MaLT: Machine-Learning-Guided Test Case Design and Fault Localization of Complex Software Systems

1st Yi Ji
DOE & Reliability
JMP Statistical Discovery LLC
Cary, US
Irene.Ji@jmp.com

2nd Simon Mak
Department of Statistical Science
Duke University
Durham, US
sm769@duke.edu

3rd Ryan Lekivetz
DOE & Reliability
JMP Statistical Discovery LLC
Cary, US
Ryan.Lekivetz@jmp.com

4th Joseph Morgan
DOE & Reliability
JMP Statistical Discovery LLC
Cary, US
Joseph.Morgan@jmp.com

***Abstract*—Software testing is essential for the reliable and robust development of complex software systems. This is particularly critical for cyber-physical systems (CPS), which require rigorous testing prior to deployment. The complexity of these systems limits the use of formal verification methods. Furthermore, testing and fault localization can be very costly. To mitigate this cost, we outline in this work a holistic machine-learning-guided test case design and fault localization (MaLT) framework, which leverages recent probabilistic machine learning methods to accelerate the testing of complex software systems. MaLT consists of three steps: (i) the construction of a suite of test cases using a covering array for initial testing, (ii) the investigation of posterior root cause probabilities via a Bayesian fault localization procedure, then (iii) the use of such Bayesian analysis to guide selection of subsequent test cases via active learning. The proposed MaLT framework can thus facilitate efficient identification and subsequent diagnosis of software faults with limited test runs. This framework has potential for integration with an assertion-based test oracle approach, which may prove to be an efficient and cost-effective way of integrating light-weight formal methods with testing.**

***Index Terms*—Active learning, Bayesian modeling, Combinatorial testing, Fault localization, Probabilistic machine learning.**

I. Introduction

Software testing – the process of executing a program with the intent of finding errors [1] – is an essential step in the reliable and robust development of complex software systems. Such testing aims to reveal and fix as many bugs as possible prior to the release of a software application, thus greatly reducing the risk of failures for the end-user. This is critical as nearly all facets of daily life involve human interaction with software applications. In particular, cyber-physical systems (CPS) require rigorous testing prior to deployment. However, the complexity of such systems introduces two critical challenges: (i) each test run can be costly to perform [2], and (ii) there may be many inputs and thus exponentially many input combinations to explore. The comprehensive testing of complex systems can therefore be highly time- and resource-intensive. We outline below a holistic machine-learning-guided test case design and fault localization (MaLT) framework, which leverages recent machine learning methods for accelerating software testing in practical turnaround times.

Given challenges (i) and (ii), a key bottleneck is that the testing of *all* input combinations is typically infeasible for complex software systems. One solution is to carefully design a small initial set of test cases, geared towards detecting as many faults as possible. In our experiments, we find a covering array [3] provides an appealing design for initial testing. Such a design ensures coverage of all combinations up to a certain level of interaction; more on this later in the MaLT pipeline.

Next, after initial tests are performed and failures observed, such data needs to be used for *fault localization* [4], i.e., to pinpoint potential root causes of the observed failures. Such a fault localization problem is also highly challenging given challenges (i) and (ii), as one needs to consider a large number of potential root cause scenarios given limited test run data. For example, consider a system with 10 inputs each with 2 levels, which yields a total of $\sum_{i=1}^{10} \binom{10}{i} 2^i = 59048$ different input combinations. Assuming each input combination is either a root cause or not, this then results in an astounding 2^{59048} different scenarios to consider for fault localization! Furthermore, much of the existing literature on tackling fault localization are *deterministic*, and thus shed little insight on the *probability* of a combination being a root cause. Such probabilities are important for reliable fault localization; they provide a principled statistical approach for assessing root cause risks, and thus a principled measure of confidence that an identified suspicious combination is (or

