TU WIEN Informatics

Causal Discovery for Metric-based Root Cause Analysis in Application Performance Monitoring

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Stephan Siegl, BSc
Matrikelnummer 01622471

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dipl.-Ing. Dr.sc. Jürgen Cito, BSc

Wien, 1. November 2022

Stephan Siegl	Jürgen Cito

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Causal Discovery for Metric-based Root Cause Analysis in Application Performance Monitoring

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Stephan Siegl, BSc
Registration Number 01622471

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Ing. Dr.sc. Jürgen Cito, BSc

Vienna, 1st November, 2022

Stephan Siegl	Jürgen Cito

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Stephan Siegl, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. November 2022

Stephan Siegl

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Diese Arbeit konnte nur durch vielseitige Unterstützung entstehen, für die ich mich an dieser Stelle herzlich bedanken möchte. Als erstes gilt mein Dank der Firma Dynatrace, für die großartige Möglichkeit meine Abschlussarbeit im Zuge einer Anstellung dort durchführen zu können – insbesondere Thomas Natschläger und Florian Perteneder für die vielen inhaltlichen Gespräche und das große Vertrauen, das sie mir entgegengebracht haben. Auch danke ich meinen Betreuer Assistant Prof. Dipl.-Ing. Dr.sc. Jürgen Cito BSc für seine konstruktiven Vorschläge und Ideen.
Des Weiteren möchte ich mich bei den vielen Menschen aus meinem persönlichen Umfeld, die mich nicht nur während der Masterarbeit, sondern auch in allen anderen Lebenslagen immer unterstützen, bedanken. Vielen Dank an meine Familie für ihr Vertrauen und ihre Liebe zu jeder Zeit und in jeder Phase meines Lebens. Ganz besonders möchte ich mich bei meiner Mutter bedanken, für ihren unermüdlichen Einsatz für ein besseres Leben für uns und mich. Abschließend auch ein besonderer Dank von Herzen an Anna, für ihre unerschütterliche Zuversicht und die schöne Zeit, seit wir uns kennen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die Hauptursache eines Softwarefehlers zu identifzieren ist aufgrund der Komplexität heutiger Softwaresysteme oft ein äußerst schwieriges und zeitaufwendiges Unterfangen und daher versuchen Root Cause Analysis (RCA) Systeme diese Ursachensuche zu optimieren, indem sie Orientierungshilfe leisten. Ein wesentlicher Bestandteil dieser Orientierungshilfe ist dabei eine Graphenstruktur, die kausale Abhängigkeiten zwischen und innerhalb verschiedener Komponenten einer Softwareanwendung abbildet. Diese Beziehungen müssen oftmals automatisch anhand von Daten – besonders häufig Leistungs- und Ressourcennutzungsmetriken – gelernt werden. Zu diesem Zweck verwenden viele bestehende RCA-Systeme statistische Algorithmen, die häufig unter dem Begriff Causal Discovery zusammengefasst sind. Obwohl diese Algorithmen bereits in verschiedenen anderen Disziplinen verwendet wurden, gibt es bisher nur vereinzelt Untersuchungen, wie nützlich und anwendbar diese Algorithmen für die Aufgabe der Ursachenanalyse im Bereich Application Performance Monitoring (APM) sind und wie gut sie im Vergleich zu einfacheren Techniken funktionieren. Deshalb vergleicht und bewertet diese Arbeit mehrere unterschiedliche Causal Discovery Algorithmen im Hinblick auf die gerade genannte Aufgabe, indem Metrik-Zeitreihendaten von einer tatsächlichen Softwareanwendung gesammelt und als Eingabedaten für die zu bewertenden Algorithmen verwendet werden. Die durchgeführte Auswertung zeigt, dass sich die untersuchten Causal Discovery Algorithmen hinsichtlich ihrer Fähigkeiten durchaus beachtlich unterscheiden können, die meisten von ihnen jedoch klar bessere Ergebnisse liefern als einfachere Techniken wie die Verwendung der Pearson-Korrelation. Diese Ergebnisse deuten darauf hin, dass viele bestehende RCA-Systeme, deren Funktionsweise eine Graphenstruktur voraussetzt, durch die Verwendung eines hier ausgewerteten Causal Discovery Algorithmus verbessert werden könnten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Identifying the root cause of a software failure in today's complex software environments is often an extremely difficult and time consuming undertaking and therefore automatic root cause analysis (RCA) systems try to optimize this root cause search by providing guidance. In many cases, a vital part for providing this guidance is a graph like structure which encodes causal dependencies between and within different components of a software application and these relationships often need to be learned automatically from data, most commonly from performance and resource utilization metrics. To do so, many existing RCA systems utilize statistical algorithms that fall under the category of causal discovery, which already have been used in various other disciplines. Nonetheless, so far, little work has been done to understand how useful and applicable different causal discovery methods are for the task of root cause analysis within the domain of application performance monitoring (APM) and how they do perform compared to simpler techniques. Therefore, this work compares and evaluates multiple different causal discovery algorithms with regard to the just stated task by collecting metric time series data from an actual software application and using it as input for the to be evaluated algorithms. This comparison revealed that the evaluated causal discovery methods can differ quite substantially with regard to their capabilities, but overall most of them still outperform simpler techniques like the Pearson correlation. These results suggest that many existing RCA systems that rely on a graph like structure for their root cause search could greatly benefit from using one of the evaluated causal discovery algorithms.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

In today's world, software systems are everywhere and the revenue and success of every company becomes more and more reliant on correctly functioning software, a fact that is particularly evident when IT systems do not work as expected. In many cases, performance degradations alone can have serious impacts on customer satisfaction like Google losing 20% of its traffic if their website responds merely 500 ms slower [1]. Downtimes of IT systems are in most cases even more expensive with technology research firm Gartner Inc. estimating the average cost of downtime, based on industry surveys, to be around 5600$ per minute [2].
Due to this increasing importance of digital services and the need for extremely high system reliability, developing and operating modern software applications have become more and more complicated, which lead to a rise in the adaption of application performance monitoring (APM) systems. These systems continuously observe the behaviour of software applications and their IT environments by collecting and analyzing logs, stack traces and system metrics to prevent potential system failures and to provide help in case of a failure [3]. A technology company that specialises in the field of application performance monitoring is Dynatrace Inc.[1] and this work was created as part of an employment there.

1.1 Motivation

Identifying the root cause of a software failure in today's complex software environments is often a extremely difficult and time consuming undertaking and therefore automatic root cause analysis (RCA) systems try to optimize this root cause search by providing guidance. In many cases, a vital part for providing this guidance is a graph like structure that encodes causal dependencies between and within different components of a software application and these relationships often need to be learned automatically from data,

[1] https://www.dynatrace.com

most commonly from performance and resource utilization metrics [4]. The root cause search itself is then performed by traversing this graph, either starting from the metric nodes, where anomalies were detected or from business critical metrics like the end-user response time measured at the application front-end [4]. For constructing such inference graphs, many RCA systems [5–10] go beyond simple techniques for learning associations between metric data, such as Pearson correlation, and use more sophisticated methods that fall under the category of causal discovery.

