



Enabling Runtime Verification of Causal Discovery Algorithms with Automated Conditional Independence Reasoning

Pingchuan Ma
pmaab@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR

Zhenlan Ji
zjiae@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR

Peisen Yao
pyaoaa@zju.edu.cn
Zhejiang University
China

Shuai Wang*
shuaiw@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR

Kui Ren
kuiren@zju.edu.cn
Zhejiang University
China

ABSTRACT

Causal discovery is a powerful technique for identifying causal relationships among variables in data. It has been widely used in various applications in software engineering. Causal discovery extensively involves conditional independence (CI) tests. Hence, its output quality highly depends on the performance of CI tests, which can often be unreliable in practice. Moreover, privacy concerns arise when excessive CI tests are performed.

Despite the distinct nature between unreliable and excessive CI tests, this paper identifies a unified and principled approach to addressing both of them. Generally, CI statements, the outputs of CI tests, adhere to Pearl's axioms, which are a set of well-established integrity constraints on conditional independence. Hence, we can either detect erroneous CI statements if they violate Pearl's axioms or prune excessive CI statements if they are logically entailed by Pearl's axioms. Holistically, both problems boil down to reasoning about the consistency of CI statements under Pearl's axioms (referred to as CIR problem).

We propose a runtime verification tool called CICHECK, designed to harden causal discovery algorithms from reliability and privacy perspectives. CICHECK employs a sound and decidable encoding scheme that translates CIR into SMT problems. To solve the CIR problem efficiently, CICHECK introduces a four-stage decision procedure with three lightweight optimizations that actively prove or refute consistency, and only resort to costly SMT-based reasoning when necessary. Based on the decision procedure to CIR, CICHECK includes two variants: ED-CHECK and P-CHECK, which detect erroneous CI tests (to enhance reliability) and prune excessive CI tests (to enhance privacy), respectively. We evaluate CICHECK on four

real-world datasets and 100 CIR instances, showing its effectiveness in detecting erroneous CI tests and reducing excessive CI tests while retaining practical performance.

CCS CONCEPTS

• **Mathematics of computing** → **Bayesian networks**; • **Security and privacy** → **Logic and verification**; • **Software and its engineering** → *Software verification and validation*.

KEYWORDS

causal discovery, conditional independence, SMT

ACM Reference Format:

Pingchuan Ma, Zhenlan Ji, Peisen Yao, Shuai Wang, and Kui Ren. 2024. Enabling Runtime Verification of Causal Discovery Algorithms with Automated Conditional Independence Reasoning. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, April 14–20, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3597503.3623348>

1 INTRODUCTION

Causality analysis has become an increasingly popular methodology for solving a variety of software engineering problems, such as software debugging [6, 18, 19, 53], performance diagnosis [13, 54], DNN testing, repairing, and explanations [25, 45, 57]. It has also been successfully adopted in many other important domains, such as medicine [38], economics [5], and earth science [40], and data analytics [29].

Causal discovery is the core technique of causality analysis. It aims to identify causal relations among variables in data and establish a causal graph. As a result, its output constitutes the foundation of the entire causality analysis pipeline and downstream applications. Many causal discovery algorithms, such as PC [42] and FCI [56], perform a series of conditional independence (CI) tests on the data and build the causal graph using rules.¹ Thus, the output quality of causal discovery algorithms is highly dependent on the way CI tests are performed.

¹A CI is by $X \perp\!\!\!\perp Y \mid Z$, denoting that X is conditionally independent of Y given a set of conditional variables Z . A CI test is a hypothesis test that examines this property.

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0217-4/24/04...\$15.00
<https://doi.org/10.1145/3597503.3623348>

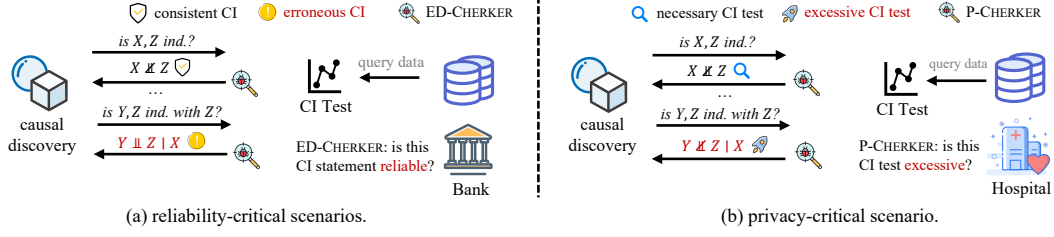


Figure 1: Two representative usage scenarios of causal discovery. “Ind.” denotes independence.

Reliability and Privacy. This research aims to enhance two key causal discovery scenarios: ① *reliability-critical* and ② *privacy-critical*. In the ① *reliability-critical* scenario (e.g., financial data processing), the focus is on the reliability of the causal discovery outputs. However, limited sample size or unsuitable statistical tests may impair CI test reliability. Unreliable CI tests may yield incorrect causal graphs, which undermines the credibility of causal discovery. To improve the reliability, we aim to ensure that all CI tests are consistent, a prerequisite for error-free causal discovery. In ② *privacy-critical* scenarios (e.g., health data processing), data privacy is typically a prime concern. In accordance to common privacy practices, access to data should be restricted unless necessary for causal discovery. Excessive CI tests may incur unnecessary data exposure and lead to a privacy concern. However, neither scenario has been thoroughly studied or addressed in the literature.

Conceptual Novelty. Our key insight is that we can address reliability and privacy concerns from CI tests in a unified way, by *reasoning about the consistency of CI statements* under the integrity constraint. We refer to this important task as Conditional Independence Reasoning (CIR). It’s important to note that CI statements within the same distribution comply with the well-established Pearl’s axioms, underpinned by statistical theories [35]. From this, we derive two key runtime verification strategies:

- In *reliability-critical* scenarios, we can detect erroneous outcomes of CI tests by checking whether the CI statements, derived from CI tests, comply with Pearl’s axioms. If not, we conclude that the CI statements contain errors and may need to abort the execution or alert users. *Example: the red CI statement is erroneous in Fig. 1(a) because its negation is entailed by existing CI statements.*
- In *privacy-critical* scenarios, we can prune excessive CI tests if their outcomes can be logically entailed by prior CI statements (under Pearl’s axioms). *Example: the red CI statement is confirmed without incurring data access in Fig. 1(b), because it can be entailed by prior CI statements.*

Technical Pipeline. It is evident that the above validation schemes can be integrated into existing causal discovery algorithms during runtime. Drawing upon the successful application of Runtime Verification (RV) tools in software security (e.g., preventing memory errors) [14, 39], we design a novel runtime verification tool, namely CICHECK, to harden the reliability and privacy of causal discovery algorithms. At the heart of CICHECK is an SMT-based reasoning system aligned with Pearl’s axioms, which facilitates consistency checking in CI statements. CICHECK comprises two variants, ED-CHECK and P-CHECK, each employing different approaches to enhance common causal discovery algorithms in accordance with *reliability-critical* and *privacy-critical* usage scenarios, respectively.

However, efficiently implementing CICHECK is challenging. First, CIR involves reasoning about first-order constraints, and while there are many off-the-shelf constraint solvers, adapting them to solve CIR problems is difficult. The challenge lies in encoding the problems soundly and compactly, while ensuring the encoded formulas are decidable. Second, the scope of all CI statements is significantly large, reaching up to $O(3^n)$ for problems involving n variables. This exponential increase in CI statements contributes to considerable overhead for SMT solving. To address the first challenge, we introduce a sound and decidable encoding scheme that translates CIR into SMT problems. To address the second challenge, we design a four-stage decision procedure as a backend for CICHECK to solve CIR problems. Crucially, we design three lightweight optimizations that actively prove or refute consistency, and only resort to the costly SMT-based reasoning for instances when these strategies don’t yield a definitive resolution.

We evaluated the ED-CHECK and P-CHECK variants of CICHECK on four real-world datasets and three synthetic datasets encompassing thousands of CI tests. The result shows the effectiveness of ED-CHECK in detecting erroneous CI tests and P-CHECK in pruning excessive CI tests and reducing privacy breaches significantly. We also evaluate 100 synthetic instances of the CIR problem and observe that our proposed optimization schemes make the expensive, nearly intractable SMT-based reasoning in CICHECK feasible in practice. Our optimizations eliminate over 90% of SMT solving procedures and substantially cut down the execution time. In summary, we present the following contributions:

- Given the prosperous success of causal discovery algorithms, we advocate validating and augmenting causal discovery algorithms with runtime verification techniques in terms of both reliability and privacy. This contribution effectively broadens the methodological horizon of formal methods while addressing emerging needs in causal inference.
- Our proposed solution, CICHECK, is rigorously established based on Pearl’s axioms. It offers a sound, complete, and decidable runtime verification framework for causal discovery algorithms. To improve performance, we present a four-stage decision procedure including SMT-based solving and three optimizations.
- We evaluate CICHECK on four real-world datasets and 100 synthetic instances of the CIR problem. The results show that CICHECK can effectively detect erroneous CI tests and reduce excessive CI tests. Moreover, the synergistic use of our optimizations makes solving the CIR problem highly practical. CICHECK is publicly available at [3].

Potential Impact. We have disseminated our findings with multiple researchers from the causality community and received positive

feedbacks. To quote one of their remarks (the developer of causal-learn [2]), “it is indeed an awesome tool and we believe that it could definitely benefit the community a lot.” This underscores our work’s prospective influence on the broader research community.