2832-6520/24/$31.00 ©2024 IEEE
DOI 10.1109/MEMOCODE63347.2024.00011

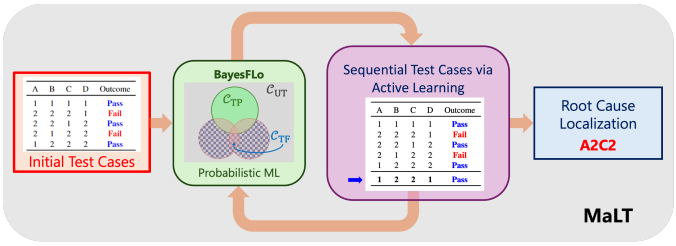

Fig. 1. Workflow for the proposed MaLT framework.

is not) a root cause. In what follows, we introduce within MaLT the Bayesian fault localization procedure in [5], which leverages recent probabilistic machine learning (ML) modeling and optimization techniques for estimating such probabilities.

Finally, with this Bayesian analysis in hand, we can leverage such analysis to select a subsequent case (or a set of cases) for further testing. In ML, this strategy of "actively" using learned information for subsequent data collection is known as *active learning*; see [6], [7]. For active learning, a desirable ingredient is a probabilistic quantification of model uncertainty [8], [9], to help guide the selection of subsequent data to reduce model uncertainty and maximize learning. We introduce later within MaLT a novel risk-based active learning method, which leverages the aforementioned Bayesian fault localization analysis for sequential test run design.

MaLT can be considered as a "pseudo-exhaustive" testing approach [10], which leverages carefully designed test cases to obtain empirical observations of the software system for efficient and effective location of software faults. As a machine-learning-guided approach, it has potential to be integrated with an assertion-based test oracle approach and can thus be regarded as a "light-weight formal method" [11].

Figure 1 visualizes the workflow of the proposed MaLT framework; each part will be elaborated on. Section II outlines the covering array approach for initial test case design. Section III describes the Bayesian fault localization procedure in [5] using this data. Section IV outlines an active learning procedure for designing subsequent test runs using such Bayesian analysis. Section V concludes the paper.

II. MaLT: Initial Test Cases via Covering Arrays

Consider first the problem of designing initial test cases for fault localization. As mentioned earlier, the sheer number of possible input combinations renders the testing of every input combination to be infeasible in practical systems. Thus, a carefully-designed initial test set is needed for efficient and effective fault localization. The goal is to design test sets to cover as many combinations as possible.

One promising design strategy is the covering array (CA; [3]). A covering array is a $M \times I$ array designed such that every column combination of order $k \leq I$ appears at least once in the M rows. Here, k is known as the strength of the CA. The columns of the CA can be thought of as factors (or inputs), in which case the rows can be thought of as runs. Table I shows a strength-2 CA design, using $I = 4$ input factors with $J = 2$ levels. The levels here refer to distinct values of each factor.

Input Factors	A	B	C	D
Run 1	1	1	1	1
Run 2	2	2	2	1
Run 3	2	2	1	2
Run 4	2	1	2	2
Run 5	1	2	2	2

TABLE I
A STRENGTH-2 CA FOR INITIAL TESTING OF A SYSTEM WITH FOUR INPUT FACTORS, EACH WITH TWO LEVELS.

We see that, with only $M = 5$ runs, every possible two-factor combination appears in at least one test run. Thus, assuming all bugs arise from an input combination of at most order k, a strength-k CA test set would "cover" every bug in the system resulting in one or more failed test cases. Comparing to the robust testing approach using orthogonal arrays [12], which forces each combination to appear "equally often", CAs provide a more economical design since the "equal-run" requirement is relaxed [13]. We thus make use of CAs as initial designs in MaLT: its reduced run size allows for efficient testing and effective fault localization with limited (expensive) test runs, and the additional runs saved can then be used for sequential test runs via active learning (see Section IV).

There is a rich body of literature and software on efficient CA construction [13]. [14] provides a comprehensive review of construction algorithms for CAs, including direct [15], recursive [16], optimization [17], genetic [18], and backtracking [19] algorithms. [20] reviews a list of useful tools for constructing CAs; in particular, the Advanced Combinatorial Testing System (ACTS, [21]) is widely used in practice [13].

