Budgeted Testing Through an Algorithmic Lens

Myra B. Cohen
Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE, 68588-0115 USA
myra@cse.unl.edu

A. Pavan
Department of Computer Science
Iowa State University
Ames, IA, 50011 USA
pavan@cs.iastate.edu

N. V. Vinodchandran
Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE, 68588-0115 USA
vinod@cse.unl.edu

ABSTRACT

Automated testing has been a focus of research for a long time. As such, we tend to think about this in a coverage centric manner. Testing budgets have also driven research such as prioritization and test selection, but as a secondary concern. As our systems get larger, are more dynamic, and impact more people with each change, we argue that we should switch from a coverage centric view to a budgeted testing centric view. Researchers in other fields have designed approximation algorithms for such budgeted scenarios and these are often simple to implement and run. In this paper we present an exemplar study on combinatorial interaction testing (CIT) to show that a budgeted greedy algorithm, when adapted to our problem for various budgets, does almost as well coverage-wise as a state of the art greedy CIT algorithm, better in some cases than a state of the art simulated annealing, and always improves over random. This suggests that we might benefit from switching our focus in large systems, from coverage to budgets.

CCS Concepts

•Software and its engineering → Software verification and validation;

Keywords

Software Testing, Algorithms, Combinatorial Testing

1. INTRODUCTION

Over the years research on automated testing has taken a front and center position in the software engineering community. Techniques have been built to generate faster, smaller, more efficient test suites that can be both generated and run with little intervention. Many of these techniques focus on how much we have covered (code, products, methods, events, requirements). Yet significant challenges remain and no end is in sight. First, our systems have become highly-configurable allowing users to customize individual instances of the program while retaining a core set of functionality. The configuration spaces in these systems have been shown to be larger than can ever be exhaustively tested [4].

Second, many of our systems are interactive, using an event-driven paradigm, where tests are (unbounded) sequences, and systematic coverage is a challenge due the the explosion in their numbers [10]. Third, new testing paradigms have led to the notion of continuous test integration [5] or nightly builds [6] where software is evolving quickly and must be re-tested in short time sprints. This leads to a tightly constrained test environment where exact time allotments may be unknown. Last, we are moving to cloud-based computing, or software services where a change leading to a regression fault can impact the entire user base at once, creating additional testing risk.

As our regression cycles shorten, our configuration spaces increase, our testing sequences lengthen and our risk increases, we argue that our focus should be on a *budgeted testing problem*, rather than on a *coverage testing problem*. In systems testing we already utilize many sampling algorithms, such as combinatorial interaction testing (CIT) [3], i.e. cover all pairs or t-sets using a minimal number of tests. Yet even using these techniques, we have a need for prioritization techniques to run the most important tests first. We ponder instead, what happens if we formulate this as a budget problem and ask, "What is our coverage under a budget of N tests/configurations/event sequences/..?"

Recent work by Arcuri et al. [2] has started in this direction. They evaluate the effectiveness of CIT versus random testing and suggest that random testing stands a good probability of finding the types of faults (interaction faults) discovered by this technique. While researchers in CIT continue to build new (efficient) algorithms for making samples smaller [9], or for solving practical problems such as scaling to ultra-large size programs [8], and handling constraints (dependencies between inputs of the system) efficiently [4], they are unable to answer the question "What is the best technique to use if I will run out of time" or "How much coverage am I guaranteed to obtain given a particular algorithm and a time budget". We believe that formulating this as a budgeted problem, and viewing this under an algorithmic lens may benefit testing in new ways.

In this paper we present a first step at budgeted testing for CIT, where budget rather than coverage drives our algorithmic choice. We describe a simple budgeted greedy algorithm with a provable coverage guarantee where an explicit set of configurations to choose from are given (we call this Explicit BCA). Since in practice the search space is

too large, we apply this algorithm on a random subset of configurations, adapting it to work only on a tiny part of the search space (For MySQL which has over 100 million configurations, it randomly selects only 1000). This means that we can provably work with only a small part of the configuration space and still achieve coverage within a known percent of the maximal coverage for a given budget. In a feasibility study we implement and test this on two commonly used highly configurable systems and show that our algorithm can cover close to that of a state of the art greedy algorithm and outperforms simulated annealing. In all cases, it performs significantly better than random and in practice it appears to do much better than the theoretical bound.