Causal discovery methods have been used on numerous data sets from different disciplines to detect relationships between variables or to identify causal factors within various processes [11] and therefore it is no surprise that these methods are also being used within the domain of software monitoring. Nonetheless, during the course of a literature search, two knowledge gaps with regard to the combination of causal discovery and root cause analysis became apparent. First, all of the root cause analysis systems found use the same two types of causal discovery methods with the vast majority of the systems even using the exact same algorithm despite the fact that new causal discovery approaches and concrete algorithms are regularly proposed. Furthermore, although there exist works that experimentally compare multiple newer types of causal discovery algorithms, most of these evaluations are done using either synthetic data or data from different domains [12, 13], meaning that their results are unlikely to be applicable for the domain of interest. Second, the results and findings regarding the performance and capabilities of the found root cause analysis systems are similarly difficult to compare as the construction of inference graphs using causal discovery methods in all cases represents just a small part of the each of these systems.
Overall these findings can be summarized by stating that, so far, little work has been done to understand how applicable different causal discovery methods are for the task of root cause analysis within the domain of application performance monitoring.

1.2 Research Questions

Therefore, this thesis compares and evaluates existing statistical algorithms that fall under the category of causal discovery regarding their usefulness for the just stated task. More specifically, within the course of this work the following research questions (RQ) should be answered:

1. How well can causal discovery algorithms detect connections between the causes and the effects of software failures based on metric time series data alone? How successfully can these methods identify the software component that is causing the failure?

2. How well do these algorithms perform in comparison to two simple baseline approaches – one using the Pearson correlation and one using the graphical lasso – with regard to the tasks stated in RQ1?

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3. How sensitive is the performance of the examined causal discovery algorithms and the baseline approaches with regard to the start and end points of the metric time series data?

1.3 Methodological Approach

The methodological approach to answer the just posed research questions will be based on the well-known CRISP-DM process model [14] and will include the following steps. First, multiple existing causal discovery algorithms – each based on a different concept – will be selected according to a literature review with an emphasis on methods that can handle high dimensional data sets. Second, a software application needs to be selected and set up such that both metric time series data can be collected and various software failures can be intentionally evoked. Third, multiple evaluation criteria and scenarios need to be defined that allow the answering of the research questions in a quantifiable way. In a last step, these criteria should then be used to evaluate the selected algorithms according to the beforehand specified evaluation scenarios.

1.4 Structure

The rest of this master thesis is structured as follows: After Chapter 2 provides important background knowledge on the topics of causal discovery, application performance monitoring and root cause analysis, Chapter 3 introduces existing literature that has either considerable thematic overlap with this master thesis or has influenced the creation of it in a meaningful way. Following this, Chapter 4 addresses the methodological approach outlined above and describes the course of action taken to answer the research questions in more detail. Chapter 5 then objectively states the study results obtained during the course of this work, which are thereupon discussed in Chapter 6 with regard to the originally posed research questions. Last, Chapter 7 concludes this master thesis with a short summary and highlights potential future research opportunities.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background

This chapter provides the background knowledge necessary for understanding the topic of this thesis in full. The first section introduces the idea of causal discovery, which will be the method of choice for extracting relevant knowledge from data. The second section discusses the field of application performance monitoring, which is the area of application. The last section covers the concept of root cause analysis, which will partly build upon the contents of the first two segments of this chapter.

2.1 Causal Discovery

Causal questions are omnipresent in everyone's daily life, but causality plays an even bigger role in science. For many disciplines, real interventional experiments allow scientists to disentangle relevant from non-relevant influences and help to distinguish cause from effect. One example are randomized controlled trials, which are often considered the gold standard for effectiveness research in the field of medicine [15]. Unfortunately, these experiments are often expensive and time consuming, which begs the question if similar conclusions can be reached without experiments or deliberate interventions [15]. Causal discovery aims to accomplish this by identifying causal relations from observational data – in the case of this work time series – while avoiding spurious relationships due to mere correlation. From a practical view, the general process can be described as an algorithm which performs the following task. Given a matrix $X_{T\times n}$ where T represents the length of the time series and n the number of different time series, output a graph

$$\mathcal{G} = (V, E) \text{ where } |V| = n \text{ and } E = \{(v, u) \in V^2 | \text{ v causes u } \}$$

in which every variable represents a node and edges indicate causal influences or independence between variables depending on their presence or absence.

The rest of this section is structured as follows: The first subsection introduces general assumptions and concepts of causal discovery, while the second subsection highlights

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Notation & Description	
X, Y, Z, ξ	Random variables and noise
$\boldsymbol{X}$	Multivariate time series data
$\boldsymbol{X}^j$, $\boldsymbol{X}_t^j$	jth time series, single observation of jth time series
P	Joint distribution
$\mathcal{G} = (V, E)$	Graph with nodes V and edges E
$X \not\perp\!\!\!\perp Y$	Random variables X and Y are dependent
$X \perp\!\!\!\perp Y$	Random variables X and Y are independent
$X \perp\!\!\!\perp Y \mid Z$	Random variables X and Y are independent conditioned on Z
$X - Y$	X and Y are adjacent
$X \to Y$	X is a cause of Y and Y is an effect of X
$X \not\to Y$	X is not a cause of Y

Table 2.1: Notation used in this section

idiosyncrasies specific to the case of time series data. After this, different method families of causal discovery are presented including a more detailed description of the algorithms used in the practical part of this thesis. The most relevant notation can be found in Table 2.1, which is used throughout this section unless explicitly stated otherwise.

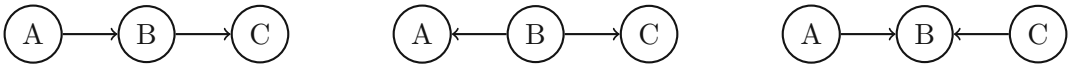
2.1.1 General Concepts and Assumptions

This subsection lays the foundation to talk about causal discovery in general by introducing needed terminology and presenting relevant aspects regardless of type of data or method used. Beginning with defining relevant terms from the field of graph theory and followed by common concepts and assumptions under which it is possible to infer a causal graph structure from a joint distribution, this subsection acts as precondition for the rest of this section.