III. MaLT: Bayesian Fault Localization

With initial test cases performed, consider next the problem of fault localization, which aims to pinpoint potential root causes that triggered the observed failures. As mentioned earlier, such fault localization should preferably shed light on the probability of each combination being a root cause. One way to achieve this is via a Bayesian learning approach, where prior root cause probabilities are assigned on each input combination, then updated by conditioning on the observed test results. Such a Bayesian framework offers two key advantages over the current deterministic approaches: it provides a flexible framework for integrating prior *structural* information on expected root cause behavior, and permits the incorporation of prior *domain* knowledge from test engineers [22] for accelerating fault localization. We summarize below the Bayesian fault localization (BayesFLo) learning model in [5], which achieves this goal. Figure 2 presents the workflow of the BayesFLo model, which takes in test cases with outcomes and computes the posterior root cause probability of each suspicious input combination.

We first introduce some notation. Consider a software system with I categorical factors, with factor i having J_i distinct levels. Let $(\mathbf{i}, \mathbf{j})_K$ denote an input combination of K factors $\mathbf{i} = (i_1, \cdots, i_K)$, with corresponding levels $\mathbf{j} = (j_1, \cdots, j_K)$. For example, $((1, 2), (1, 1))$ denotes an input combination with both inputs 1 and 2 at level 1. For single-factor inputs (i.e., $K = 1$), this notation can be simplified to (i, j). In practice,

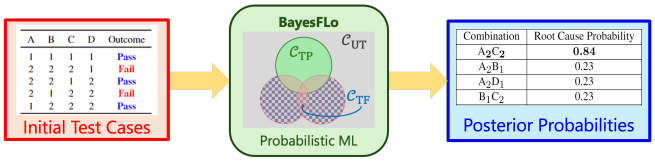

Fig. 2. Workflow of MaLT: Bayesian Fault Localization.

we recommend setting the largest K to $t+1$ where t is the strength of the covering array, unless it leads to excessive computational cost. This is to allow for some flexibility in root cause exploration.

Next, let $Z_{(\mathbf{i},\mathbf{j})_K}$ be a binary indicator for whether the input combination $(\mathbf{i},\mathbf{j})_K$ is indeed a root cause. Following the Bayesian paradigm, we assign the following independent Bernoulli priors on $Z_{(\mathbf{i},\mathbf{j})_K}$:

$$Z_{(\mathbf{i},\mathbf{j})_K} \overset{indep.}{\sim} \text{Bern}(p_{(\mathbf{i},\mathbf{j})_K}), \tag{1}$$

where $p_{(\mathbf{i},\mathbf{j})_K}$ captures the engineer's *a priori* probability of $(\mathbf{i},\mathbf{j})_K$ being a root cause. The elicitation of all prior probabilities can however be cumbersome; a simpler approach may be to adopt the following product form:

$$p_{(\mathbf{i},\mathbf{j})_K} = \prod_{k=1}^{K} p_{(i_k,j_k)}. \tag{2}$$

Here, the user only needs to specify prior probabilities $p_{(i,j)}$ on the single-factor combinations (i,j) based on their prior domain knowledge. Such a product-form prior further embeds important prior structural information on expected root cause behavior, by capturing the combination hierarchy and heredity principles in [13]. The combination hierarchy principle asserts that combinations involving fewer inputs are more likely to be failure-inducing than those involving more inputs; this is captured in (2) by assigning increasingly smaller prior probabilities on combinations with higher interaction order. The combination heredity principle asserts that a combination is more likely to be failure-inducing when some of its component inputs are more likely to be failure-inducing; this is captured via the product form in (2), where prior probabilities are multiplied over each input.