2 BACKGROUND

Notations. We denote a random variable by a capital letter, e.g., X , and denote a set of random variables by a bold capital letter, e.g., \mathbf{X} . We use \mathbf{V} to denote the set of all random variables. As a slight abuse of notation, depending on the context, we also use a capital letter X or a bold capital letter \mathbf{X} to denote a node or a set of nodes in the causal graph, respectively. We use $X \perp\!\!\!\perp Y \mid Z$ to represent a general CI relationship and use $X \perp_G Y \mid Z$ to represent d-separation (i.e., a graphical separation criterion in a graph [42]; see details in Sec. ?? of Supp. Material [31]). A statement of conditional independence/dependence is represented by lower Greek letters, e.g., $\tau := (X \perp\!\!\!\perp Y \mid Z)$ or $\rho := (X \not\perp\!\!\!\perp Y \mid Z)$. We also use dependent/independent CI statements to refer to the above two classes of statements. We use \models to denote semantical entailment and \neg to denote negation. For instance, if $\tau := (X \perp\!\!\!\perp Y \mid Z)$, then $\neg\tau \models (X \not\perp\!\!\!\perp Y \mid Z)$. We use Σ to denote a set of CI statements and Φ to denote Pearl’s axioms. And, we use KB to denote a knowledge base built on Σ and Φ . $KB \models \perp$ indicates that it is inconsistent.

2.1 CI & Pearl’s Axioms

DEFINITION 2.1 (CONDITIONAL INDEPENDENCE AND DEPENDENCE).

Let $X, Y, Z \subseteq \mathbf{V}$ be three disjoint sets of random variables from domain \mathbf{V} . We say that X and Y are conditionally independent given Z (denoted by $X \perp\!\!\!\perp Y \mid Z$) if and only if $P(Z) > 0$ and

$$P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z) \quad (1)$$

We say that X and Y are conditionally dependent given Z (denoted by $X \not\perp\!\!\!\perp Y \mid Z$) if and only if $P(Z) > 0$ and

$$P(X, Y \mid Z) \neq P(X \mid Z)P(Y \mid Z) \quad (2)$$

Empirically, statistical tests like χ^2 -squared test examine CI. Like many statistical tests, a p -value indicates the error probability when rejecting the null hypothesis. Here, the null hypothesis assumes conditional independence between X and Y given Z . Typically, the null hypothesis is rejected if $p\text{-value} < 1 - \alpha$ (α is the level of significance). A $p\text{-value} < 1 - \alpha$ rejects the null hypothesis, implying $X \not\perp\!\!\!\perp Y \mid Z$. Otherwise, $X \perp\!\!\!\perp Y \mid Z$.

Beyond CI tests on empirical distribution, efforts are made to axiomatize CI statements based on their mathematical definition. Pearl’s axioms are well-known rules that specify the integrity constraint of CI statements within a domain [34]. They can be represented as *Horn clauses* over a class of probability distributions \mathcal{P} to verify them [20]. Here, we use \mathcal{P} to denote the universal set of probability distributions of interest. In particular, we consider probability distributions over faithful Bayesian networks (a standard setup in causality analysis [42]). Intuitively, it implies the correspondence between CI in the probability distribution and the graphical structure (see the definition in Supp. Material [31]). We present Pearl’s axioms in the form of Horn clauses as follows.

Symmetry.

$$(X \perp\!\!\!\perp Y \mid Z) \iff (Y \perp\!\!\!\perp X \mid Z) \quad (3)$$

Decomposition.

$$(X \perp\!\!\!\perp Y \cup W \mid Z) \implies (X \perp\!\!\!\perp Y \mid Z) \quad (4)$$

Weak Union.

$$(X \perp\!\!\!\perp Y \cup W \mid Z) \implies (X \perp\!\!\!\perp Y \mid Z \cup W) \quad (5)$$

Contraction.

$$(X \perp\!\!\!\perp Y \mid Z) \wedge (X \perp\!\!\!\perp W \mid Z \cup Y) \implies (X \perp\!\!\!\perp Y \cup W \mid Z) \quad (6)$$

Intersection.

$$(X \perp\!\!\!\perp Y \mid Z \cup W) \wedge (X \perp\!\!\!\perp W \mid Z \cup Y) \implies (X \perp\!\!\!\perp Y \cup W \mid Z) \quad (7)$$

Composition.

$$(X \perp\!\!\!\perp Y \mid Z) \wedge (X \perp\!\!\!\perp W \mid Z) \implies (X \perp\!\!\!\perp Y \cup W \mid Z) \quad (8)$$

Weak Transitivity.

$$(X \perp\!\!\!\perp Y \mid Z) \wedge (X \perp\!\!\!\perp Y \mid Z \cup \{U\}) \implies (U \perp\!\!\!\perp Y \mid Z) \quad (9)$$

Chordality.

$$(X \perp\!\!\!\perp Y \mid \{Z, W\}) \wedge (Z \perp\!\!\!\perp W \mid \{X, Y\}) \implies (X \perp\!\!\!\perp Y \mid Z) \quad (10)$$

X, Y, Z, W, U are single variables in Eqn. 9 and Eqn. 10.

CI Axioms vs. CI Tests. The above properties are termed *axioms* as, considering certain assumptions such as faithful Bayesian networks, there are no probability distributions that contradict them. They essentially serve as integrity constraints useful in refuting inaccurate CI statements. However, they are not sufficient to deduce whether a CI statement is correct because axioms are subject to the whole class of probability distributions while the correctness of a CI statement is only concerned with a specific distribution. In other words, these CI statements can only be examined by CI tests on a particular probability distribution. In this regard, a CI test can be deemed an oracle to CI statements. Theoretically, CI tests are asymptotically accurate although they may be inaccurate with finite samples. Due to inaccuracy, the axioms may be violated. The reliability- and privacy-enhancing variants of CICHECK (ED-CHECK and P-CHECK) aim to ① check if a CI statement obtained by CI tests complies with Pearl’s axioms, and ② prune excessive CI tests that are directly entailed by CI axioms.

2.2 Constraint-based Causal Discovery

In this section, we briefly introduce the application of CICHECK, i.e., causal discovery. Causal discovery aims to learn causal relations from observational data and is a fundamental problem in causal inference. It has been widely adopted in many fields. Causal relations among data are represented in the form of a directed acyclic graph (DAG), where the nodes are constituted by all variables \mathbf{V} in data and an edge $X \rightarrow Y$ in the DAG indicates a cause-effect relation between the parent node X and the child node Y .

Constraint-based causal discovery is the most classical approach to identifying the causal graph (precisely, its Markov equivalence class) from observational data [42]. It is based on the assumptions below that affirm the equivalence between CI and d-separation.

DEFINITION 2.2 (GLOBAL MARKOV PROPERTY (GMP)). Given a distribution $P_{\mathbf{V}}$ and a DAG G , G is said to be faithful to $P_{\mathbf{V}}$ if $X \perp_G Y \mid Z \implies X \perp\!\!\!\perp Y \mid Z$.

DEFINITION 2.3 (FAITHFULNESS). Given a distribution $P_{\mathbf{V}}$ and a DAG G , $P_{\mathbf{V}}$ is said to satisfy GMP relative to G if $X \perp\!\!\!\perp Y \mid Z \implies X \perp_G Y \mid Z$.

Peter-Clark (PC) Algorithm. To practically identify the causal graph under the above assumptions, the PC algorithm is the most widely used. The *PC algorithm*, a prominent causal discovery approach [42], consists of two principal stages: (1) **skeleton identification** and (2) **edge orientation**. In the first stage, the algorithm starts from a fully connected graph, iteratively searches for separating sets (Z) that d-separate nodes, consequently removing associated edges. To expedite this process, a greedy strategy is implemented. In the second stage, edge orientations are determined according to specific criteria (see the full algorithm in Supp. Material [31]). When Def. 2.2 and Def. 2.3 hold and CI tests are correct, the PC algorithm is guaranteed to identify a genuine causal graph. From a holistic view, one can comprehend the PC algorithm as iteratively executing a series of CI tests, which aims to construct a causal graph with pre-defined rules and results of CI tests.

3 RESEARCH OVERVIEW

In this section, we present an illustrative example to show how CI-CHECK can be reduced to CIR problem solving. Then, we formulate the CIR problem and show how to connect it with CICHECK.

3.1 Illustrative Example

We present a case in accordance with Fig. 1 where ED-CHECK detects an erroneous CI statement in the PC algorithm. To ease presentation, we analyze the case from the view of ED-CHECK while the procedure also applies to P-CHECK.

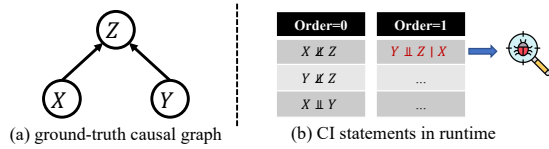


Figure 2: An erroneous CI statement occurred when launching the PC algorithm.

Consider a set of random variables $V = \{X, Y, Z\}$ whose ground-truth causal graph is shown in Fig. 2(a). The PC algorithm aims to identify the causal structure from the dataset corresponding to $V = \{X, Y, Z\}$. When $order = 0$, the algorithm enumerates all (pairwise) marginal independence relations.² As shown in the left-side table in Fig. 2(b), it identifies that $\{(X \perp Z), (Y \perp Z), (X \perp Y)\}$. Since $Y \perp Z$, when $order = 1$, the PC algorithm would further query whether $Y \perp Z | X$ holds by invoking a CI test. However, due to errors occurred in the current CI test, $\sigma = Y \perp Z | X$ is deemed true. Hence, the PC algorithm incorrectly concludes that Y and Z are non-adjacent, and yields an erroneous causal graph accordingly.

The above error is detectable under Pearl's axioms: $\neg\sigma$ can be logically entailed by considering the marginal CI statements (shown in the left-side table of Fig. 2(b)) and Pearl's axioms, which can be proved easily by contradiction. Consider the proof below:

PROOF. Suppose $Y \perp Z | X$ is true. By Contraction axiom, $Y \perp Z | X$ and $X \perp Y$ implies that $Y \perp \{X, Z\}$. Then, by Decomposition axiom, $Y \perp \{X, Z\}$ implies that $Y \perp X$ and $Y \perp Z$. This contradicts the fact that $X \not\perp Y$. \square

² $order$ is a parameter in the algorithm, denoting the condition size of current CI tests.

