro, A. et al. “SMPSO: A new PSO-based metaheuristic for multi-objective optimization”. *Computational intelligence in miulti-criteria decision-making, 2009. mcdm ’09. ieee symposium on*. 2009, pp. 66–73.

[Nie18] Niel, C. et al. “SMMB–A stochastic Markov-blanket framework strategy for epistasis detection in GWAS”. *Bioinformatics* (2018).

[Ong03] Ong, Y. S. et al. “Evolutionary optimization of computationally expensive problems via surrogate modeling”. *AIAA journal* **41**.4 (2003), pp. 687–696.

[Oun17] Ouni, A. et al. “Search-based software library recommendation using multi-objective optimization”. *Information and Software Technology* **83** (2017), pp. 55–75.

[Owe02a] Owen, D. et al. “An alternative to model checking: Verification by random search of AND-OR graphs representing finite-state models”. *High Assurance Systems Engineering, 2002. Proceedings. 7th IEEE International Symposium on*. IEEE. 2002, pp. 119–126.

[Owe02b] Owen, D. et al. “What makes finite-state models more (or less) testable?” *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*. IEEE. 2002, pp. 237–240.

[Pan10] Pandey, S. et al. “A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments”. *Advanced information networking and applications (AINA), 2010 24th IEEE Intl. conference on*. IEEE. 2010, pp. 400–407.

[Pla05] Platt, J. C. “Fastmap, metricmap, and landmark MDS are all nystrom algorithms”. *In Proceedings of 10th International Workshop on Artificial Intelligence and Statistics*. 2005, pp. 261–268.

[Por08] Port, D. et al. “Using simulation to investigate requirements prioritization strategies”. *ASE 2008 - 23rd IEEE/ACM International Conference on Automated Software Engineering, Proceedings*. 2008, pp. 268–277.

[Red16] Redgate, S. et al. “Principal component analysis as a method of respiratory motion detection on a solid state CZT dedicated cardiac camera”. *Journal of Nuclear Medicine* **57**.supplement 2 (2016), pp. 1969–1969.

[Ree07] Reed, P. et al. “Using interactive archives in evolutionary multiobjective optimization: A case study for long-term groundwater monitoring design”. *Environmental Modelling & Software* (2007).

[RS10] Rehman, M. S. & Sakr, M. F. “Initial findings for provisioning variation in cloud computing”. *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second Intl. Conference on*. IEEE. 2010, pp. 473–479.

[RB14] Rodriguez, M. A. & Buyya, R. “Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds”. *IEEE Trans. on Cloud Computing* **2**.2 (2014), pp. 222–235.

[RB17] Rodriguez, M. A. & Buyya, R. “A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments”. *Concurrency and Computation: Practice and Experience* **29**.8 (2017).

[Say13a] Sayyad, A. et al. “Scalable Product Line Configuration: A Straw to Break the Camel’s Back”. *ASE’13, Palo Alto, CA*. 2013.

[Say13b] Sayyad, A. S. et al. “On the value of user preferences in search-based software engineering: a case study in software product lines”. *Software engineering (ICSE), 2013 35th international conference on*. IEEE. 2013, pp. 492–501.

[Say13c] Sayyad, A. S. et al. “On the Value of User Preferences in Search-based Software Engineering: A Case Study in Software Product Lines”. *Proceedings of the 2013 International Conference on Software Engineering*. ICSE ’13. 2013, pp. 492–501.

[Sch10] Schad, J. et al. “Runtime measurements in the cloud: observing, analyzing, and reducing variance”. *Proceedings of the VLDB Endowment* **3**.1-2 (2010), pp. 460–471.

[Sel93] Selman, B. et al. “Local search strategies for satisfiability testing.” *Cliques, coloring, and satisfiability* **26** (1993), pp. 521–532.

[Sha16] Shahrzad, H. et al. “Estimating the advantage of age-layering in evolutionary algorithms”. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM. 2016, pp. 693–699.

[She11] She, S. et al. “Reverse engineering feature models”. *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE. 2011, pp. 461–470.

[SM12] Shepperd, M. J. & MacDonell, S. G. “Evaluating prediction systems in software project estimation”. *Information & Software Technology* **54**.8 (2012), pp. 820–827.

[Shi17] Shi, J.-F. et al. “Spacecraft Pose Estimation using Principal Component Analysis and a Monocular Camera”. *AIAA Guidance, Navigation, and Control Conference*. 2017, p. 1034.

[Shi01] Shi, Y. et al. “Particle swarm optimization: developments, applications and resources”. *evolutionary computation. Proceedings of Congress on*. Vol. 1. IEEE. 2001, pp. 81–86.

[SK12] Siddiqui, J. H. & Khurshid, S. “Scaling symbolic execution using ranged analysis”. *ACM SIGPLAN Notices* **47**.10 (2012), pp. 523–536.

[Sie12] Siegmund, N. et al. “Predicting performance via automated feature-interaction detection”. *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press. 2012, pp. 167–177.

[Ste46] Stevens, S. S. “On the theory of scales of measurement” (1946).

[SP97] Storn, R. & Price, K. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. *Journal of global optimization* **11**.4 (1997), pp. 341–359.

[Tan16] Tantithamthavorn, C. et al. “Automated Parameter Optimization of Classification Techniques for Defect Prediction Models”. *38th International Conference on Software Engineering*. 2016.

[Thi14] Thibodeau, P. *Data centers are the new polluters, August 26*. Available from http://v.gd/TpSdZn. 2014.