Terms and Definitions

A **graph** $\mathcal{G} = (V, E)$ is a pair, where V is a set of **nodes** or **vertices** and E is a set of **edges** $E \subseteq \{(x, y) \mid (x, y) \in V^2 \wedge x \neq y\}$. Two nodes x and y are **adjacent**, if either $(x, y) \in E$ or $(y, x) \in E$. A node x is referred to as a **parent** if $(x, y) \in E$ and $(y, x) \notin E$ and as a **child** if $(y, x) \in E$ and $(x, y) \notin E$. If both $(x, y) \in E$ and $(y, x) \in E$, the link between the nodes is called **undirected edge**, otherwise it is called (unambiguously) **directed edge**. Likewise, a graph is called **undirected graph** if all edges are undirected and **directed graph** if all edges are directed. The **skeleton** of a graph $\mathcal{G}$ does not consider the orientation of edges, i.e. the skeleton graph contains an undirected edge for each adjacent node pair of the original graph. A sequence of edges, which are oriented in the same direction is called a **directed path**. Further on, a graph, which does not contain any directed cycles, that is, there is no node pair (u, v) with a directed path from u to v and v to u is called **partially directed acyclic graph (PDAG)**. If additionally, all edges are directed, it is called **directed acyclic graph (DAG)**.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Mediation graph (b) Confounding graph (c) Collider graph

Figure 2.1: Building blocks of causal networks [16]

Graph structures can vary widely regarding their size and overall complexity, but as pointed out by Judea Pearl, "there are three basic types of junctions, with the help of which we can characterize any pattern of arrows in the network" [16]. Therefore, the following paragraph is based on Pearl's book *The Book of Why* [16] and explains these junctions in more detail. Graph 2.1a can be seen as a simple mediation model where B transmits the effect of A to C. An example would be $Fire \rightarrow Smoke \rightarrow Alarm$, because fire alarms act upon smoke and not fire. Conditioning on B by eliminating smoke will make A and C independent resulting in no alarm regardless of the presence of a fire. Graph 2.1b represents confounding, where A and C are linked by their common cause B, which makes "A and C statistically correlated even though there is no direct causal link between them" [16]. An example would be *Shoe size* $\leftarrow$ *Age of Child* $\rightarrow$ *Reading ability*, as children with bigger shoe sizes tend to have better reading skills, resulting in spurious correlation between the two variables, which can be eliminated by conditioning on the confounder *Age of Child*. Graph 2.1c is called a collider or v-structure or immorality, which works in exactly the opposite way as the first two structures, as conditioning on the B will make the normally independent variables A and C dependent. An example would be *Talent* $\rightarrow$ *Film star* $\leftarrow$ *Beauty* under the assumption that both talent and beauty are important for becoming a successful actor, while being unrelated in the general population.

Reflecting upon these examples already highlights some of the difficulties when deriving graphical models from observational data. Depending on the actual graph structure, which in reality remains unknown, conditioning on the *right variables* is necessary to avoid spurious correlation, while conditioning on the *wrong variables* results in detecting links, which are not there.

A generalization of this concept of graphical dependence or independence is the one of d-separation originally formulated by Pearl (1988) [17]. Here it is given as a definition, closely following Assaad et al. (2022) [13].

Definition 1 (Pearl's d-separation) *If $\mathcal{G}$ is a DAG in which X and Y are two vertices and Z is a set of vertices, then X and Y are d-connected by Z in G if and only if there exists a path U between X and Y such that for every collider V on U, either V or a descendant of V is in Z and no non-collider on U is in Z. Otherwise, X and Y are d-separated given Z.*

Peters et al. (2017) [18] use the notation $A \perp\!\!\!\perp_{\mathcal{G}} B \mid C$ to indicate that vertices A and B are d-separated by a subset C.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Although the concept of d-separation is a purely graphical one, together with the definition of the Markov property, a commonly used assumption for graphical models, one can relate "statements about graph separation to conditional independences" [18], which is crucial for the task of causal discovery.

In fact, there is not one but three Markov properties, but as long as the joint distribution has a density these definitions are equivalent [18] and therefore we refer to all of them simply as Markov property. The presented definition follows Peters et al. (2017) [18], but can also be found in a similar form in older publications like Lauritzen et al. (1990) [19].

Definition 2 (Markov property) *Given a DAG $\mathcal{G}$ and a joint distribution P, this distribution is said to satisfy*

*(i) the **global Markov property** with respect to DAG $\mathcal{G}$ if*

$$A \perp\!\!\!\perp_{\mathcal{G}} B \mid C \implies A \perp\!\!\!\perp B \mid C$$

for all disjoint vertex Sets A, B, C

*(ii) the **local Markov property** with respect to the DAG $\mathcal{G}$ if each variable is independent of its non-descendants given its parents, and*

*(iii) the **Markov factorization property** with respect to the DAG $\mathcal{G}$ if*

$$p(\boldsymbol{x}) = p(x_1, \ldots, x_d) = \prod_{j=1}^{d} p(x|parents_j^{\mathcal{G}})$$

assuming that $P_{\boldsymbol{X}}$ has a density p.

To indicate that a joint distribution satisfies the Markov property, one can also say that a DAG $\mathcal{G}$ and a joint distribution are compatible [13] or that the joint distribution is Markovian with respect to $\mathcal{G}$ [18]. Although the Markov property is valid for all kinds of graphic models, e.g. Bayesian networks, in the context of causality the local Markov property is often rephrased that each variable X given its direct causes is independent of all the variables that are not direct causes or effects of X, which is sometimes referred to as Causal Markov condition [20].

While the Markov Property, if satisfied, allows reading off statements of independence from the graph structure, Faithfulness enables the opposite, deriving knowledge of the graph structure based on independence observed in the data.

Definition 3 (Faithfulness) *A DAG $\mathcal{G}$ and a compatible probability distribution P are faithful to one another if all and only the conditional independence relations true in P are entailed by the Markov condition applied to $\mathcal{G}$.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The given definition can be found in similar form in Assaad et al. (2022) [13] and older publications like Spirtes et al. (2000) [21]. Furthermore, Peters et al. (2017) [18] define Faithfulness by stating that P and $\mathcal{G}$ are faithful to one another if

$$A \perp\!\!\!\perp B \mid C \implies A \perp\!\!\!\perp_{\mathcal{G}} B \mid C$$

for all disjoint vertex sets A, B and C.

Reflecting upon the logical notation of Faithfulness, two interesting aspects become apparent: First, the negation of the formula shows that Faithfulness allows the inference of dependences from the graph structure and second, Faithfulness represents the opposite of the global Markov property, as introduced above.

Furthermore if both Faithfulness and the Markov property are satisfied, then Causal Minimality is also satisfied [18].

Definition 4 (Minimality Condition) *A DAG $\mathcal{G}$ compatible with a probability distribution P is said to satisfy the minimality condition if P is not compatible with any proper sub graph of $\mathcal{G}$.*

This definition is given in Assaad et al. (2022) [13], but can also be found in similar form in Peters et al. (2017) [18] and Spirtes et al. (2000) [21]. To put it more simply, minimality requires that "the graph does not contain dependencies not present in the observational data" [13].