Therefore, σ is not warranted by Pearl's axioms and we can flag it as inconsistent.

Extension to P-CHECK. Similar to earlier case, P-CHECK preempts CI testing by concurrently validating if $Y \perp Z | X$ or $Y \not\perp Z | X$ is warranted. Given that Pearl's axioms only warrant $Y \not\perp Z | X$, we directly infer $Y \not\perp Z | X$ as the correct statement. Besides, inaccurate CI tests may undermine the effectiveness of P-CHECK. We discuss this issue in Sec. 5 and Sec. 7.

3.2 Problem Statement

The CIR Problem. The above reasoning process in the proof is ad-hoc. To deliver an automated reasoning-based runtime verification, we need a general reasoning tool to automate the above process. We define the following problem (Conditional Independence Reasoning; CIR) to formalize the above reasoning process.

CIR PROBLEM STATEMENT

Input: $\Sigma = \{\sigma_1, \dots, \sigma_k\}$, where $\sigma_1, \dots, \sigma_k$ are CI statements.

Output: Whether $KB := \{\Sigma, \Phi\} \models \perp$, where Φ is Pearl's axioms.

If the CIR problem returns a true value, we conclude that there is an inconsistency error among Σ under Pearl's axioms. Otherwise, Σ is consistent under Pearl's axioms.

Building CICHECK via CIR Reasoning. Fig. 1 illustrates two typical and representative usage scenarios of CICHECK, in the form of reliability- and privacy-enhancing variants, ED-CHECK and P-CHECK, respectively. We hereby build them, by recasting their specific objective into one (or multiple) instances of the CIR problem. In this way, the two variants of CICHECK can be implemented in a unified manner. Alg. 1 presents the algorithmic details of CICHECK.

Algorithm 1: Two variants of CICHECK

Input: Variable Pair X, Y , Condition Set Z
Output: $X \perp Y | Z$ or $X \not\perp Y | Z$

```

1 Function ED-CHECK ( $X, Y, Z$ ):
2    $\gamma \leftarrow \text{CITest}(X, Y, Z)$ ;
3   assert  $KB := \{\Sigma \cup \{\gamma\}, \Phi\} \models \perp$ ;
4   return  $\gamma$ ;
5 Function P-CHECK ( $X, Y, Z$ ):
6    $\gamma \leftarrow X \perp Y | Z$ ;
7   if  $KB := \{\Sigma \cup \{\gamma\}, \Phi\} \models \perp$  then
8     | return  $X \not\perp Y | Z$ 
9   else if  $KB := \{\Sigma \cup \{\neg\gamma\}, \Phi\} \models \perp$  then
10    | return  $X \perp Y | Z$ 
11   else
12    | return  $\text{CITest}(X, Y, Z)$ 

```

ED-CHECK initiates the CI test (line 2), obtaining the output CI statement γ . It then checks if γ is consistent under Pearl's axioms (line 3), ensuring that $KB := \Sigma \cup \gamma, \Phi \not\models \perp$. If $KB \models \perp$, ED-CHECK can either abort execution or notify the user. In this way, ED-CHECK can detect erroneous CI statements.

P-CHECK checks the consistency of both $X \perp Y | Z$ and $X \not\perp Y | Z$ under Pearl's axioms. In particular, before conducting the CI test (line 12; which accesses the private user data), P-CHECK first checks whether the incoming CI statement can be deduced from the existing CI statements (lines 7 and 9). If it is deducible, we omit the CI test and directly return the deduced CI statement (lines 8 and 10). Otherwise, we resort to the CI test (line 12). Hence, there is at most one CI test performed throughout the process.

Notably, P-CHECK reduces privacy leakage by pruning unnecessary CI tests. This is because eliminating a CI test prevents access to its associated private data. For instance, in a hospital scenario, we may deduce one CI relation among multiple diseases (e.g., $D_1 \not\perp D_2 \mid D_3$) from the existing CI statements (e.g., $D_1 \not\perp D_2$, $D_1 \perp D_3$ and $D_2 \not\perp D_3$). This eliminates the need to access the private information of $D_1 \not\perp D_2 \mid D_3$.

Challenges. While the CIR problem appears simple, solving it efficiently is far from trivial. On a theoretical level, it requires a sound, compact, and decidable encoding scheme to convert the CIR problem into an SMT problem. Practically speaking, the states of CI statements can be overwhelming, resulting in as many as $O(3^n)$ CI statements for problems involving n variables. This exponential growth can result in significant overhead for SMT solving and make solving complex CIR problems impractical. Next, we show how these technical challenges are addressed in Sec. 4.

4 AUTOMATED REASONING FOR CIR

This section presents an automated reasoning framework for solving any instances of the CIR problem (defined in Sec. 3). We first show how CIR can be formulated as an SMT problem, and present a decidable, SMT-based encoding scheme to the problem. We then present several optimizations to boost the solving of this problem, which makes it practical in real-world usage.

4.1 Encoding the CIR Problem

Formulation. Our research leverages the observation that the CIR problem, based on Pearl's axioms, can translate to a satisfiability issue. Consider an instance of the CIR problem; we expect the solver to verify whether $\langle \Sigma, \Phi \rangle := \bigwedge_{i=1}^k \sigma_i \wedge \bigwedge_{j=1}^m \Phi_j$ is satisfiable, where σ_i is a CI statement and Φ_j is a rule of Pearl's axioms. If the formula is unsatisfiable, we can conclude that the CIR instance yields a contradiction (i.e., $KB \models \perp$).

EXAMPLE 4.1. Consider the example in Fig. 2. The CIR problem is to check whether the following formula is satisfiable:

$$\{X \not\perp Z\} \wedge \{Y \not\perp Z\} \wedge \{X \perp Y\} \wedge \{Y \perp Z \mid X\} \wedge \bigwedge_{j=1}^m \Phi_j \quad (11)$$

where Φ_j represents a rule of Pearl's axioms. If the formula is unsatisfiable, then the CI statements are inconsistent under Pearl's axioms.

Despite the simplicity of the procedure above, it is non-trivial to automatically reason about the satisfiability of $\langle \Sigma, \Phi \rangle$ in a sound, decidable, and efficient manner. First, the space of all possible CI statements should be appropriately modeled, where unseen CI statements are also considered. It is noteworthy that, while addressing the consistency of CI statements, some (intermediate) CI statements do not necessarily appear in the set. We present an example in Sec. 3.1 to illustrate this. The check process first deduces an unseen CI statement (i.e., $Y \perp X, Z$). Then, it uses this previously unseen CI statement to refute the consistency. Consequently, simply ignoring unseen CI statements may lead to an incorrect result. Second, since CI statements are defined on three sets of variables (i.e., X, Y, Z), an efficient encoding of sets and their primitives is necessary. Third, we need to encode Pearl's axioms as quantified first-order formulas. We address them in what follows.

Modeling CI Statements. We first show how to model all possible CI statements as SMT formulas, including potentially unseen CI statements in Σ . We utilize the theory of Equality with Uninterpreted Functions (EUF) [4, 27] for this modeling. EUF extends the Boolean logic with equality predicates in the form of $UF(x) = y$, where UF is an uninterpreted function symbol. Notably, UF adheres to the *axiom of functional congruence*, implying identical inputs yield identical outputs (for instance, $x = y \wedge UF(x) \neq UF(y)$ is unsatisfiable). This inherent feature holds importance in CI statement encoding as it guarantees their invariance for fixed X, Y, Z . Besides, modern SMT solvers for the EUF theory have efficient enforcement for the functional congruence axiom.

Hence, the characteristics of the EUF theory make it both suitable and practical to model CI statements. An uninterpreted function symbol $CI : 2^V \times 2^V \times 2^V \rightarrow 0, 1^2$ is defined, having three arguments X, Y, Z that are V 's subsets. $CI(X, Y, Z)$ outputs a size-2 bit-vector ($b_{n-1} \dots b_0, b_i \in 0, 1$ sequence). The semantics of $CI(X, Y, Z)$'s outputs are defined next.

$$CI(X, Y, Z) := \begin{cases} 01 & \text{if } X \perp Y \mid Z \\ 00 & \text{if } X \not\perp Y \mid Z \\ 11 & \text{if } X, Y, Z \text{ are invalid} \end{cases} \quad (12)$$

Here, we create an additional state $CI(X, Y, Z) = -1$ to ensure that the uninterpreted function CI is well-defined for all possible X, Y, Z inputs. X, Y, Z are invalid in the following case:

- (1) $X \cap Y \neq \emptyset$; or $X \cap Z \neq \emptyset$; or $Y \cap Z \neq \emptyset$.
- (2) $X = \emptyset$; or $Y = \emptyset$.

Representing Sets. Nevertheless, the argument types in the definition of CI given above — sets — are not supported in many state-of-the-art SMT solvers. The Z3 solver, a widely-used SMT solver with limited set support, relies on arrays to simulate sets and is inefficient. To overcome this, we use the bit-vector theory to deliver a *compact and decidable* encoding of X, Y, Z , as well as set primitives and Pearl's axioms over these variables.

We define a function $set2vec$ that encodes each set variable into a bit-vector variable: $set2vec : 2^V \rightarrow \{0, 1\}^{|V|}$. Next, we use BV_X, BV_Y, BV_Z to represent the bit-vector variables corresponding to X, Y, Z , respectively. The bit-vector variables BV_X, BV_Y, BV_Z can constitute the arguments of the uninterpreted function CI (with a slight abuse of notation, we “override” the symbol CI (Eqn. 12) and make it to take bit-vectors as input).