[Top02] Topcuoglu, H. et al. “Performance-effective and low-complexity task scheduling for heterogeneous computing”. *IEEE Trans. on parallel and distributed systems* **13**.3 (2002), pp. 260–274.

[Tov84] Tovey, C. A. “A simplified NP-complete satisfiability problem”. *Discrete applied mathematics* **8**.1 (1984), pp. 85–89.

[Tsa14] Tsai, C.-W. et al. “A hyper-heuristic scheduling algorithm for cloud”. *IEEE Trans. on Cloud Computing* **2**.2 (2014), pp. 236–250.

[VD00] Vargha, A. & Delaney, H. D. “A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong”. *Journal of Educational and Behavioral Statistics* (2000).

[VL11] Veerappa, V. & Letier, E. “Understanding clusters of optimal solutions in multi-objective decision problems”. *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*. 2011, pp. 89–98.

[VL98] Veldhuizen, D. A. V. & Lamont, G. B. “Evolutionary Computation and Convergence to a Pareto Front”. *Stanford University, California*. Morgan Kaufmann, 1998, pp. 221–228.

[Vis06] Visser, W. et al. “Test input generation for java containers using state matching”. *Proceedings of the 2006 international symposium on Software testing and analysis*. ACM. 2006, pp. 37–48.

[Vöc11] Vöckler et al. “Experiences using cloud computing for a scientific workflow application”. *Proceedings of the 2nd Intl. workshop on sci. cloud computing*. ACM. 2011, pp. 15–24.

[Wan13a] Wang, S. et al. “Minimizing test suites in software product lines using weight-based genetic algorithms”. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 1493–1500.

[Wan16] Wang, S. et al. “A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering”. *Proceedings of the 38th International Conference on Software Engineering*. ACM. 2016, pp. 631–642.

[Wan13b] Wang, T. et al. “Searching for Better Configurations: A Rigorous Approach to Clone Evaluation”. *9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013.

[Wan13c] Wang, T. et al. “Searching for better configurations: a rigorous approach to clone evaluation”. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM. 2013, pp. 455–465.

[Wei04] Wei, W. et al. “Towards efficient sampling: Exploiting random walk strategies”. *AAAI*. Vol. 4. 2004, pp. 670–676.

[Whi15] Whigham, P. A. et al. “A Baseline Model for Software Effort Estimation”. *ACM Trans. Softw. Eng. Methodol.* **24**.3 (2015), 20:1–20:11.

[WM97a] Wolpert, D. H., Macready, W. G., et al. “No free lunch theorems for optimization”. *IEEE transactions on evolutionary computation* **1**.1 (1997), pp. 67–82.

[WM97b] Wolpert, D. & Macready, W. “No free lunch theorems for optimization”. *Evolutionary Computation, IEEE Transactions on* **1**.1 (1997), pp. 67–82.

[Xia18] Xia, T. et al. “Hyperparameter optimization for effort estimation”. *arXiv preprint arXiv: 1805.00336* (2018).

[YJ16] Yang, Y. & Jiang, J. “Hybrid sampling-based clustering ensemble with global and local constitutions”. *IEEE transactions on neural networks and learning systems* **27**.5 (2016), pp. 952–965.

[YB06] Yu, J. & Buyya, R. “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms”. *Scientific Programming* **14**.3-4 (2006), pp. 217–230.

[Yua99] Yuan, J. et al. “Modeling design constraints and biasing in simulation using BDDs”. *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*. IEEE Press. 1999, pp. 584–590.

[Yua04] Yuan, J. et al. “Simplifying Boolean constraint solving for random simulation-vector generation”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **23**.3 (2004), pp. 412–420.

[ZL07] Zhang, Q. & Li, H. “MOEA/D: A multiobjective evolutionary algorithm based on decomposition”. *IEEE Trans. on evolutionary computation* **11**.6 (2007), pp. 712–731.

[Zha07] Zhang, Y et al. “The Multi-Objective Next Release Problem”. *In ACM Genetic and Evolutionary Computation Conference (GECCO 2007*. 2007, p. 11.

[Zha12] Zhang, Y. et al. “The SBSE repository: A repository and analysis of authors and research articles on search based software engineering”. *crestweb. cs. ucl. ac. uk/resources/sbse repository* (2012).

[Zho06] Zhou, A. et al. “Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion”. *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE. 2006, pp. 892–899.

[Zhu16] Zhu, Z. et al. “Evolutionary multi-objective workflow scheduling in cloud”. *IEEE Trans. on parallel and distributed Systems* **27**.5 (2016), pp. 1344–1357.

[Zit01a] Zitzler et al. “SPEA2: Improving the strength Pareto evolutionary algorithm”. *TIK-report* **103** (2001).

[ZK04] Zitzler, E. & Kunzli, S. “Indicator-based selection in multiobjective search”. *International Conference on Parallel Problem Solving from Nature*. Springer. 2004, pp. 832–842.

[ZT99] Zitzler, E. & Thiele, L. “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach”. *IEEE transactions on evolutionary computation,* **3**.4 (1999), pp. 257–271.

[Zit01b] Zitzler, E. et al. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Tech. rep. 2001.

[Zit02] Zitzler, E. et al. “SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization”. *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, Barcelona, Spain, 2002, pp. 95–100.

[Zul13] Zuluaga, M. et al. “Active Learning for Multi-Objective Optimization”. *International Conference on Machine Learning (ICML)*. 2013.