While Minimality is easier to understand, Faithfulness is less intuitive. To better understand it, an example, taken from Peters et al. (2017) [18], which violates Faithfulness, is given.

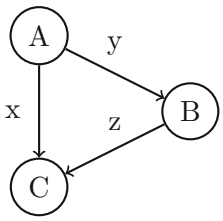


Figure 2.2: Example of Faithfulness violation [18]

In this example, the variables are all subject to normally distributed noise and are linearly dependent according to:

$$A := \xi_A$$
$$B := yA + \xi_B$$
$$C := xA + zB + \xi_C$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

In the case of

$$y \cdot z + x = 0$$

two paths cancel each other and the distribution becomes not faithful with respect to the graph shown in Figure 2.2 since the independence $A \perp\!\!\!\perp C$ is not implied by the graph structure, while minimality is still satisfied if none of the parameters vanish. Therefore, many studies and causal discovery methods, which aim at inferring causal structures from observational data require Faithfulness [13].

Having a distribution P and its underlying DAG $\mathcal{G}$ satisfy both the Markov and Faithfulness condition, furthermore, creates "a one-to-one correspondence between the d-separation statements in the graph $\mathcal{G}$ and its corresponding conditional independence statements in the distribution" [18]. While this already greatly diminishes the number of graphs a distribution is potentially compatible with, there are graphs which encode the exact same set of conditional independences and between which one can not differentiate [18]. This will be discussed in more detail in Section 2.1.3.

So far, it was assumed that the set of *relevant* variables is known and all of these variables have been measured. In practice however, as these variables need to be selected, this begs the question how to determine if a variable is *relevant* or not. Going back to Graph 2.1b at the beginning of this subsection, one may realize that *relevant* can not simply be limited to the variables of interest, as a non-selected confounder of two selected variables could lead to the inference of spurious correlation among these variables, although they are independent in reality. A classical example is Simpson's paradox, where not including a confounder can lead to wrong causal conclusions or contradicting statements [18]. To avoid such problems, one needs to make sure that all common causes are actually observed, which is often summarized under the term Causal Sufficiency.

Definition 5 (Causal Sufficiency) *A set of variables is said to be causally sufficient if all common causes of all variables are observed.*

Again, this definition can be found in Assaad et al. (2022) [13], Peters et al. (2017) [18] and Spirtes et al. (2000) [21].

In practice, Causal Sufficiency is hard to achieve as most applications can not be observed in complete isolation and therefore some causally relevant variables will always be unobserved [18].

2.1.2 Causal Discovery for Time Series Data

All the definitions and concepts presented so far are not bound to any specific kind of data, which does not mean that the process of causal discovery is always the same. Knowing that observations of variables have an inherent time structure, because they are

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

collected over time, results in some core differences. First, individual data points of a time series variable are not independent, which differs from the usual i.i.d setting, where every variable is observed several times. Second, due to the fact that a cause always precedes its effects, causal discovery in time series becomes easier in some sense [13, 18]. Despite this, there are still a few remaining issues, which will be discussed at the end of this subsection.

Graph Representations for Time Series

Different to the notation of the previous subsection, where samples are i.i.d drawn from the joint distribution $P_{\boldsymbol{X}}$, the d-variate time series data is denoted by $\boldsymbol{X}$, a single time series is denoted by $\boldsymbol{X}^j$ and a single observation is denoted by $\boldsymbol{X}_t^j$.

Based on this, the causal graph structure underlying the time series data can be represented in different ways, two of which will be introduced here. Both definitions closely follow Assaad et al. (2022) [13], but can also be found in a similar form in Peters et al. (2017) [18].

Definition 6 (Full Time Graph) *Let X be a multivariate discrete-time stochastic process and $\mathcal{G} = (V, E)$ the associated full time graph. The set of vertices in that graph consists of the set of components $X^1, \dots, X^d$ at each time $t \in \mathbb{Z}$. The edges E of the graph are defined as follows: variables X_t^p and X_{t+i}^q are connected by a lag-specific directed link $X_t^p \to X_{t+i}^q$ in $\mathcal{G}$ pointing forward in time if and only if X^p causes X^q at time t with a time lag of $i > 0$ for $p = q$ and with a time lag of $i \geq 0$ for $p \neq q$.*

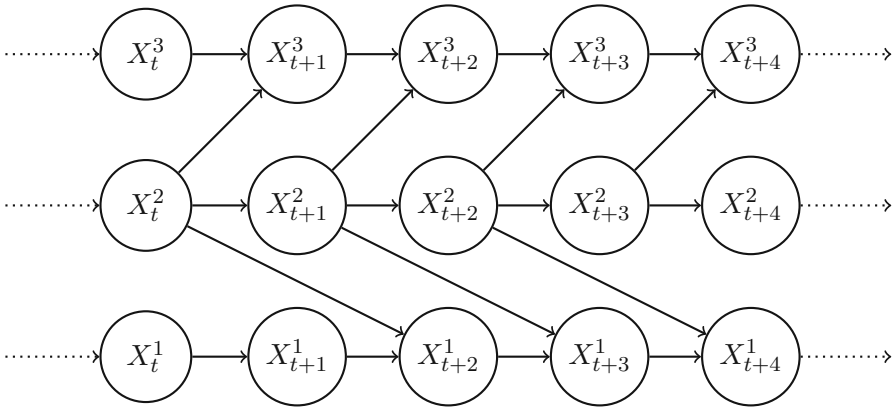


Figure 2.3: Full Time Graph

This DAG is a graph with infinitely many nodes and acts more as a theoretical concept as inferring this graph based on a single observation for each time series and time instant is practically impossible [13]. Nonetheless, under the assumptions that the causal relations stay consistent over time, one is able to infer the causal relations between individual time series including the respective lag between them [13]. This assumption is sometimes

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

referred to as Causal Stationarity, which represents a weaker form of Stationarity [22]. A more compact representation, which is often sufficient in practice, is the summary graph. Here, different to the full time graph, edges represent causal relations between time series as a whole without any time information.

Definition 7 (Summary Graph) *Let X be a multivariate discrete-time stochastic process and $\mathcal{G} = (V, E)$ the associated summary graph. The set of vertices in that graph consists of the set of time series $X^1, \ldots, X^d$. The edges E of the graph are defined as follows: Variables X^p and X^q are connected if and only if there exists some time t and some time lag i such that X_t^p causes X_{t+i}^q at time t with a time lag of $i > 0$ for $p = q$ and with a time lag of $i \geq 0$ for $p \neq q$.*

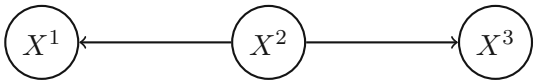


Figure 2.4: Summary Graph

This also means that although a full time graph is acyclic, its corresponding summary graph may still contain cycles.