With the bit-vector representation of a set, we now encode set primitives using bit-vector operations:

$$X = \emptyset \iff BV_X = 0 \quad (13)$$

$$|X| = 1 \iff \sum_{i=0}^{|V|-1} \text{extract}(i, i, BV_X) = 1 \quad (14)$$

$$\text{abbreviated to } |BV_X| = 1 \quad (15)$$

$$X \cup Y \iff BV_X \mid BV_Y \quad (16)$$

$$X \cap Y \iff BV_X \& BV_Y \quad (17)$$

$$X \subseteq Y \iff BV_X \& BV_Y = BV_X \quad (18)$$

$$X \in Y \iff |BV_X| = 1 \wedge BV_X \& BV_Y = BV_X \quad (19)$$

Next, we show an instantiation of the encoding scheme.

EXAMPLE 4.2. Consider $V = \{X, Y, Z\}$, $X = \{X\}$, $Y = \{Y\}$, and $Z = \{Y\}$. Without loss of generality, we have $BV_X = 001$, $BV_Y = 010$, $BV_Z = 100$ and $BV_0 = 000$. Then, the CI statements in Example 4.1 where $\Sigma := \{(X \not\perp Z), (Y \not\perp Z), (X \perp Y), (Y \perp Z \mid X)\}$ can be encoded as:

$$\begin{aligned} CI(BV_X, BV_Z, BV_0) &= CI(001, 100, 000) = 00 \\ CI(BV_Y, BV_Z, BV_0) &= CI(010, 100, 000) = 00 \\ CI(BV_X, BV_Y, BV_0) &= CI(001, 010, 000) = 01 \\ CI(BV_Y, BV_Z, BV_X) &= CI(010, 100, 001) = 01 \end{aligned} \quad (20)$$

As shown in Example 4.2, the CI statements are encoded as EUF constraints using bit-vector variables. Similarly, we can encode the other CI relations.

Encoding Axioms. We only apply Pearl's axioms to valid arguments of the uninterpreted function CI . Therefore, prior to encoding Pearl's axioms, we define the validity constraint to check whether the argument of CI is valid; these validity constraints form the premises of all axioms.

$$\begin{aligned} \text{Valid}(BV_X, BV_Y, BV_Z) &:= \\ BV_X \& BV_Y &= 0 \wedge BV_X \& BV_Z = 0 \\ \wedge BV_Y \& BV_Z &= 0 \wedge BV_X \neq 0 \wedge BV_Y \neq 0 \end{aligned} \quad (21)$$

Next, we encode Pearl's axioms. For each rule, we create free-variables corresponding to the variables used in this rule, and use universal quantification to specify the rule in all possible concretization to these variables.

EXAMPLE 4.3. Consider the Decomposition axiom:

$$\begin{aligned} (X \perp Y \cup W \mid Z) &\implies (X \perp Y \mid Z) \\ (X \perp Y \cup W \mid Z) &\implies (X \perp W \mid Z) \end{aligned}$$

We define the following constraint to encode the axiom (and we omit the validity constraint for ease of presentation).

$$\begin{aligned} \forall BV_X, BV_Y, BV_Z, BV_W \in \{0, 1\}^{|V|} \\ CI(BV_X, BV_Y \mid BV_W, BV_Z) = 01 &\implies CI(BV_X, BV_Y, BV_Z) = 01 \\ CI(BV_X, BV_Y \mid BV_W, BV_Z) = 01 &\implies CI(BV_X, BV_W, BV_Z) = 01 \end{aligned} \quad (22)$$

In the above constraint, we first create a universal quantifier and four free bit-vector variables BV_X, BV_Y, BV_Z, BV_W corresponding to X, Y, Z, W respectively. Then, we create two implication constraints (\implies) corresponding to the two Horn clauses of the Decomposition axiom, where the set union is encoded as bitwise-or operations.

Similarly, we can encode the remaining axioms via universal quantifiers (see the encoding in Supp. Material [31]).

PROPOSITION 4.1. The encoded SMT problem is a sound and complete representation of the CIR problem. And, the problem is decidable.

PROOF SKETCH. *Soundness & Completeness.* Let's consider an instance of CIR problem with Σ being the set of CI statements and let ϕ_c be an oracle to CIR problem. Following the encoding procedure, we can construct an SMT problem with C being the constraints and let ϕ_s be an SMT solver. The soundness of the encoding scheme implies that if $\phi_s(\mathcal{T})$ returns satisfiable, then $\phi_c(\Sigma)$ returns consistent. The completeness of the encoding scheme implies that if $\phi_c(\Sigma)$ returns consistent, then $\phi_s(\mathcal{T})$ returns satisfiable.

LEMMA 4.2. If there exists a joint distribution P that admits all CI statements in Σ , then Σ is consistent.

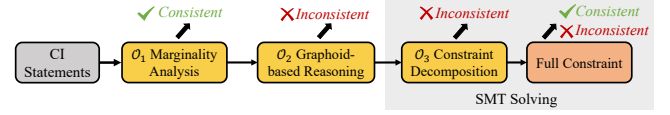


Figure 3: Overview of the CIR problem solving pipeline.

LEMMA 4.3. If there does not exist a joint distribution P that admits all CI statements in Σ , then Σ is inconsistent.

LEMMA 4.4. There does not exist a joint distribution P that simultaneously admits one CI statement and its negation.

LEMMA 4.5. Constraints only containing the Pearl's axioms are always satisfiable.

Lemma 4.2 and Lemma 4.3 are direct consequences by the definition of Pearl's axioms. Lemma 4.4 is a direct consequence of the definition of conditional independence (Eqn. 1 and Eqn. 2). Lemma 4.5 (i.e., self-consistency of Pearl's axioms) can be derived from the graphical criterion of conditional independence [35].

Assume that $\phi_s(\mathcal{T})$ is satisfiable. From this premise, it directly follows that there exists an uninterpreted function CI satisfying Pearl's axioms. We define CI as in Eqn. 12, ensuring that it is well-defined for all potential inputs X, Y, Z . Given the existence of CI , we can construct a corresponding direct acyclic graph (DAG) G in accordance to the SGS algorithm [42]. Then, based on the result of [32], there exists a joint distribution P that admits G . By Lemma 4.2, we affirm that the CI statements are consistent under Pearl's axioms when the encoded SMT problem is satisfiable. Conversely, suppose that $\phi_s(\mathcal{T})$ is unsatisfiable. By Lemma 4.5, we know that the minimum unsatisfiable core contains at least one CI statement. Without loss of generality, we assume

$$\{\sigma_{i-1}, \sigma_i, \dots, \sigma_j, \Phi_k, \dots, \Phi_l\}$$

be the minimum unsatisfiable core of \mathcal{T} . We have

$$\{\sigma_i, \dots, \sigma_j, \Phi_k, \dots, \Phi_l\} \models \neg \sigma_{i-1}$$

In the meantime, we also have σ_{i-1} in Σ . By Lemma 4.4, we know that there does not exist a joint distribution P that simultaneously admits σ_{i-1} and its negation. By Lemma 4.2, this indicates that the CI statements are inconsistent under Pearl's axioms when the encoded SMT problem is unsatisfiable. Therefore, the encoded SMT problem gives a sound and complete representation of the CIR problem, which deals with the consistency of CI statements.

Decidability. In our encoded SMT problem, we find constraints that belong to the combined theory of uninterpreted functions and quantified bit-vectors. According to Wintersteiger et al. [51], it is known that this combined theory is decidable. This ensures the decidability of the constraints in our problem, hence confirming the decidability of the entire encoded SMT problem. \square

4.2 Solving the CIR Problem

We describe the encoding of the CIR problem in Sec. 4.1. Despite the rigorously formed encoding, this approach is *NEXP-complete* [51], meaning that it may become intractable for complex problem instances. In this section, we present several optimizations and design choices to make CICHECK practical at a moderate cost.

Fig. 3 outlines the workflow of our solving procedure. In short, before launching the SMT solver over the full constraint obtained by our encoding scheme, we leverage the following three lightweight

optimizations to prove or refute the satisfiability of the constraint and presumably “early-stop” the reasoning to avoid costly SMT solving. Particularly, Marginality Analysis is performed first and can prove the satisfiability of the constraint. If Marginal Analysis fails, we then apply the following two optimizations (Graphoid-based Reasoning and Constraint Decomposition) to refute the satisfiability. SMT solving over the full constraint is only performed when none of the optimizations reaches a conclusive result.

4.2.1 O_1 Marginality Analysis. In general, we need to check every CI test to ensure that each one is valid (or can be pruned), depending on the application. However, we notice that when Σ only contains non-degenerate marginal CI statements, check is *not* necessary. The following theorem characterizes this property:

THEOREM 4.6. *Let Σ be a set of non-degenerate marginal CI statements. Here, Σ is said to be non-degenerate if Σ does not contain both a marginal CI statement and its negation. A marginal CI statement is a CI statement with an empty condition set, such as $X \perp\!\!\!\perp Y$. Then, Σ is always consistent under Pearl’s axioms.*

PROOF SKETCH. By definition, Σ is consistent under Pearl’s axioms if there exists a joint distribution P such that P admits every CI statement in Σ . Below, we demonstrate that such a joint distribution P always exists for arbitrary Σ , as long as it only contains non-degenerate marginal CI statements. Let G_u be an undirected graph where each node in G_u represents a random variable with respect to P . Two nodes X, Y are connected by an edge if and only if $X \not\perp\!\!\!\perp Y$. According to the “acyclic orientation” in [43], any undirected graph can be converted to a directed acyclic graph (DAG). Let G_d be a DAG of G_u . Then, according to Theorem 5 of [32], a faithful multinomial P exists for any DAG G_d .³ In this regard, Σ is at least warranted in P and is thus consistent. \square

Theorem 4.6 directly implies we can omit CICHECK, when Σ exclusively holds non-degenerate marginal CI statements. Often, this occurs in the initial phases of common causal discovery algorithms, shown in the example below.