With this prior specified, the desired posterior probabilities $\mathbb{P}(Z_{(\mathbf{i},\mathbf{j})_K} = 1|\text{data})$ can then be computed as follows. We first categorize all input combinations into three groups:

1) *Tested-and-Passed (TP)*: This group, denoted as $\mathcal{C}_{\text{TP}}$, consists of combinations $(\mathbf{i},\mathbf{j})_K$ that are included in at least one passed test case.
2) *Tested-and-Failed (TF)*: This group, denoted as $\mathcal{C}_{\text{TF}}$, consists of combinations $(\mathbf{i},\mathbf{j})_K$ that are included in at least one failed test case but not included in any passed test cases.
3) *Untested (UT)*: This group, denoted as $\mathcal{C}_{\text{UT}}$, consists of remaining combinations $(\mathbf{i},\mathbf{j})_K$ that are not contained in the earlier groups.

The reason for this categorization is as follows. For a TP combination $(\mathbf{i},\mathbf{j})_K$, one can easily show (see [5]) that its posterior probability of being a root cause is 0, since such a combination has been observed in a passed test case and thus cannot be a root cause. For a UT combination $(\mathbf{i},\mathbf{j})_K$, since such a combination was untested, its posterior probability is simply its prior probability $p_{(\mathbf{i},\mathbf{j})_K}$ as no data was observed on this combination (see [5] for a rigorous argument).

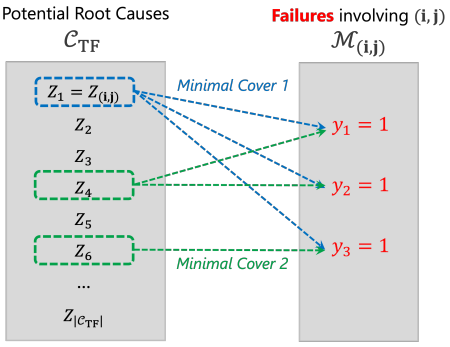

Fig. 3. Visualizing the bipartite graph representation and two minimal covers for failures involving the combination $(\mathbf{i},\mathbf{j})_K$.

The remaining computation of posterior probabilities for TF combinations is a more challenging task. As noted in [5], the brute-force computation of this probability for a single TF combination may be doubly-exponential in the number of inputs I, which is prohibitively expensive. The following reformulation of this probability can permit tractable computation. For a TF combination $(\mathbf{i},\mathbf{j})_K$, let $\mathcal{M}_{(\mathbf{i},\mathbf{j})_K}$ be the failure cases that include $(\mathbf{i},\mathbf{j})_K$, and $\mathcal{M}_{-(\mathbf{i},\mathbf{j})_K}$ be the failure cases that do not contain $(\mathbf{i},\mathbf{j})_K$. One can show (see [5]) that its desired posterior probability can be rewritten as:

$$\mathbb{P}(Z_{(\mathbf{i},\mathbf{j})_K} = 1|\text{data}) = \mathbb{P}(Z_{(\mathbf{i},\mathbf{j})_K} = 1|E_{(\mathbf{i},\mathbf{j})_K}) = \frac{p_{(\mathbf{i},\mathbf{j})_K}}{\mathbb{P}(E_{(\mathbf{i},\mathbf{j})_K})}. \tag{3}$$

Here, $E_{(\mathbf{i},\mathbf{j})_K}$ is the event that all failure cases containing $(\mathbf{i},\mathbf{j})_K$ (namely, $\mathcal{M}_{(\mathbf{i},\mathbf{j})_K}$) can be explained by a collection of TF combinations being root causes.

To compute the probability $\mathbb{P}(E_{(\mathbf{i},\mathbf{j})_K})$ in Equation (3), [5] makes use of an interesting link between the event $E_{(\mathbf{i},\mathbf{j})_K}$ and a related problem of minimal set covering on an appropriate bipartite graph representation. Figure 3 shows an example of this bipartite graph representation, where left nodes represent all TF combinations and right nodes represent all failed test cases in $\mathcal{M}_{(\mathbf{i},\mathbf{j})_K}$. Here, an edge between a left node and a right node suggests that the TF combination on the left is included in the failed test case on the right. A minimal set cover of this bipartite graph is then defined as an irreducible collection of TF combinations (on the left) that covers or explains all failed test cases (on the right). With this, one can show (details in [5]) that the computation of $\mathbb{P}(E_{(\mathbf{i},\mathbf{j})_K})$ amounts to finding all minimal set covers of the above bipartite graph representation, for which polynomial-time algorithms exist [23], [24]. A promising practical approach is to formulate this as an integer linear program (ILP; [25]), which can be solved via state-of-the-art optimization solvers such as Gurobi [26]; details on this can be found in [5].