2. AN ALGORITHMIC FRAMEWORK FOR BUDGETED CIT

We begin by defining the CIT coverage problem and then view this from the alternative budgeted perspective. Let $\langle f_1, \cdots, f_k \rangle$ be k-factors, so that f_i can take a value from an alphabet Σ_i of size v_i. A *test* is a k-tuple $\langle a_1, \cdots, a_k \rangle$ such that each $a_i \in \Sigma_i$. A *t-set* is a set of size t where each element of the set is a tuple of the form $\langle i, \alpha_i \rangle$ such that $1 \leq i \leq k$ and $\alpha_i \in \Sigma_i$. Given a test $T = \langle a_1, \cdots, a_k \rangle$ and a t-set c, we say that T *covers* c if for every $\langle i, \alpha_i \rangle$ in c, $a_i = \alpha_i$. A *t-wise covering array*, denoted as $\mathrm{CA}(t, k, v_1, \cdots, v_k)$, is a set of tests such that *every* t-set is covered by some test from $\mathrm{CA}(t, k, v_1, \cdots, v_k)$. The size of the covering array is the cardinality of the set $\mathrm{CA}(t, k, v_1, \cdots, v_k)$. The parameter t tells us how strongly to test the combinations of settings. If $t = 2$ we call this *pairwise* CIT. The covering array problem is to find a t-wise covering array of *smallest* size.

In the *budgeted version* of the covering array problem we are given a budget B with a number of tests (or configurations) that we can run. For a collection of tests $\mathcal{T}$, we say that $\mathcal{T}$ covers a t-set c, if there exist some test in $\mathcal{T}$ that covers c. Given $\mathcal{T}$, let $\mathrm{Cov}(\mathcal{T})$ be the number of t-sets it covers. We define the Budgeted Covering Array Problem (in short BCA problem).

Definition 1 (BCA Problem). *Given a budget B, compute a set of tests $\mathcal{T}$ so that $|\mathcal{T}| = B$ and $\mathrm{Cov}(\mathcal{T})$ is maximum.*

An algorithm for the BCA problem is of practical interest, however, this problem is unlikely to have a guaranteed efficient algorithm. Hence designing an efficient algorithm with provable approximation guarantee will be of practical significance. An immediate question is: what is the coverage of a random set of B tests? Using the probabilistic method [1] the following can be shown.

Theorem 2. *Let $\mathcal{T}$ be a set of B tests chosen uniformly at random. Then with probability $> (1 - \delta)$, $\mathrm{Cov}(T) \geq \delta(\frac{v^t}{v^t - 1})^B$.*

One of the challenges in designing approximation algorithm for the BCA problem is that the search space of all possible tests is $\Pi_{i=1}^{k} v_i$ which is exponential in k even if $|v_i| = 2$ for all i. We formulate the following Explicit BCA problem for which we give an efficient algorithm with a provable approximation guarantee.

Definition 3 (Explicit BCA Problem). *Given a collection $\mathcal{T}$ of tests, and a budget B. Find a subset $\mathcal{S}^* \subseteq \mathcal{T}$ of size B so that $\mathrm{Cov}(\mathcal{S}^*)$ is maximum. That is, we want to compute $\mathcal{S}^*$ so that:*

$$\mathcal{S}^* = \arg\max_{\mathcal{S} \subseteq \mathcal{T}} \mathrm{Cov}(\mathcal{S})$$

Definition 4. *An algorithm $\mathcal{A}$ is an c (for $c < 1$) approximation algorithm for a maximization problem if for any instance x of the problem, $\mathcal{A}(x)$ outputs a solution whose value is at least c times the value of the best solution.*

Theorem 5. *Algorithm 1 is an efficient $(1-1/e)$ approximation algorithm for the* Explicit BCA *Problem. Here e is the base to natural logarithm.*

For this algorithm, the approximation is always within a constant of $(1 - 1/e) = 63.2\%$ regardless of the coverage criteria. This algorithm gets an initial set of test $\mathcal{T}$ and a budget B as input, and greedily selects tests from the set $\mathcal{T}$.