EXAMPLE 4.4. *Consider the first three CI statements in Example 4.1, with $\Sigma := (X \not\perp\!\!\!\perp Z), (Y \not\perp\!\!\!\perp Z), (X \perp\!\!\!\perp Y)$. As Σ consists solely of non-degenerate marginal CI statements, according to Theorem 4.6, we can directly infer that Σ is consistent.*

4.2.2 O_2 Graphoid-based Reasoning. In causal inference, CI statements are often represented as graphoids [35], which are special graphical data structures that enable efficient reasoning about ① *whether a CI statement is always independent*. As a result, graphoid-based reasoning cannot distinguish “② a CI statement is always dependent” from “③ a CI statement can be either independent or dependent”. Despite this limitation, graphoid-based reasoning is useful in our context owing to its efficiency. In practice, we find graphoid-based reasoning to be significantly faster than solving standard CIR problems. Below, we show the translation from an instance of the CIR problem to a sequence of graphoid-based reasoning.

Let the knowledge base $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ be the input of a CIR problem. We split Σ into two parts: (1) the first part, Σ_1 , contains all

dependent CI statements (e.g., $X \not\perp\!\!\!\perp Y \mid Z$), and (2) the second part, Σ_2 , contains all independent CI statements (e.g., $X \perp\!\!\!\perp Y \mid Z$). We can then use graphoid-based reasoning to check if Σ_2 implies the negation of any instances in Σ_1 (e.g., $\Sigma_2 \implies \neg\sigma_i$ where $\sigma_i \in \Sigma_1$). If this occurs, we can conclude that Σ is inconsistent.

EXAMPLE 4.5. *Consider the CI statements in Example 4.1, with $\Sigma := \{(X \not\perp\!\!\!\perp Z), (Y \not\perp\!\!\!\perp Z), (X \perp\!\!\!\perp Y), (Y \perp\!\!\!\perp Z \mid X)\}$. Initially, to evaluate Σ ’s consistency, we partition it into $\Sigma_1 = (X \not\perp\!\!\!\perp Z), (Y \not\perp\!\!\!\perp Z)$ and $\Sigma_2 = (X \perp\!\!\!\perp Y), (Y \perp\!\!\!\perp Z \mid X)$. Then, using graphoid-based reasoning, we verify if Σ_2 suggests either $X \perp\!\!\!\perp Z$ or $Y \perp\!\!\!\perp Z$. If Σ_2 infers any of these two statements, Σ is deemed inconsistent.*

To boost the standard SMT constraint-solving process, one may sample a few concrete inputs to rapidly disprove the satisfiability of a constraint [12]. Similarly, graphoid-based reasoning serves as a lightweight testing-based error detecting scheme. With this optimization, we may avoid costly SMT solving without compromising the completeness of the overall CIR solving procedure.

4.2.3 O_3 Constraint Decomposition. While the graphoid-based reasoning is efficient, it cannot sufficiently leverage the dependent CI statements in the knowledge base. Hence, an SMT solver is still required, leading to a significant overhead. To address this issue, we propose applying constraint decomposition to the CIR problem using domain knowledge. This method segments the full constraint set into smaller subsets, resolved independently and concurrently.

The key insight is that inconsistencies among CI statements frequently arise from two sources: ① individual axioms and ② a small, closely related subset of CI statements. Technically, given an incoming CI statement γ , we check whether it is consistent with a particular axiom and a small subset Σ^* of Σ . To identify this subset of CI statements, we examine the overlap between γ and each CI statement in Σ , selecting only those with non-empty overlaps. In this context, we define the overlap between γ and σ . Without loss of generality, let γ be $X \perp\!\!\!\perp Y \mid Z$ and σ be $X' \perp\!\!\!\perp Y' \mid Z'$. Then, the overlap between γ and σ is defined as the intersection: $U = (X \cup Y \cup Z) \cap (X' \cup Y' \cup Z')$. If U is non-empty, we deem σ and γ have non-empty overlaps.

We can break down the full constraint set into several smaller sets and resolve them concurrently along with the full constraint. If any smaller set is inconsistent, it infers the full constraint’s inconsistency, allowing an immediate result return without waiting for complete constraint resolution. If not, we can securely return the outcome derived from the solution of the full constraint. In essence, O_3 concentrates on maximizing modern CPUs’ parallelism and interrupting constraint solving early when feasible.

4.2.4 The Overall Decision Procedure. Our decision procedure for the CIR problem combines three optimizations (Sec. 4.2.1–Sec. 4.2.3): and an off-the-shelf SMT solver, as illustrated in Figure 3. By integrating the three optimizations, the four-stage procedure enhances the efficiency and scalability of the CIR solving process, making it suitable for real-world usages.

Soundness and Completeness. Although we make certain levels of over- or under-approximations in our three optimizations, we only take the conclusive results from these optimizations and resort to the SMT solver in the absence of a conclusive result. Our encoding of the CIR problem is faithful and does not make any

³Precisely, Theorem 5 in [32] implies that there is almost always a P that is faithful to G_d in a measure-theoretic sense. The Lebesgue measure (conceptually similar to probability) of the set of unfaithful distributions P is zero.

approximations. The functionality of de facto SMT solvers is guaranteed by the SMT-LIB standard [8] and their implementations are well-tested [50]. Hence, it should be accurate to assume that modern SMT solvers (e.g., Z3 [16] adopted in this research) will return a correct result. Hence, the overall procedure is sound and complete with respect to Pearl’s axioms.

5 IMPLEMENTATION OF CICHECK

Based on the CIR problem, we implement two checkers: ED-CHECK and P-CHECK, as depicted in Alg. 1. We integrate CICHECK with the standard PC algorithm, one highly popular causal discovery algorithm. However, CICHECK is independent of the causal discovery algorithm and can be integrated with any causal discovery algorithm as long as it uses CI tests (see Sec. 7 on extension). We implement CICHECK in Python with approximately 2,000 lines of code. Our implementation is optimized for multi-core parallelism. We use Z3 [16] as the SMT solver. Our implementation can be found at [3]. Next, we describe two design choices in our implementation. **Handling Inconsistent Knowledge Bases in P-CHECK.** Unreliable CI tests may introduce errors into the knowledge base, and these errors may render the knowledge base inconsistent (i.e., $KB \models \perp$). In the context of ED-CHECK, we can simply abort the causal discovery process. However, in P-CHECK, it is desirable to continue the causal discovery process at the best of efforts even with an inconsistent knowledge base. However, an inconsistent knowledge base may cause our solving pipeline to fail to determine the consistency of an incoming CI statement. To address this issue, we backtrack to the last consistent knowledge base and continue solving to the best of our ability. If inconsistencies occur frequently, after reaching a threshold (ten in our evaluation; see Sec. 6), we revert to using standard CI tests for all CI statement queries.

Handling Timeout. Time constraints could cause the SMT solver to yield an `unknown` when trying to determine the satisfiability of a constraint. As a common tactic in program analysis, `unknown` is interpreted as “satisfiable.” This tactic could cause false negatives, wherein ED-CHECK overlooks some incorrect CI statements and P-CHECK misses some pruning opportunities. Despite these, CICHECK remains complete because it spots wrong or excessive CI statements based solely on unsatisfiable outcomes.

Application Scope of ED-CHECK. Certain errors in causal discovery, which do not contradict Pearl’s axioms, may be inherently undetectable. While adhering to Pearl’s axioms is crucial, it is not the sole determinant of reliability. Holistically, these errors can arise from two factors — the partial nature of Pearl’s axioms and the flexibility of the probability distribution.

In the former case, the completeness of Pearl’s axioms was traditionally assumed correct, but Studeny [44] disproved this by showing that some CI statements complying with Pearl’s axioms are not entailed by any probability distribution. However, we argue that these CI statements pose minimal practical concerns due to their rarity. In the latter case, the versatility of the probability distribution presents a critical limitation to the CIR problem. For example, in a dual-variable system (X, Y) , both $X \perp\!\!\!\perp Y$ and $X \not\perp\!\!\!\perp Y$ conform to Pearl’s axioms. In this scenario, ED-CHECK fails to spot the error as these incorrect CI statements could perfectly align with another valid probability distribution. Analogous to this, the control-flow

integrity (CFI) checker cannot detect all control-flow hijacking attacks [11, 15] when these attacks follow a valid control-flow path. Nevertheless, we conjecture that, in normal usages of ED-CHECK (excluding adversarial use), such cases are not likely to happen.

Therefore, we expect ED-CHECK to effectively uncover the majority of errors in practical scenarios. Since Pearl’s axioms serve as a stringent integrity constraint, erroneous CI statements, even undetectable immediately, likely breach Pearl’s axioms eventually, especially when Σ is non-trivially large. As will be shown in Sec. 6.2 and also in Supp. Material [31], ED-CHECK is highly sensitive in detecting subtle errors in practice.

6 EVALUATION

In this section, we study the effectiveness of our framework by answering the following research questions:

- RQ1** Does ED-CHECK achieve high error detectability on erroneous CI statements?
- RQ2** Does P-CHECK effectively prune excessive CI tests in the privacy-critical setting?
- RQ3** How effective are our optimizations?

6.1 Evaluation Setup

Table 1: Datasets.