IV. MaLT: Sequential Test Cases via Active Learning

Finally, with the above Bayesian fault localization analysis in hand, consider the problem of leveraging such analysis for selecting subsequent test cases. Such an active learning approach is particularly useful when there are many input factors (or factor levels) or when there are many bugs in the system; in such cases, the localization of root causes may be difficult with an initial test set, and "actively-designed" sequential runs can help accelerate localization after initial testing. Much of the active learning literature in ML, however, focuses on active learning for improving model predictive accuracy, which is not the goal here. We thus present next a novel *risk-based* active learning procedure, which leverages the trained BayesFLo learning model to target subsequent test runs for localization.

We first require a criterion for selecting a subsequent test run $\mathbf{t}_{M+1}$ given an initial test set of M runs. A natural approach is to define a criterion that captures the tester's risk for false positive and false negative detection of a fault. To make this concrete, let $\delta_{(\mathbf{i},\mathbf{j})_K} \in \{0,1\}$ denote a binary classifier for predicting whether a combination $(\mathbf{i},\mathbf{j})_K$ is a root cause (i.e., $\delta_{(\mathbf{i},\mathbf{j})_K} = 1$) or not (i.e., $\delta_{(\mathbf{i},\mathbf{j})_K} = 0$). Given the true root cause indicator $Z_{(\mathbf{i},\mathbf{j})_K}$, a reasonable loss function might be:

$$L(Z_{(\mathbf{i},\mathbf{j})_K}, \delta_{(\mathbf{i},\mathbf{j})_K}) = \begin{cases} W, & Z_{(\mathbf{i},\mathbf{j})_K} = 1,\ \delta_{(\mathbf{i},\mathbf{j})_K} = 0, \\ 1, & Z_{(\mathbf{i},\mathbf{j})_K} = 0,\ \delta_{(\mathbf{i},\mathbf{j})_K} = 1, \\ 0, & Z_{(\mathbf{i},\mathbf{j})_K} = \delta_{(\mathbf{i},\mathbf{j})_K}, \end{cases} \tag{4}$$

where $W > 1$ is a user-specified value. The first line in (4) considers the case where a true root cause combination is incorrectly classified by the learning model as a non-root-cause, the second line is the case where a non-root-cause combination is incorrectly classified as a root cause, and the last line is the case of correct classification. Here, $W > 1$ reflects the fact that the risk of missing a root cause is typically greater than the risk of misclassifying a non-root-cause.

From the previous Bayesian analysis with M test runs, we have already computed the posterior root cause probabilities $\hat{p}_{(\mathbf{i},\mathbf{j})_K} := \mathbb{P}(Z_{(\mathbf{i},\mathbf{j})_K} = 1|\text{data})$ for each combination $(\mathbf{i},\mathbf{j})_K$. Using this, one can show the Bayes-optimal classifier $\delta^{\text{opt}}_{(\mathbf{i},\mathbf{j})_K}$ (see [27]) under the loss function (4) takes the form:

$$\delta^{\text{opt}}_{(\mathbf{i},\mathbf{j})_K} = \begin{cases} 1, & \hat{p}_{(\mathbf{i},\mathbf{j})_K} \geq \frac{1}{1+W}, \\ 0, & \hat{p}_{(\mathbf{i},\mathbf{j})_K} < \frac{1}{1+W}. \end{cases} \tag{5}$$