Algorithm 1: Greedy Explicit BCA

```
Input: Set of tests T and budget a B
S = φ;
For i = 1 to B
    Find T ∈ T so that Cov(S ∪ {T}) − Cov(S) is
    maximum;
    S = S ∪ {T};
Output S.
```

The proof that Algorithm 1 is a $(1 - 1/e)$ approximation algorithm follows from the theory of sub-modular function optimization [7] and we omit the details here. Note that the run time of this algorithm is proportional to the product of size of the initial test $\mathcal{T}$ and the budget B.

We propose the following approach for designing an efficient approximation algorithm for the BCA problem: first produce a set of tests $\mathcal{T}$ and run the above approximation algorithm for Explicit BCA with $\mathcal{T}$ as input. This raises an important question regarding the choice of initial test set $\mathcal{T}$. Notice that choosing $\mathcal{T}$ to be the set of all possible tests (exponentially many) will give a solution to the BCA problem. However this takes exponential time. This leads to the following definition.

Definition 6. *Given a budget B and a set of tests $\mathcal{T}$ is* B-maximal *if the optimum value of the* Explicit BCA *problem with budget B and test set $\mathcal{T}$ as input is same as the optimum value of the* BCA *problem with budget B as input.*

Designing an efficient algorithm to construct a B-maximal test set of small size (polynomial in k and v^t) is a significant future direction to explore. Such an algorithm will yield an efficient solution to the BCA problem with a provable guarantee on the quality of solution. We provide a preliminary evidence that the proposed direction to solve the BCA problem will be fruitful, however, in this paper. An intuitive choice for a B-maximal set is a random test of size which with a high probability is a covering array. The following theorem gives a bound on this number [1]. Here $v = \max\{v_1, \cdots, v_k\}$.

Theorem 7. *If we randomly choose N tests where $N \geq v^t(\ln 1/\delta + t \ln vk)$, then with probability $\geq 1 - \delta$, the tests forms a covering array.*

Based on the above theorem, a heuristic for BCA (called GreedyBCA) is as follows: given a budget B, pick a random test set $\mathcal{T}$ of size N and run Algorithm 1 with $\mathcal{T}$ and B as input. In the next section we perform a feasibility study of this Greedy BCA.

3. FEASIBILITY STUDY

We set out to ask questions about whether the proposed greedy budgeted covering array algorithm (GreedyBCA) is potentially useful in practice. We ask two research questions.

RQ1: Does GreedyBCA produce coverage within the expected threshold?

RQ2: How does GreedyBCA compare with existing CIT algorithms for different budgets?

Study Setup. We selected two applications that have been widely used in testing studies for configurable software and that have slightly different characteristics, MySQL and GCC. The first application, MySQL, a configurable database application, has been studied in a nightly build environment [6]. The second application, GCC, a popular open source compiler, has been used in multiple studies on configurability. The exhaustive configuration space for MySql is $141,557,760$ and it is in the order of 10^{61} for GCC [4]. For this study we ignore the model constraints and leave their incorporation for future work.

The CIT models and other information about these are shown in Table 1. The second column provides the details of the model. We use a common shortcut where x^y means there are y factors with x values. For instance, GCC has 189 binary and 10 ternary options, while MySql has only 23 factors, of which 18 are binary, three are ternary, one has four values and one five values.

Table 1: Characteristics of Subjects. x^y means y factors have x values. PrCount is the count of pairs to be covered. Size is the initial random test set.