Dataset	#Nodes	#Edges	Max Degree	Max Indegree
Earthquake	5	4	4	2
Cancer	5	4	4	2
Survey	6	6	4	2
Sachs	11	17	7	3

We use four popular Bayesian networks from the bnlearn repository [1] and list their details in Table 1. They come from diverse fields and some of them even contain highly skewed data distributions which are very challenging for CI tests (e.g., the “Earthquake” dataset). Overall, we clarify that the evaluated datasets are practical and challenging, manifesting comparable complexity to recent works with a similar focus [22, 55]. We illustrate the complexity of CIR problem instances involved in the benchmark in Supp. Material [31]. Besides, we further follow [52] to evaluate P-CHECK (in RQ2) with three synthetic datasets ranging from five to 15 variables. The preparation procedure is detailed in Supp. Material [31]. In RQ1, we assess ED-CHECK by first using a perfect CI oracle that returns the ground-truth CI statement. Then, we randomly inject errors with different scales into the CI statements and assess the effectiveness of ED-CHECK in error detection. In RQ2, we explore the use of P-CHECK in two imperfect CI test schemes in addition to the perfect CI oracle, where the χ^2 test is for normal causal discovery and Kendall’s τ test [48] is for differentially private causal discovery. To handle inconsistent knowledge bases (noted in Sec. 5), we resort to the standard CI tests when inconsistencies occur over ten times. All experiments are conducted on a server with an AMD Threadripper 3970X CPU and 256 GB memory.

6.2 Effectiveness of ED-CHECK

In line with prior research [14], we utilize a collection of manually-crafted erroneous cases to assess whether ED-CHECK can detect them. Given a dataset from Table 1, we initially integrate the causal discovery process with ED-CHECK without injecting additional

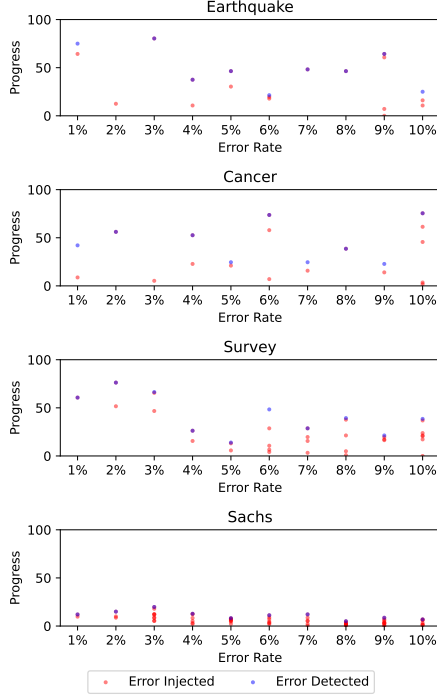


Figure 4: ED-CHECK on four datasets.

Table 2: Sensitivity of error detection.

Dataset	Earthquake	Cancer	Survey	Sachs
Sensitivity	15.1%	25.0%	21.5%	8.0%

errors, verifying that no false alarms arise from ED-CHECK. Let K represents the total number of CI tests conducted on this dataset and $r\%$ an error injection rate. We then randomly select $k = K \times r\%$ indices from $1, \dots, k$ (e_1, \dots, e_k) and inject errors by flipping the outcome of the e_i -th CI test for each $e_i \in \{e_1, \dots, e_k\}$. It is important to note that ED-CHECK is not aware of specific error injection locations and validates errors after every CI test. The process is repeated for $r \in \{1, \dots, 10\}$, ensuring at least one error per run.

We present the error detection results in Fig. 4. Red dots represent the normalized position (by the total number of CI tests) where errors are injected, and blue dots indicate where ED-CHECK detects them. Overlapping red and blue dots indicate immediate error detection upon injection. We observe that ED-CHECK successfully detects errors in most cases (38 out of 40), excluding “2%” of Earthquake and “3%” of Cancer. Error detection sensitivity is also quantified in Table 2, calculated by subtracting the normalized progress of the first error detection from that of the error injection. For instance, sensitivity in Fig. 4 is determined by the distance between the first red and blue dot in each plot. Overall, ED-CHECK exhibits high sensitivity to errors, with six detected immediately and 25 within 15% of the total CI tests.

Failure Case Analysis. We also inspected the two failure cases in Fig. 4. In both cases, errors are injected into a marginal CI statement. As aforementioned in Sec. 4.2.1, marginal CI statements themselves would not violate Pearl’s axioms. If there is no other high-order CI statements that trigger the inconsistency, ED-CHECK cannot detect the error by design. Unfortunately, the injected errors do not violate

Table 3: Performance with the perfect CI Oracle.

Dataset	#CI Tests w/o P-CHECK	#CI Tests w/ P-CHECK
Earthquake	56	22 (-60.7%)
Cancer	57	22 (-61.4%)
Survey	115	51 (-55.7%)
Sachs	1258	818 (-35.0%)

any high-order CI statements. Indeed, despite the error, we find that the learned causal skeleton is identical to the ground truth. Hence, we presume these errors may be too subtle to be detected.

Answer to RQ1: When benchmarked using common causal discovery datasets with sufficient complexity, ED-CHECK uncovers erroneous CI statements with high accuracy and also high sensitivity.

6.3 Effectiveness of P-CHECK

In this evaluation, we employ the perfect CI oracle as well as two distinct CI test schemes to evaluate the performance of P-CHECK in reducing the number of CI tests produced by the PC algorithm. The first CI test scheme is the well-established χ^2 test [36], a statistical hypothesis test that assesses the independence of two categorical variables. This test is widely used in causal discovery and often serves as the default CI test scheme in many causal discovery tools. Hence, the result of the χ^2 test provides a baseline for assessing the effectiveness of P-CHECK in realistic settings. The second scheme is the differentially private Kendall’s τ test [48], which is designed to protect the privacy of the dataset by introducing controlled noise into the test statistics. This test is particularly useful for evaluating the performance of P-CHECK in privacy-sensitive applications, where the traditional χ^2 test may not be suitable due to the disclosure risks associated with its non-private nature. By incorporating these two CI test schemes, we aim to demonstrate P-CHECK’s applicability in handling diverse CI test schemes.

P-CHECK with Perfect CI Oracle. We present the results of our experiments with the perfect CI oracle on four datasets in Table 3. We compared the number of CI tests generated by the PC algorithm with and without P-CHECK using a perfect CI oracle. The results are shown in Table 3, where we observe that P-CHECK is highly successful in decreasing the number of required CI tests across all datasets. Specifically, for datasets with five or six variables, namely Earthquake, Cancer, and Survey, we achieve a reduction ranging from 55.7% to 60.7%. Even on the challenging Sachs dataset with 11 variables, where the UF to a CI statement can have up to 2^{33} states, P-CHECK demonstrates a significant reduction of 35.0%.

As discussed in Sec. 4.2.4, the CIR problem solving is sound and complete, which means that P-CHECK does not yield any erroneous CI statements. Indeed, we observe that the learned causal skeleton is identical to the ground-truth skeleton when using the perfect CI oracle, with no additional errors introduced.

P-CHECK with Imperfect CI Tests. The efficacy of P-CHECK is validated on two imperfect CI test schemes: DP Kendall’s τ and χ^2 Tests. The results are in Table 4. SHD represents the “edit distance” from the learned graph to the ground truth; a lower value is preferable. Total privacy for executing DP Kendall’s τ tests is denoted by ϵ ; a lower value is preferable. Owing to the probabilistic character of the DP-enforced algorithm, the DP Kendall’s τ Test is repeated ten times and the average outcome is reported. P-CHECK efficiently reduces the necessary CI tests count while maintaining competitive SHD accuracy across all data sets. On average, P-CHECK results

Table 4: Effectiveness of P-CHECK in terms of #CI Test, Structural Hamming Distance (SHD) and privacy budget (ϵ). “ER- n ” denotes the synthetic dataset with n variables and “ER-15” denotes the largest synthetic dataset with 15 variables.

Metric	Earthquake	Cancer	Survey	Sachs	ER-5	ER-10	ER-15
DP Kendall’s τ Test (averaged on ten runs)							
#CI Test	37.7→16.2	16.8→10.7	45.9→16.1	264.3→156.2	13.3→10.0	86.7→67.0	190.0→153.3
SHD	2.7→2.6	2.8→2.6	2.5→0.7	13.9→13.0	1.7→2.0	7.3→7.0	20.3→18.7
ϵ	54.6→34.3	38.3→32.1	60.2→40.8	277.9→208.5	35.2→29.9	141.7→118.8	280.6→238.4
χ^2 Test							
#CI Test	56→22	75→36	69→30	1277→816	15→11	208→136	610→475
SHD	0→0	2→2	0→0	1→1	0→0	1→1	4→4

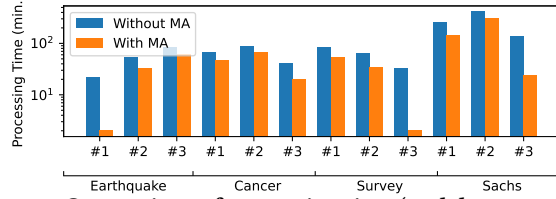


Figure 5: Comparison of processing time (each bar represents an individual run).

in a reduction of 38.0% and 41.2% in CI tests using DP Kendall’s τ and χ^2 Tests respectively, across all datasets. Moreover, there is an observable slight increase in accuracy (lower SHD) under the DP setting compared to Priv-PC [48]. This is presumably because P-CHECK sidesteps the high number of CI tests that are typically incorrect. In addition, P-CHECK offers superior privacy protection, as indicated by a substantially lower ϵ .

Answer to RQ2: P-CHECK effectively prunes excessive CI tests in both perfect and imperfect CI test schemes. It reduces CI tests by 35.0% to 60.7% with the perfect CI oracle and retains accuracy while providing better privacy protection in imperfect CI tests.

6.4 Ablation Study

In RQ3, we aim to evaluate the effectiveness of our proposed optimizations and explore their impact on the overall performance. Recall that there are three optimizations in CICHECK, namely O_1 MA, O_2 GR, and O_3 CD. In particular, O_1 is used to prove consistency while O_2 and O_3 are used to refute consistency. Thus, given their different purposes, we employ *distinct evaluation strategies* for O_1 and the other two optimizations. We measure O_1 ’s effectiveness through the entire lifecycle of the causal discovery. On the other hand, O_2 , O_3 are particularly effective when dealing with inconsistent CI statements. In ED-CHECK, there is at most one inconsistent case (since we abort execution once an inconsistency is detected). Hence, we measure the effectiveness of O_2 , O_3 on multiple instances of the CIR problem. Below, we present the impact of them.