Given observed data from the initial M-run test set (denoted as $\mathcal{D}_M$), the posterior Bayes risk of this classifier for $(\mathbf{i},\mathbf{j})_K$ can then be evaluated as:

$$\begin{aligned} r_{(\mathbf{i},\mathbf{j})_K}(\mathcal{D}_M) &= \mathbb{E}_{\boldsymbol{Z}|\mathcal{D}_M}\left[L(Z_{(\mathbf{i},\mathbf{j})_K}, \delta_{(\mathbf{i},\mathbf{j})_K})\right] \\ &= W\hat{p}_{(\mathbf{i},\mathbf{j})_K} \cdot \mathbb{I}\left(\hat{p}_{(\mathbf{i},\mathbf{j})_K} < \frac{1}{1+W}\right) \\ &\quad + (1 - \hat{p}_{(\mathbf{i},\mathbf{j})_K}) \cdot \mathbb{I}\left(\hat{p}_{(\mathbf{i},\mathbf{j})_K} \geq \frac{1}{1+W}\right), \end{aligned} \tag{6}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

We can now define the proposed risk-based utility criterion for active learning. Intuitively, the next test run $\mathbf{t}_{M+1}$ should ideally maximize the reduction of Bayes risk as defined in (6). This risk reduction can be formulated as:

$$\begin{aligned} &\Delta r_{(\mathbf{i},\mathbf{j})_K}(\boldsymbol{t}_{M+1}) := \\ &\qquad r_{(\mathbf{i},\mathbf{j})_K}(\mathcal{D}_n) - \mathbb{E}_{y_{M+1}|\mathcal{D}_M,\boldsymbol{t}_{M+1}}\left[r_{(\mathbf{i},\mathbf{j})_K}(\mathcal{D}_{M+1})\right]. \end{aligned} \tag{7}$$

Here, y_{M+1} denotes the outcome of the new test case $\mathbf{t}_{M+1}$; as such an outcome is unknown, we can average its risk over its posterior distribution given observed test data, i.e., $y_{M+1}|\mathcal{D}_M, \boldsymbol{t}_{M+1}$. With this, we can then select the next test case via the following optimization problem:

$$\boldsymbol{t}^{\text{opt}}_{M+1} \leftarrow \text{argmax}_{\boldsymbol{t}_{M+1}} \sum_{(\mathbf{i},\mathbf{j})_K \in \mathcal{C}_{\text{TF}}} \Delta r_{(\mathbf{i},\mathbf{j})_K}(\boldsymbol{t}_{M+1}). \tag{8}$$

In other words, the selected $\boldsymbol{t}^{\text{opt}}_{M+1}$ should maximize the total risk reduction over all combinations $(\mathbf{i},\mathbf{j})$ in $\mathcal{C}_{\text{TF}}$, the set of TF combinations for which we wish to localize observed faults. An analogous active learning formulation can be used for designing multiple (i.e., batches) of subsequent test runs.

Finally, such an active learning procedure can be iteratively performed to accelerate fault localization of complex systems. After selecting a next test case (or batch of cases) from (8), one then investigates this case and adds its outcome (pass or failure) to the updated training data $\mathcal{D}_{M+1}$. We then refit the BayesFLo learning model (Section III), and use its updated posterior probabilities $\hat{p}_{(\mathbf{i},\mathbf{j})_K}$ to actively design further test cases. These two steps are then iteratively repeated until either a computational budget is exhausted, or until the analysis shows only a few combinations have high posterior probabilities (in which case a test engineer can directly investigate such combinations).

V. Conclusion

We presented here a novel machine-learning-guided test case design and fault localization framework (MaLT), with potential for improving the efficiency and effectiveness of testing and fault localization of expensive software systems. MaLT consists of three main steps: a carefully-designed initial test set via covering arrays, a scalable Bayesian learning model for fault localization, and the use of such Bayesian analysis for active learning. In leveraging such recent developments in probabilistic machine learning, the proposed MaLT can potentially greatly accelerate the identification and subsequent diagnosis of software faults with limited test runs. Numerical experiments and applications (see [5]) show promising performance for the first two steps of MaLT (covering arrays and Bayesian fault localization); the last step on sequential test case design is under development. Finally, the MaLT framework also has potential for integration with an assertion-based test oracle approach as proposed in [11], and may prove to be an efficient and cost-effective way of integrating formal methods with testing.