Subject	Model	PrCount	Size
MySql	$2^{18}3^34^15^1$	1388	1000
GCC	$2^{189}3^{10}$	82809	360

We first calculated the size of the initial random set used in Algorithm 1. The size of the initial random set is obtained based on Theorem 7. For MySql, the maximum alphabet size v is 5. To make the error very small we set δ as $1/2^{30}$. Then $\ln 1/\delta$ is approximately 21, and $t \ln vk = 2 \times \ln 115$ (for $t = 2$) whose value approximately equals 10. Thus $\ln 1/\delta + t \ln vk \sim 31$ and $v^t = 25$, and hence the initial random set of size 1000 is a generous random sample to make sure that we have a covering array to start with. Similar calculations yield 360 as a choice for the size of the initial test set for GCC. Note that while GCC is the larger model (199 configurable options versus 23), since MySql has a higher maximum v the initial random set to choose from is larger in MySql. We then ran GreedyBCA as well as a budgeted version of two existing algorithms (AETG and Simulated Annealing that were used in [4]). We also collect the first *budget* rows from our random sample for comparison. We used budgets of 10 (approximately half the size of a full covering array), the average size of 10 independent runs of the simulated annealing algorithm, and the average size of 10 runs of the AETG algorithm. These are 20 and

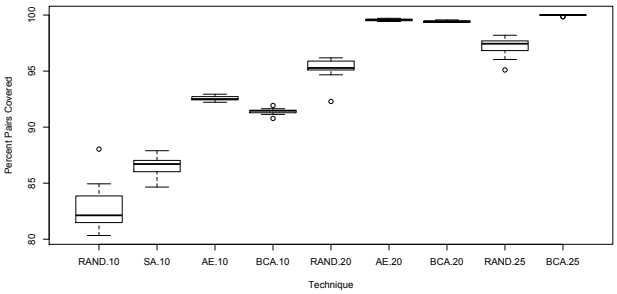

Figure 1: MySQL: Budgeted coverage

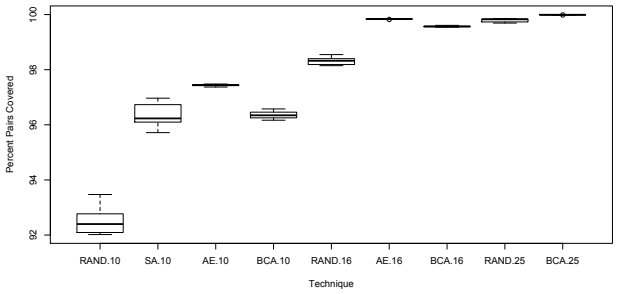

Figure 2: GCC: Budgeted coverage

25 respectively for MySQL and 16 and 25 for GCC. We measure the pairwise coverage of the resulting test suites for each budget. For the CIT algorithms we count the coverage for the first N rows, where N is our budget. All experiments were run 10 times.

The results are shown in Table 2 and boxplots (Figures 1 and 2). In the table we show the number of missing pairs for each budget. We use RAND-N, SA-N, AE-N and BCA-N to represent the random, simulated annealing, AETG and GreedyBCA with a budget of N. In the boxplots we show the percent coverage on the `y-axis`. In GCC there are more pairs to be covered so the percentage differences are on a smaller scale, but the relative results are similar.

The simulated annealing (SA) performs worse than the budgeted version of AETG, but both outperform random. Our algorithm performs better than SA and is very competitive with AETG. A possible intuitive explanation is that (and is similar to results from prioritization of CIT) greedy algorithms pack more pairs into the test suite early and then exhibit a long tail, while simulated annealing works on the entire set of configurations at once. We note that our coverage of our base algorithm is within $1 - 1/e$ which is approximately 63.2% of the maximum coverage for our budget. We don't have the ground truth for small budgets to empirically evaluate this. For the larger budgets our algorithm achieves 100 percent coverage in some runs (for example in MySql). This illustrates that the practical performance is much better than the guaranteed 63.2%.