Effectiveness of O_1 MA. The effectiveness of O_1 is first evaluated through a comparison of processing times between ED-CHECK with and without O_1 . Our experiment involves injecting 5% errors into CI statements. The random seed remains fixed per comparison group. Each dataset undergoes three comparative groupings. As shown in Fig. 5, O_1 notably reduces the time. Specifically, CICHECK immediately flags the error in the initial non-marginal CI statements in the first case of Earthquake and the third case of Survey. As a result, in such situations, O_1 makes CICHECK almost effortless. On average, O_1 achieves a speedup of 48.9% ($\pm 25.7\%$) across all datasets,

Table 5: Effectiveness of O_2 GR and O_3 CD. “avg.”, “mdn.” and “max.” denotes the mean, median and maximum.

	Refutability	Processing time (avg. / mdn. / max.)
O_2	91/100	0.6 / 0.6 / 0.7
O_3 w/o Full	53/100	14.3 / 10.9 / 25.8
O_3	100/100	25.6 / 5.3 / 58.2
$O_2 + O_3$	100/100	2.4 / 0.6 / 53.8

with a maximum reduction of 93.8%. Our interpretation is that O_1 effectively accelerates CICHECK without compromising accuracy.

Effectiveness of O_2 GR and O_3 CD. We then evaluate the effectiveness of O_2 and O_3 . To fully unleash the power of O_2 and O_3 , we use the Sachs dataset, which is the largest dataset in our experiments. On the remaining datasets, we observe similar performance boosts from O_2 and O_3 (see Supp. Material [31]). In particular, we collect a total of 1,258 CI statements from the error-free execution on the Sachs dataset. Then, we randomly flip some CI statements (as well as their symmetric counterparts) to create a knowledge base such that each knowledge base contains roughly 5% erroneous CI statements. Hence, we obtain one instance of the CIR problem with erroneous CI statements that may be inconsistent with the Pearl’s axioms. We check whether the inconsistent knowledge bases can be refuted by O_2 or O_3 , respectively. We also measure the processing time. We repeat the experiments 100 times and report the average running time and the overall refutability.

As shown in Table 5, in a series of 100 experiments, O_2 has an efficacy of refuting 91% of the inconsistent knowledge bases, and O_3 — when combined with full constraint solving — manages to refute 100% of the inconsistencies, while the decomposed constraints alone can only address 53% of the errors. When both O_2 and O_3 are applied together, they detect all errors, enhancing their collective effectiveness. We notice that the median processing time of O_2 is significantly lower (0.6 seconds) as compared to that of using decomposed constraints alone, which requires 10.9 seconds, and extends to 5.3 seconds when used alongside the full constraint, marking O_2 as both efficient and highly sensitive towards inconsistent knowledge bases. Besides, we observe notable variations in the mean, median, and maximum processing time for $O_2 + O_3$. We interpret the result as follows. The processing time primarily clusters around two scenarios; the majority approximating 0.6 seconds (case 1), with the remainder around 53.8 seconds (case 2). The first case corresponds to situations where O_2 , a lightweight scheme, swiftly refutes consistency. The latter case occurs when an SMT solver is involved and requires more time to resolve the CIR problem.

On the other hand, O_3 provides a comprehensive overview of the CIR problem and detects all errors, serving as an effective complement to O_2 by refuting additional inconsistent knowledge bases. From another perspective, O_2 significantly reduces the costs associated with O_3 by detecting numerous errors early on. Simultaneous adoption of O_2 and O_3 trims down the processing time to 2.3 seconds, yielding a solution that is more accurate than using O_2 alone, and remarkably faster than just utilizing O_3 .

Answer to RQ3: All three optimizations significantly contribute to the performance of CICHECK. Specifically, O_1 reduces the processing time by a significant margin, while O_2 and O_3 are effective and efficient in detecting and refuting inconsistent knowledge bases. The synergistic effect of these optimizations makes CICHECK practical.

7 DISCUSSION

Potential Impact in Software Engineering. We articulate the potential impact of our work in the SE community from two perspectives. ❶ *Causal Inference in SE*: Causal inference is gaining traction as an important methodology within SE [18, 19, 21, 23, 23–26, 41, 45, 57]. The prosperous use of causal inference algorithms makes CICHECK an important contribution that uniquely addresses both privacy and reliability concerns. This positions CICHECK as a useful tool in the fast-evolving SE landscape. ❷ *Methodological Integration*: CICHECK delivers a novel adoption of classical SE techniques, e.g., formal verification and constraint solving. This synergy, to a degree, enables CICHECK to expand the existing formal method realm into a novel and crucial domain – causal discovery.

Scalability. Automated reasoning is often costly, and the scalability of our framework is on par with that of other SMT/SAT-based causal discovery algorithms [22, 55] (e.g., causal graph with roughly a dozen of nodes). Causal discovery algorithms often struggle with high-dimensional data, which is why ED-CHECK focuses on datasets with moderate complexity; de facto causal discovery algorithms are anticipated to behave reliably and perform well over these datasets. In terms of P-CHECK, existing constraint-based privacy-preserving causal discovery algorithms, such as EM-PC [52], share a similar scalability regime. However, Priv-PC [48] shows a better scalability which handles larger datasets like ALARM and INSURANCE [1]. Notably, CICHECK has a unique role compared to these algorithms, potentially enhancing their privacy-preserving abilities, evidenced by P-CHECK’s improvement on Priv-PC (Sec. 6.3). Looking ahead, we plan to boost CICHECK with advanced SMT solvers like CVC5 [7].

Extension to Other Causal Discovery Algorithms. Since CICHECK does not rely on any execution information about the underlying algorithms, it is agnostic to specific causal discovery implementations. Hence, our framework can be smoothly extended to other causal discovery algorithms by replacing the employed standard CI test algorithm with the one hardened by CICHECK. For instance, we envision that CICHECK can be applied to enhance the federated causal discovery algorithm [49] to reduce privacy leakage. However, CICHECK is not applicable to enhance causal discovery algorithms that do not use CI tests (e.g., functional causal models [30, 37] or score-based methods [47]) or leverage quantitative information beyond qualitative CI relations (e.g., p-values [17, 28]).

Threats to Validity. Our framework is implemented using standard tool chains and the experiments are conducted with widely-used datasets and CI test algorithms. Nonetheless, potential threats to the validity of our results persist. Firstly, our framework’s performance may be contingent on the choice of CI test algorithms and datasets. To alleviate this risk, we have tested our framework with multiple CI test algorithms and datasets. Secondly, our framework’s scalability may face limitations when employed on large-scale datasets or complex causal structures. To address this, we propose several optimizations to improve efficiency. Thirdly, the generalizability of our results may be influenced by the specific implementation choices within our framework. To mitigate this threat, we conducted experiments on several datasets and CI test algorithms repetitively to ensure the robustness of our results. Besides, using random error injection in Sec. 6.2 may not perfectly simulate real-world cases. In this regard, our findings’ applicability

might be limited. Integrating CICHECK with causal learning tools [2] would be a beneficial way to confirm the validity. As mentioned in Sec. 5, it is theoretically conceivable that multiple incorrect facts could weaken the effectiveness of ED-CHECK. While we deem this scenario rare due to the tight integrity constraints by Pearl’s axioms and empirically show its unlikelihood in practice (Sec. 6.3 and Supp. Material [31]), we acknowledge it as a threat to validity. Statistical errors could cause P-CHECK to malfunction because of inconsistent knowledge bases or faulty reasoning (i.e., deriving an incorrect CI from another wrong CI). We address inconsistencies in the knowledge bases by employing a backtracking mechanism to restore previous states (Sec. 5). We demonstrate empirically that P-CHECK does not compromise causal discovery accuracy, proving its resilience to statistical errors (Sec. 6.3).

8 RELATED WORK

We have reviewed the related works in causal discovery in Sec. 2. In this section, we discuss the related work in formal methods for CI implication and causal inference in software engineering.

CI Implication. Some studies aim to infer CI statements through implications. Instead of using Pearl’s axioms, they attempt to characterize specific classes of distribution and infer CI statements using algebraic or geometric reasoning methods [9, 10, 33, 46]. It is worth noting that these approaches are not applicable in our context. Firstly, they cannot reason over conditional dependence – a prevalent and essential inference case in our scenario. Secondly, they often exhibit limited scalability and narrow applicability (e.g., the racing-based method [10] can only handle five variables).

Causality in Software Engineering. Causality has become an increasingly prevalent technique in software engineering [18, 19, 21, 23, 23, 24, 26, 41]. These tools typically concentrate on examining the effects of program inputs or configurations on software behavior, such as execution time or crashes. Moreover, due to its inherent interpretability, causality analysis is extensively utilized in the testing and repair of machine learning models [25, 45, 57].

9 CONCLUSION

We have presented CICHECK, a novel runtime verification tool for causal discovery algorithms that addresses both reliability and privacy concerns in *reliability-critical* and *privacy-critical* usage scenarios in a unified approach. Experiments on various synthetic and real-world datasets show that CICHECK enhances both reliability-critical and privacy-critical scenarios with a moderate overhead. We envision our work can inspire more research on the extension of SE methodology to the domain of causal inference.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments. The authors would also like to thank Rui Ding, Haoyue Dai, Yujia Zheng and Joseph Ramsey for their helpful discussions. This work is supported in part by National Key R&D Program of China under Grant No. 2022YFB4501903, National Natural Science Foundation of China under Grant No. 62302434, the Qizhen Scholar Foundation of Zhejiang University and the HKUST 30 for 30 research initiative scheme under the the contract Z1283.