Practical Limitations of Cause Preceding Effect

In an ideal setting the inherent direction within time series data makes it easier to orient detected causal relations. However, in practice the time difference between cause and effect of individual time series often has not been observed, because the sampling process was "slower than the time scale of the causal process" [18]. For some applications the underlying process happens so fast, that measuring becomes either impossible or extremely expensive, while for others the overhead associated with storing and processing more data due to a higher sampling rate is just not economically feasible. Regardless of the reason, in these cases cause and effect can appear instantaneous in the time series data, i.e. as if they happened at the same time instance. Such causal relations are therefore often called instantaneous or contemporaneous. As sub sampling is such a common problem, many causal discovery methods for time series data often have mechanisms in place to decide on a direction also for these kind of links.

In addition to sub sampling, time series data can also be aggregated, i.e. each data point consists of averages or sums of consecutive observations of the original measurements. Discovering the underlying causal structure for aggregated and sub-sampled time series can make the task of causal discovery even more difficult, but under specific circumstances it can nonetheless be possible to recover the causal relations at the original frequency [23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.1.3 Causal Discovery Method Families

In this subsection four groups of causal discovery methods, each based on a different concept, will be presented and for each group the one method, which was evaluated in the practical part of this work, will be described in more detail.

Causal discovery in time series is an active research field, where new techniques are regularly proposed, and thus classification into different, distinct, groups is not always precise [13]. The most relevant and therefore selected method families are (conditional) independence-based methods, Granger Causality, score-based methods and noise-based methods, which is in line with the works of Peters et al. (2017) [18] and Assaad et al. (2022) [13]. The description for each of these method families is constructed likewise. After a general explanation and potential references to relevant ground work, the mechanics of one concrete method for time series data are explained and a reference to the implementation, which was used for the evaluation, is given.

Independence-based Methods

Independence-based methods use the information of conditional independences available in the distribution to construct the underlying causal graph under the assumption that the distribution is faithful [13]. This causal graph "is however not unique as several DAGs can be used to represent the same set of conditional independencies" [13]. A set of such DAGs is called Markov equivalence class. An example are the two three-variable junctions [2.1a, 2.1b], which both encode the independence statement $A \perp\!\!\!\perp C \mid B$ and are therefore Markov equivalent. Verma and Pearl (1991) [24] have shown that two DAGs are Markov equivalent if and only if they have the same skeleton and the same v-structures.

In practice, "most independence-based methods first estimate the skeleton" [18], i.e. an undirected version of the final causal graph. This involves trying to d-separate each pair of nodes (X, Y) by searching for a set of variables $\boldsymbol{A}$ not containing X or Y, which satisfy $X \perp\!\!\!\perp Y \mid \boldsymbol{A}$. After this, the skeleton can be oriented using the information of the separating sets and additional orientation rules [18].

So far it was assumed that the derived independence statements are unambiguous, but in reality they have to be inferred from a finite amount of data, which can lead to "testing results [that] might even contradict each other in the sense that there is no graph structure that encodes the exact set of inferred conditional independences" [18], which is especially true for non-parametric conditional independence tests. In the case of linear dependence, one can test for vanishing partial correlation.

One of the oldest independence-based methods is the SGS algorithm [21], which searches through all possible separating sets, as described above, and therefore becomes impractical as the number of variables increase. This shortcoming was addressed by the PC algorithm [21, 25], which greatly improved the run time, especially for sparse graphs. Although both of these methods were designed for non temporal data, many independence-based methods for time series still partly function in a similar way like the PCMCI+ algorithm [26].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

PCMCI+ algorithm

The PCMCI+ algorithm is an enhanced version of the PCMCI algorithm [27], which is named after the two stages it consists of: (1) a PC like condition selection phase to identify relevant conditions $\boldsymbol{P}(X_t^j)$ for all time series variables $X_t^j \in \{X_t^1, \dots, X_t^d\}$ and (2) the momentary conditional independence (MCI) test, which tests whether $X_{t-\tau}^i \to X_t^j$ with

$$X_{t-\tau}^i \perp\!\!\!\perp X_t^j \mid \boldsymbol{P}(X_t^j) \setminus \{X_{t-\tau}^i\}, \boldsymbol{P}(X_{t-\tau}^i)$$

The goal of the first phase is to remove as many irrelevant conditions for each of the d variables to ensure that the remaining conditioning set of each variable only represents a small subset of its causal parents. This avoids having to condition on the whole past of all the processes in the second phase, as the parents of a variable are already a sufficient conditioning set according to the causal Markov condition. More specifically, the MCI test in the second phase uses the estimated conditions for independence testing in two ways: (1) conditioning on the parents of X_t^j tries to control for indirect and common causes while (2) conditioning on the parents of $X_{t-\tau}^i$ tries to account for autocorrelation effects [27].
While the original PCMCI algorithm was designed to only consider lagged causal links, its extension PCMCI+ allows for the discovery of contemporaneous links [26, 27]. Both methods assume that the Markov property, Causal Sufficiency, Faithfulness and Causal Stationarity are satisfied. Their implementations are open source[1] and available as part of the python package *tigramite*.

Granger Causality

Granger Causality is one of the oldest concepts of causal discovery and its various realizations are still among the most popular approaches to infer causal links between time series [13, 18]. Due to its simplicity it is so well known and ubiquitous that it is not possible to cover all existing concepts and even for the introduced concepts, many different variations exist. Nonetheless a small overview should be given, but completeness is not claimed.

In its original form it states that "Y_t is causing X_t if we are better able to predict X_t using all available information than if the information apart from Y_t had been used" [28].

The simplest version of Granger Causality is the bivariate case, where dependencies are assumed to be linear. In this case, two autoregressive models, an restricted model, which only uses past values of itself for prediction, is compared to an augmented model, which uses both past values of itself and of its potential cause. In case the augmented model is significantly more accurate, using some statistical test like the F-test, one concludes that X^i Granger-causes X^j.

[1] https://github.com/jakobrunge/tigramite

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

$$\mathcal{M}_{res} : X_t^j = \sum_{k=1}^{\tau} a_k X_{t-k}^j + \xi_t$$

$$\mathcal{M}_{aug} : X_t^j = \sum_{k=1}^{\tau} a_k X_{t-k}^j \sum_{k=1}^{\tau} b_k X_{t-k}^i + \tilde{\xi}_t$$

However the use of the bivariate version is highly inappropriate in most cases as Causal Sufficiency would not be satisfied, which Granger already emphasized by requiring the use of all available information for prediction [28].

A natural way to overcome the problem of Causal Insufficiency and to include all relevant information is to change the autoregressive models to vector autoregressive (VAR) models, where the augmented model uses all time series to predict a variable X^q, whereas the restricted model uses all time series except for X^p to test whether X^p Granger-causes X^q [13]. In addition to the just mentioned extension, Arnold et al. (2007) [29] examined several multivariate Granger methods, one of which is the *Granger Lasso*.