We believe that designing an algorithm to construct a B-maximal set and using such a set as input to the greedy algorithm may also outperform the budgeted version of AETG and may lead to new practical algorithms. This direction is future work. We also note that we implemented this version of GreedyBCA inside of our AETG framework (removing

Table 2: Number of missing pairs for 10 runs of each algorithm

	MySQL								
No.	RAND-10	SA-10	AE-10	BCA-10	RAND-20	AE-20	BCA-20	RAND-25	BCA-25
1	258	168	98	112	53	7	9	25	0
2	166	213	108	119	54	4	9	68	0
3	249	185	105	118	68	7	9	44	2
4	209	184	105	123	107	6	8	44	0
5	247	194	104	116	63	5	8	34	0
6	257	170	105	119	63	6	6	55	0
7	235	184	104	118	74	6	6	32	0
8	224	190	101	121	57	7	7	35	0
9	250	180	103	118	68	8	9	26	0
10	273	202	98	128	68	4	9	36	2
Avg	236.8	187.0	103.1	119.2	67.5	6.0	8.0	39.9	0.4
	GCC								
No	RAND-10	SA-10	AE-10	BCA-10	RAND-16	AE-16	BCA-16	RAND-25	BCA-25
1	5530	2664	2177	2835	1402	131	378	142	6
2	5406	3112	2135	3043	1295	146	351	221	4
3	6457	2913	2105	2932	1441	133	350	255	8
4	6548	3128	2114	3137	1374	128	356	142	4
5	6565	3547	2107	2959	1501	134	329	132	6
6	6610	2706	2087	3103	1515	133	324	122	6
7	6090	2512	2097	2932	1325	129	368	225	11
8	6476	3186	2125	3174	1535	135	375	128	6
9	5987	3385	2142	3028	1202	137	363	140	12
10	6132	3231	2118	3025	1343	140	355	190	7
Avg	6180.1	3038.4	2120.7	3016.8	1393.3	134.6	354.9	169.7	7.0

the AETG logic), hence we know that it runs at least as fast as AETG. We expect that we can optimize this further.

4. CONCLUSIONS

We have argued for viewing testing as a budgeted problem rather than a coverage problem, and using an algorithmic lens to achieve provable guarantees. We presented a simple budgeted greedy algorithm for CIT and have evaluated this against two existing CIT algorithms and purely random selection of tests for various budgets. We have shown that while the existing greedy algorithm (AETG) provides the best coverage for smaller budgets, the budgeted greedy algorithm performs much better than the theoretical bound. We also note that if the budget is large enough, then the simulated annealing has the highest coverage. In future work we will formalize this problem further, incorporating constraints and fault detection results as well as performing larger scale empirical studies. We are also exploring ways to find a more optimal starting set.

5. ACKNOWLEDGMENTS

This work is supported in part by NSF grants CCF-1161767 for the first author, CCF-1421163 for the second author, and CCF-1422668 for the third author.

6. REFERENCES

[1] N. Alon and J. Spencer. *The Probabilistic method*. Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & sons, New York, Chichester, Brisbane, 1992.

[2] A. Arcuri and L. Briand. Formal analysis of the probability of interaction fault detection using random testing. *IEEE Trans. on Soft. Eng.*, 38(5):1088–1099, 2012.

[3] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Trans. on Soft. Eng.*, 23(7):437–444, 1997.

[4] M. B. Cohen, M. B. Dwyer, and J. Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. on Soft. Eng.*, 34(5):633–650, 2008.

[5] S. Elbaum, G. Rothermel, and J. Penix. Techniques for improving regression testing in continuous integration development environments. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 235–245, 2014.

[6] S. Fouché, M. B. Cohen, and A. Porter. Incremental covering array failure characterization in large configuration spaces. In *Intl. Symp. on Soft. Test. and Analysis*, pages 177–187, July 2009.

[7] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

[8] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. l. Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *Intl. Conf. on Soft. Test., Verif. and Valid.*, pages 459–468, 2010.

[9] A. Rodriguez-Cristerna and J. Torres-Jimenez. A simulated annealing with variable neighborhood search approach to construct mixed covering arrays. *Electronic Notes in Discrete Mathematics*, 39:249 – 256, 2012.

[10] X. Yuan, M. Cohen, and A. Memon. GUI interaction testing: Incorporating event context. *IEEE Trans. on Soft. Eng.*, 37(4):559 –574, 2011.