REFERENCES

- [1] 2023. bnlearn. <https://www.bnlearn.com/bnrepository>.
- [2] 2023. causal-learn. <https://github.com/py-why/causal-learn>.
- [3] 2023. Source code and data. <https://anonymous.4open.science/t/CISan>.
- [4] Wilhelm Ackermann. 1954. Solvable cases of the decision problem. (1954).
- [5] Peter Martey Addo, Christelle Manibialoa, and Florent McIsaac. 2021. Exploring nonlinearity on the CO₂ emissions, economic production and energy use nexus: A causal discovery approach. *Energy Reports* 7 (2021), 6196–6204.
- [6] Mona Attariyan and Jason Flinn. 2008. Using Causality to Diagnose Configuration Bugs.. In *USENIX Annual Technical Conference*. 281–286.
- [7] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. 2022. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*. Springer, 415–442.
- [8] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. 2010. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, Vol. 13. 14.
- [9] Remco Bouckaert, Raymond Hemmecke, Silvia Lindner, and Milan Studeny. 2010. Efficient Algorithms for Conditional Independence Inference. *Journal of Machine Learning Research* 11 (2010), 3453–3479.
- [10] Remco R Bouckaert and Milan Studeny. 2007. Racing algorithms for conditional independence inference. *International Journal of Approximate Reasoning* 45, 2 (2007), 386–401.
- [11] Nicholas Carlini, Antonio Barresi, Mathias Payer, David Wagner, and Thomas R Gross. 2015. {Control-Flow} bending: On the effectiveness of {Control-Flow} integrity. In *24th USENIX Security Symposium (USENIX Security 15)*. 161–176.
- [12] Mahinthan Chandramohan, Yinxing Xue, Zhengzi Xu, Yang Liu, Chia Yuan Cho, and Hee Beng Kuan Tan. 2016. Bingo: Cross-architecture cross-os binary search. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 678–689.
- [13] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1887–1895.
- [14] Zhe Chen, Zheming Wang, Yunlong Zhu, Hongwei Xi, and Zhibin Yang. 2016. Parametric runtime verification of C programs. In *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings 22*. Springer, 299–315.
- [15] Mauro Conti, Stephen Crane, Lucas Davi, Michael Franz, Per Larsen, Marco Negro, Christopher Liebchen, Mohaned Qunaibit, and Ahmad-Reza Sadeghi. 2015. Losing control: On the effectiveness of control-flow integrity under stack attacks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 952–963.
- [16] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29–April 6, 2008, Proceedings 14*. Springer, 337–340.
- [17] Rui Ding, Yanzhi Liu, Jingjing Tian, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2020. Reliable and efficient anytime skeleton learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 10101–10109.
- [18] Clemens Dubsiaff, Kallistos Weis, Christel Baier, and Sven Apel. 2022. Causality in configurable software systems. In *Proceedings of the 44th International Conference on Software Engineering*. 325–337.
- [19] Anna Fariha, Suman Nath, and Alexandra Meliou. 2020. Causality-guided adaptive interventional debugging. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 431–446.
- [20] Dan Geiger and Judea Pearl. 1993. Logical and algorithmic properties of conditional independence and graphical models. *The annals of statistics* 21, 4 (1993), 2001–2021.
- [21] Chun-Hung Hsiao, Satish Narayanasamy, Essam Muhammad Idris Khan, Cristiano L Pereira, and Gilles A Pokam. 2017. Asynclock: Scalable inference of asynchronous event causality. *ACM SIGPLAN Notices* 52, 4 (2017), 193–205.
- [22] Antti Hyttinen, Patrik Hoyer, Frederick Ederhardt, and Matti Järvisalo. 2013. Discovering Cyclic Causal Models with Latent Variables: A General SAT-Based Procedure. In *Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 301–310.
- [23] Zhenlan Ji, Pingchuan Ma, and Shuai Wang. 2022. PerfCE: Performance Debugging on Databases with Chaos Engineering-Enhanced Causality Analysis. *arXiv preprint arXiv:2207.08369* (2022).
- [24] Zhenlan Ji, Pingchuan Ma, Shuai Wang, and Yanhui Li. 2023. Causality-Aided Trade-off Analysis for Machine Learning Fairness. *arXiv preprint arXiv:2305.13057* (2023).
- [25] Zhenlan Ji, Pingchuan Ma, Yuanyuan Yuan, and Shuai Wang. 2023. CC: Causality-Aware Coverage Criterion for Deep Neural Networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1788–1800.
- [26] Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2020. Causal testing: understanding defects’ root causes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 87–99.
- [27] Daniel Kroening and Ofer Strichman. 2016. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Springer. <https://doi.org/10.1007/978-3-662-50497-0>
- [28] Pingchuan Ma, Rui Ding, Haoyue Dai, Yuanyuan Jiang, Shuai Wang, Shi Han, and Dongmei Zhang. 2022. ML4S: Learning Causal Skeleton from Vicinal Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1213–1223.
- [29] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. XInsight: eXplainable Data Analysis Through The Lens of Causality. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.
- [30] Pingchuan Ma, Zhenlan Ji, Qi Pang, and Shuai Wang. 2022. NoLeaks: Differentially Private Causal Discovery Under Functional Causal Model. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2324–2338.
- [31] Pingchuan Ma, Zhenlan Ji, Peisen Yao, Shuai Wang, and Kui Ren. 2023. Enabling Runtime Verification of Causal Discovery Algorithms with Automated Conditional Independence Reasoning (Extended Version). *arXiv preprint arXiv:2309.05264* (2023).
- [32] C MEEK. 1995. Strong completeness and faithfulness in Bayesian networks. In *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI-95)*. 411–418.
- [33] Mathias Niepert, Marc Gyssens, Bassem Sayrafi, and Dirk Van Gucht. 2013. On the conditional independence implication problem: A lattice-theoretic approach. *Artificial Intelligence* 202 (2013), 29–51.
- [34] Judea Pearl. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann.
- [35] Judea Pearl and Azaria Paz. 1986. Graphoids: Graph-Based Logic for Reasoning about Relevance Relations or When would x tell you more about y if you already know z? In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI 1986)*.
- [36] Karl Pearson. 1900. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50, 302 (1900), 157–175.
- [37] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. 2017. *Elements of causal inference: foundations and learning algorithms*. The MIT Press.
- [38] Andrea Pinna, Nicola Soranzo, and Alberto De La Fuente. 2010. From knockouts to networks: establishing direct cause-effect relationships through graph analysis. *PLoS one* 5, 10 (2010), e12912.
- [39] Grigore Roşu, Wolfram Schulte, and Traian Florin Şerbănuţă. 2009. Runtime verification of C memory safety. In *Runtime Verification: 9th International Workshop, RV 2009, Grenoble, France, June 26–28, 2009. Selected Papers 9*. Springer, 132–151.
- [40] Jakob Runge, Sebastian Bathiany, Erik Bollt, Gustau Camps-Valls, Dim Coumou, Ethan Deyle, Clark Glymour, Marlene Kretschmer, Miguel D Mahecha, Jordi Muñoz-Mari, et al. 2019. Inferring causation from time series in Earth system sciences. *Nature communications* 10, 1 (2019), 1–13.
- [41] Julien Siebert. 2023. Applications of statistical causal inference in software engineering. *Information and Software Technology* (2023), 107198.
- [42] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. 2000. *Causation, prediction, and search*. MIT press.
- [43] Richard P Stanley. 1973. Acyclic orientations of graphs. *Discrete Mathematics* 5, 2 (1973), 171–178.
- [44] Milan Studeny. 1992. Conditional independence relations have no finite complete characterization. In *Transactions of the 11th Prague Conference In Information Theory, Statistical Decision Functions and Random Processes*. 377–396.
- [45] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. 2022. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*. 338–349.
- [46] Kentaro Tanaka, Milan Studeny, Akimichi Takemura, and Tomonari Sei. 2015. A linear-algebraic tool for conditional independence inference. *Journal of Algebraic Statistics* 6, 2 (2015), 150–167.
- [47] Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. 2006. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine learning* 65, 1 (2006), 31–78.
- [48] Lun Wang, Qi Pang, and Dawn Song. 2020. Towards practical differentially private causal graph discovery. *Advances in Neural Information Processing Systems* 33 (2020), 5516–5526.
- [49] Zhaoyu Wang, Pingchuan Ma, and Shuai Wang. 2023. Towards Practical Federated Causal Structure Learning. *arXiv preprint arXiv:2306.09433* (2023).
- [50] Dominik Winterer, Chengyu Zhang, and Zhendong Su. 2020. Validating SMT solvers via semantic fusion. In *Proceedings of the 41st ACM SIGPLAN Conference on programming language design and implementation*. 718–730.

- [51] Christoph M Wintersteiger, Youssef Hamadi, and Leonardo De Moura. 2013. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design* 42, 1 (2013), 3–23.
- [52] Depeng Xu, Shuhan Yuan, and Xintao Wu. 2017. Differential privacy preserving causal graph discovery. In *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*. IEEE, 60–71.
- [53] Tianyin Xu and Yuanyuan Zhou. 2015. Systems approaches to tackling configuration errors: A survey. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–41.
- [54] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data*. 1599–1614.
- [55] Zhalama Zhalama, Jiji Zhang, Frederick Eberhardt, Wolfgang Mayer, and Mark Junjie Li. 2019. ASP-based discovery of semi-Markovian causal models under weaker assumptions. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 1488–1494.
- [56] Jiji Zhang. 2008. On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence* 172, 16–17 (2008), 1873–1896.
- [57] Mengdi Zhang and Jun Sun. 2022. Adaptive fairness improvement based on causality analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 6–17.