Granger Lasso

The Granger Lasso combines the notation of Granger Causality with the usage of *Lasso Regression* [30] to leverage its ability of variable selection and regularization. For each variable the standard lasso optimization problem, finding the best parameters $\vec{w}$, while minimizing

$$\frac{1}{n} \sum_{(\vec{x},y) \in S} |\vec{w} \cdot \vec{x} - y|^2 + \lambda \, ||\vec{w}||_1$$

is used, with y representing the prediction target and $\vec{x}$ the lagged versions of all the variables, up to a maximum lag [29]. A implementation of this idea is available as part of the python package *causal-learn*, which is open source[2]. For this specific implementation the regularization parameter λ is selected through cross validation and, different to other causal discovery methods, it only returns the regression parameters $\vec{w}$. Therefore, these parameters are then interpreted as causal influences, with regression coefficients being zero or below a certain threshold seen as no causal dependency.

Score-based Methods

In the case of score-based methods, the problem of inferring a causal graph is treated as an optimization problem, where the objective is to find the best matching network structure given the data with the rational being that graph structures encoding the wrong conditional independences will yield bad model fits [18]. The notation of best fit is normally based on a score that considers both the "the likelihood of the data given the

[2] https://github.com/cmu-phil/causal-learn

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

network and a penalty term related to the complexity of the network“ [13]. Furthermore, in most cases the graph structure is subject to some constraint, the most common one being that it represents a DAG.
While the availability of a score to assess the conformity of a graph structure with respect to the data is great, in practice, finding the best scoring graph among all possible ones becomes quickly infeasible as the number of potential DAGs is growing super-exponentially with the number of variables [18]. For example, there are already 4175098976430598143 possible DAGs for a directed graph with only 10 variables [31]. Even when employing more sophisticated search methods or heuristics, the combinatorial constraint makes it difficult to find good solutions for problems with a high number of variables.

Nonetheless and despite this bad looking premise, Zheng et al. (2018) [32] were able to reformulate the structure learning problem, which consists of optimizing a graph score under a combinatorial constraint, into a purely continuous optimization problem, completely avoiding the constraint, and thus making it much easier to apply score-based methods to high dimensional problems. Based on this insight, Pamfil et al. (2020) [33] created a new score-based causal discovery method for time series called *DYNOTEARS*.

DYNOTEARS

Here, the data is modeled using the following *structural vector autoregressive* (SVAR) model

$$\boldsymbol{X} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{Y}_1\boldsymbol{A}_1 + \ldots \boldsymbol{Y}_\tau\boldsymbol{A}_\tau + \boldsymbol{Z}$$

where $\boldsymbol{X}$ is an $n \times d$ matrix with d being the number of individual time series and n the effective sample size, $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_\tau$ are time-lagged versions of X and Z is a matrix of centered error variables. This model can be written in a more compact form by stacking the time-lagged matrices $\boldsymbol{Y}_i$ and their respective weights like

$$\boldsymbol{X} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{Y}\boldsymbol{A} + \boldsymbol{Z}$$

The adjacency matrices $\boldsymbol{W}$ and $\boldsymbol{A}$ are the estimated under the constraint that contemporaneous links are not allowed to form cycles, which is equivalent to the requirement that $\boldsymbol{W}$ is acyclic. The resulting optimization problem can then be formulated as

$$\begin{aligned} \underset{\boldsymbol{W},\boldsymbol{A}}{\text{minimize}} \quad & f(\boldsymbol{W},\boldsymbol{A}) \\ \text{subject to} \quad & \boldsymbol{W} \textit{ is acyclic} \end{aligned}$$

with

$$f(\boldsymbol{W},\boldsymbol{A}) = \frac{1}{2n}||\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W} - \boldsymbol{Y}\boldsymbol{A}||_F^2 + \lambda_{\boldsymbol{W}}||\boldsymbol{W}||_1 + \lambda_{\boldsymbol{A}}||\boldsymbol{A}||_1$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This is a classical continuous optimization problem, except for the acyclicity constraint of $\boldsymbol{W}$, which however can be replaced by the trace exponential function $h(\boldsymbol{W}) = trace(e^{\boldsymbol{W} \circ \boldsymbol{W}}) - d$, due to Zheng et al. (2018) [32], as they showed that $h(\boldsymbol{W}) = 0$ if and only if $\boldsymbol{W}$ is acyclic.[3] This allowed the authors of the paper to reformulate the original optimization problem into an equality constrained problem, which they solved using the augmented Lagrangian method.
In their paper, the authors of the *DYNOTEARS* method state that stationarity for the time series data is required and that guarantees for identifying the correct SVAR model can only be given for cases, where either the errors in Z are non-Gaussian or if they are standard Gaussian $\mathcal{N}(0, I)$ [33]. In general, the guarantees for inferring the correct underlying graph using score-based method are weaker compared to, e.g. independence-based approaches [13]. An implementation of their approach is available as part of the python package *causalnex* and is open source[4].

Noise-based Methods

Noise-based methods rest on the idea that the asymmetry of the data generating process alone – effects are results of their causes and not vice versa – allows the inference of causal relations from data. In this scenario, a causal system is formally described by a set of assignments, where variables are the result of functions, which take other variables and noise as their input.

Consider a simple problem with two variables, where X causes Y ($X \to Y$) through some function f:

$$X := \xi_X$$
$$Y := f(X, \xi_Y)$$

While in the most general case, where the type of the function f is not restricted, it is not possible to infer the correct structure from the distribution $P_{X,Y}$ alone (*Non-uniqueness of graph structures*), the underlying data generation process may still become identifiable in certain cases [18]. One of them is where the effect Y is a linear function of its cause X up to an additive noise term:

$$Y := aX + \xi_Y, \quad \xi_Y \perp\!\!\!\perp X$$

Assuming that either the cause X or the effect noise ξ_Y are non-Gaussian renders the causal direction identifiable, a fact used by Shimizu et al. (2006) [34] to develop a causal discovery algorithm named *LiNGAM*, short for *Linear Non-Gaussian Acyclic Model*, which estimates causal models from continuous-valued data. In essence, this method

[3] Here, $\circ$ denotes the Hadamard product of two matrices.
[4] https://github.com/quantumblacklabs/causalnex

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

works by decomposing the multivariate data into independent non-Gaussian signals using *independent component analysis* (ICA) [35], which are then permuted to find an order that aligns best with the assumption of a DAG structure [34].
Not long afterwards, the applicability of the *LiNGAM* approach was extended to time series data by Hyvärinen et al. (2010) [36].

VARLiNGAM

Their method, called *VARLiNGAM*, combines the non-Gaussian instantaneous model with an autoregressive model, resulting in a *structural vector autoregressive* (SVAR) model, similar to the one in the previous section. The estimation of the weights is done in a two stage process: (1) a classical autoregressive model is fitted using least-squares regression and the resulting residuals are then used in (2) to perform the *LiNGAM* analysis, which gives an estimate of the instantaneous causal model. At the end the autoregressive coefficients are adjusted to incorporate the information of the instantaneous model.