Acknowledgment

This work was supported by NSF CSSI 2004571, NSF DMS 2210729, NSF DMS 2220496, NSF DMS 2316012 and DE-SC0024477.

REFERENCES

[1] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The Art of Software Testing*. Wiley Online Library, 2004, vol. 2.

[2] A. Corso, R. Moss, M. Koren, R. Lee, and M. Kochenderfer, "A survey of algorithms for black-box safety validation of cyber-physical systems," *Journal of Artificial Intelligence Research*, vol. 72, pp. 377–428, 2021.

[3] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche*, vol. 59, no. 1, 2, pp. 125–172, 2004.

[4] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, and D. Li, "Software fault localization: an overview of research, techniques, and tools," *Handbook of Software Fault Localization: Foundations and Advances*, pp. 1–117, 2023.

[5] Y. Ji, S. Mak, R. Lekivetz, and J. Morgan, "BayesFLo: Bayesian fault localization of complex software systems," *arXiv preprint arXiv:2403.08079*, 2024.

[6] B. Settles, "Active learning literature survey," 2009.

[7] R. B. Gramacy, *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. CRC press, 2020.

[8] S. Mak, H. S. Yushi, and Y. Xie, "Information-guided sampling for low-rank matrix completion," in *ICML Workshop on Information Theoretic Methods for Rigorous, Responsible, and Reliable Machine Learning*, 2017.

[9] S. Mak and Y. Xie, "Maximum entropy low-rank matrix recovery," *IEEE Journal of Selected Topics in Signal Process*, vol. 12, no. 5, pp. 886–901, 2018.

[10] D. R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in *2006 30th Annual IEEE/NASA Software Engineering Workshop*. IEEE, 2006, pp. 153–158.

[11] D. R. Kuhn, R. N. Kacker, Y. Lei *et al.*, "Practical combinatorial testing," *NIST Special Publication*, vol. 800, no. 142, p. 142, 2010.

[12] R. Brownlie, J. Prowse, and M. S. Phadke, "Robust testing of AT&T PMX/StarMAIL using OATS," *AT&T Technical Journal*, vol. 71, no. 3, pp. 41–47, 1992.

[13] R. Lekivetz and J. Morgan, "On the testing of statistical software," *Journal of Statistical Theory and Practice*, vol. 15, no. 4, p. 76, 2021.

[14] V. V. Kuliamin and A. Petukhov, "A survey of methods for constructing covering arrays," *Programming and Computer Software*, vol. 37, pp. 121–146, 2011.

[15] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*. Springer, 2005, pp. 237–266.

[16] B. Stevens and E. Mendelsohn, "New recursive methods for transversal covers," *Journal of Combinatorial Designs*, vol. 7, no. 3, pp. 185–203, 1999.

[17] M. B. Cohen, M. B. Dwyer, and J. Shi, "Exploiting constraint solving history to construct interaction test suites," in *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007, pp. 121–132.

[18] J. Stardom, *Metaheuristics and the Search for Covering and Packing Arrays*. Simon Fraser University Burnaby, 2001.

[19] J. Yan and J. Zhang, "A backtracking search tool for constructing combinatorial test suites," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1681–1693, 2008.

[20] J. Morgan, "Combinatorial testing: an approach to systems and software testing based on covering arrays," *Analytic Methods in Systems and Software Testing*, pp. 131–158, 2018.

[21] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "ACTS: A combinatorial test generation tool," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 2013, pp. 370–375.

[22] R. Lekivetz and J. Morgan, "Fault localization: analyzing covering arrays given prior information," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 116–121.

[23] S. S. Skiena, *The Algorithm Design Manual*. Springer, 1998, vol. 2.

[24] J. E. Hopcroft and R. M. Karp, "An nˆ5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[25] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[26] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023, https://www.gurobi.com.

[27] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013, vol. 112.