Except for the method's inherent assumption of non-Gaussian error variables, the method only assumes Causal Sufficiency [36]. Its implementation is available as part of the python package *lingam* and is open source[5].

2.2 Application Performance Monitoring

APM stands for Application Performance Monitoring, respectively Application Performance Management, and is often used as an umbrella term for methods and tools that continuously monitor the state and the performance of software applications and their environments to detect, diagnose and respond to occurring problems. In this context, monitoring mainly refers to the collection and analysis of operational data, most notably, of application and system logs, resource utilization and KPI metrics, and application traces [3].

The remaining section is divided into two parts. First, APM is viewed from an end-user perspective, highlighting why IT system monitoring is becoming more and more important and what functionalities APM solutions can provide. Second, the general structure most APM systems follow is introduced with a focus on data collection and data processing.

2.2.1 End-user Perspective

In today's world, software systems are everywhere and the revenue and success of every company becomes more and more reliant on correctly functioning software. This fact becomes especially apparent, when IT systems do not work as expected with a report from IT industry group CISQ estimating the cost of poor quality software in the United States to be around 2.84 trillion $ in 2018 alone [37]. Moreover, in many cases performance degradation alone can have serious impacts on customer satisfaction like Google losing 20%

[5] https://github.com/cdt15/lingam

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

of its traffic if their web site responds 500 ms slower [1]. Downtimes of IT systems are in most cases even more expensive with technology research firm Gartner Inc. estimating the average cost of downtime, based on industry surveys, to be around 5600$ per minute [2].

As APM is not a clearly defined term, the function range of APM tools can differ widely, but at its core most of them offer some basic functionality like resource monitoring and architecture discovery supporting different platforms and programming languages [38]. While these features already provide value for customers, according to Gartner Inc., "the value proposition [...] has clearly shifted from data collection [...] toward data connection" [39], which means that functions like anomaly detection, and problem diagnosis in case of occurring failures are becoming more and more important.

When it comes to actual APM products, the most mature and feature-rich solutions, according to Gartner Inc., are commercial with companies like Dynatrace,[6] Appdynamics,[7] New Relic[8] and Data Dog[9] leading the field [39]. Besides that, there also exist several open-source products like Prometheus[10] or Apache SkyWalking.[11]

2.2.2 Reference Architecture

In this subsection, a high level overview of the architecture of a fictional APM system, including an explanation of its core components and functions, will be given. This presented architecture follows the reference architecture given by Rabiser et al. (2019) [40], which was developed by analyzing 47 monitoring systems and which shows that various monitoring approaches, despite their functional difference, exhibit significant conceptual overlap. Supplementary to this, additional information about the collection, processing and analysis of data done by APM systems is given.

The architecture of this APM system, depicted in Figure 2.5, can be divided into three main pillars. The left pillar denotes the monitoring setup, which is responsible for capturing the operational data, the middle pillar denotes the monitoring execution, which represents the heart of the monitoring system and consists of several layers with different fields of functions, and last, the right pillar denotes the monitoring support, which contains supporting functions, e.g. for data storage [40]. Depending on the mode of installation – on-premise or SaaS-based (software as a service) – either all or only part of the data processing happens on the IT system of the customer.

Monitoring Setup

The monitoring setup is responsible for capturing and transmitting different types of operational data to the monitoring execution. In most cases, this is done by deploying

[6] https://www.dynatrace.com/
[7] https://www.appdynamics.com/
[8] https://newrelic.com/
[9] https://www.datadoghq.com/
[10] https://prometheus.io
[11] https://skywalking.apache.org/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

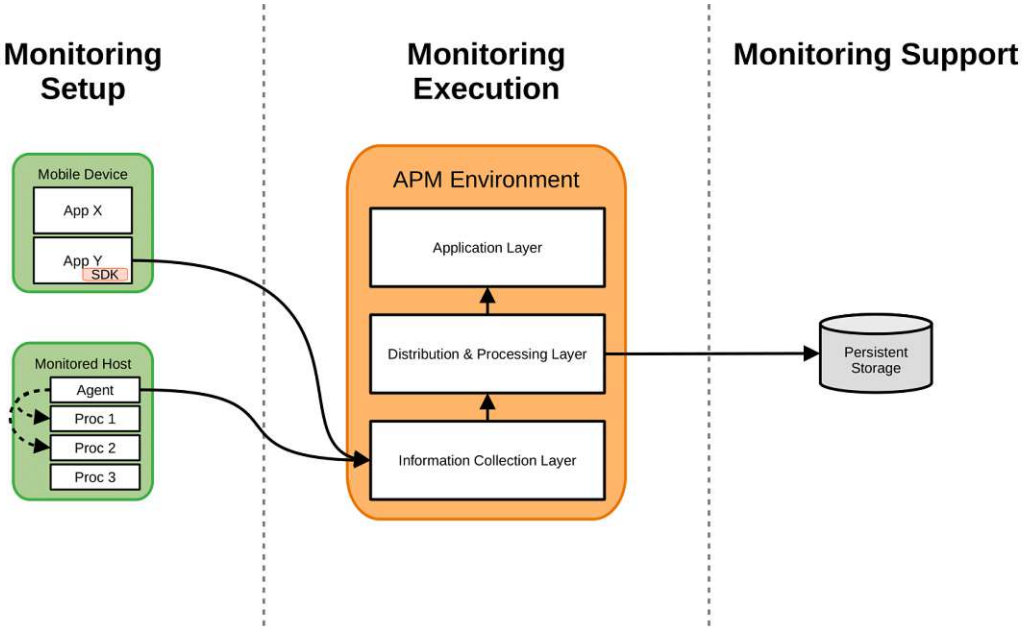


Figure 2.5: **Reference APM architecture** [40] – continuous lines indicating data flow, dashed lines indicating collection of stack traces

monitoring software – often called agents – onto the hosts of the customers, which allow the monitoring system to gather system and application logs as well as various performance and resource metrics of the host itself and of its running processes. For capturing more fine-grained information about application behaviour like stack traces, either automatic or manual software instrumentation is used, which often integrates painlessly with the already deployed agents. While each of the three types of mentioned operational data – metrics, stack traces and logs – can help to improve performance and reliability of applications, metric data is normally among the first candidates to look at to identify performance issues on a high level.

In the vast majority of cases, metric data is represented as time series and can be divided into two groups [41]. Application metrics mainly capture the current state and performance of an application with the two most common metrics being the response time and throughput. While response time describes the time for an operation like an application request, a database query or a file system operation to complete, throughput indicates how many user requests are handled per unit of time. Application metrics also sometimes extend to the client side, capturing end-user experience in more detail by measuring end-to-end response times of interactions or information about the usage of the user interface [38]. On the other hand, system metrics mainly capture low-level information of the underlying system like resource utilization. Classical examples include CPU or memory usage, IO or network throughput or less obvious measures like the number of used file descriptors.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

System metrics are often collected actively by periodic sampling, regardless of user interaction, while other metrics like end-to-end response time can only collected passively, when users interact with the application [38]. For actively collected metrics the sampling rate is crucial and requires careful consideration as the amount of data generated is strongly influenced by it. A high sampling rate provides a finer resolution and therefore more insight, but simultaneously increases the cost associated with transmitting, processing and storing the data. A lower sampling rate, on the other side, is more cost efficient, but may miss vital information when metrics indicate performance changes [41].

Regardless of the type or amount of data captured, monitoring setups are in most cases highly distributed as they need to collect data on multiple levels from different systems, which could become even more apparent with the rise of micro-service architectures.

Monitoring Execution

The monitoring execution represents the core part of the monitoring reference architecture, which can be further divided into three separate layers. On the lowest rank is the information collection layer responsible for collecting data from the monitoring setup, next is the middle layer with its main responsibility of transforming and analyzing the collected data and on top is the application layer, which contains various functionality, often not directly related to the monitoring or data analysis, but still relevant for the end user, like visualization and alerting [38].

The main responsibility of the information collection layer besides data collection is filtering and aggregating, which could be needed to reduce the amount of data to transmit and process or to convert the collected information into a more adequate form [38]. The processed data can then be passed along to the distribution and processing layer, which can be seen as the point where data is turned into valuable insights. While APM solutions can have widely different analyzing capabilities, one of the first steps for most solutions involves some form of checking or anomaly detection whether the system behaves as intended. For logs or stack traces this can be done by looking for deviations in log respectively call patterns compared to the expected state or in the case of metric data, techniques like change point detection or outlier detection can be applied. Normally, if a deviation from the expected state is detected, further investigations are initiated to identify the root cause of the anomaly – most commonly referred to as root cause analysis (RCA) – a topic, which will be discussed in more detail in Section 2.3. A side effect of RCA in addition to detecting the root cause, which is essential for remediation of the working system state, is that multiple detected anomalies belonging to the same cause, can be grouped and displayed as a single issue.
In a last step, the collected data among with information derived within the distribution and processing stage is forwarded to the application layer. Here, similar to the analyzing phase, the capabilities between different APM solutions can differ widely, ranging from basic functionality like visualization and alerting to more advanced features like automatic system adaption to minimize the impact of performance issues [38].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Monitoring Support

The last part of the reference architecture is the monitoring support with its main role of providing persistent data storage, be it in the form of a simple file-based solution or in the form of a more complex distributed database setup. Information to be stored is forwarded by the processing component, typically already filtered and aggregated to some extent to minimize the amount of storage needed. As the stored information usually becomes less important over time, some APM solutions further aggregate the data or remove it completely after some time.

2.3 Root Cause Analysis

Root cause analysis (RCA) with its objective to identify the root causes of observed problems is, at its core, strongly connected to the concept of causality [42]. It is a method of problem solving applied in many different disciplines as the understanding of and the remediation after an occurred failure is always of great importance, be it a business or technical problem. In the realm of software and IT infrastructure monitoring the perspective is much narrower with the most central questions being: (1) How to detect anomalous behaviour and (2) how to identify the root cause of observed anomalies [4, 41]? While the first question is not directly related to root cause analysis, it is a fundamental prerequisite and therefore strongly linked to it as will be discussed further below.

The rest of this section is divided as follows: The first part discusses root cause analysis more generally, focusing on relevant terminology and universal aspects of the process itself. In the second part, root cause analysis methods are discussed in more detail including how they can be differentiated and how many of them function on a high level with a closer focus on metric-based RCA methods.

2.3.1 General Concepts

This subsection introduces relevant terminology, similar to Subsection 2.1.1, which will be important throughput the rest of this thesis. After that, root cause analysis will be reviewed in more detail focusing on how the performance of RCA methods can be evaluated. Last, several trade-offs relevant in addition to the performance are discussed.

Terms and Definitions

A **software application** is often composed of multiple **services**, each performing its distinct actions while interacting with each other. A typical example would be a simple E-commerce website, which is made up of a process providing the customer front-end, a process handling back-end tasks like orders and payments and a database storing product and order information. A **failure** within such an application denotes the inability of a service to perform its function as expected while **anomalies** correspond to the observable symptoms of such failures. These can be further divided into **functional anomalies**, e.g. in the form of more generated log messages or varying call patterns and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

performance anomalies, e.g. in the form of an increased response time or higher CPU usage. Accordingly, **anomaly detection** refers to the process of detecting anomalies that affect an application or one of its services. In the case anomalies have been observed, the goal of **root cause analysis** (RCA) is to identify the reasons behind these anomalies by analyzing operational data captured while the application was running, without trying to replicate the failure or rerun the application. Both anomaly detection and root cause analysis mainly work with three types of data: (1) **metric data** capturing application state or resource utilization, (2) **log data** capturing runtime or system information and (3) **distributed stack traces** capturing method calls [4].

Assessment Criteria for RCA Methods

In the realm of software monitoring, the problem of both anomaly detection and root cause analysis can, in a wider sense, be interpreted as a classification problem. In the first case, the objective is to classify application services as a whole or individual information snippets – single metrics, log messages or traces – into two groups, anomalous and regularly functioning entities. In the second case, the objective is similar as the goal of RCA is to identify the set of entities that correspond best to the seen anomalies, i.e. the set of previously selected anomalously behaving entities. Based on this notation, one way of evaluating the performance of root cause analysis approaches is to use metrics like precision and recall or, in case of ranked results, a measure of ranking quality like discounted cumulative gain (DCG).

Similar to other engineering problems, designing a root cause analysis system involves several performance related trade-offs, some of which should be mentioned here. The first one is setup cost with tracing-based and monitoring-based techniques requiring more setup compared to log-based approaches as software processes need to be instrumented or agents for monitoring metrics need to be initiated. The second trade-off concerns the time it takes to identify the root cause, where typically two broad classes can be distinguished: *real-time analysis*, in which run time is critical and *post-mortem analysis*, in which run time is less of a concern. In most cases though the differentiation is often not that clear, as real-time algorithms often utilize precomputed information in their analysis [42]. The last trade-off involves the question how to deal with the many cases, in which the root cause can not unambiguously be determined. Returning multiple possible root causes, including several false positives increases the effort on the application operator, who needs to look into entities not responsible for the observed anomalies and thereby driving up the cost of remediation. On the other hand, returning only one root cause, potentially leading to false negatives, can result in even greater cost as not returning the actual root cause defeats the purpose of running some form of root cause analysis in the first place. As a result of this, many root cause analysis approaches opt to return multiple possible root causes [4].

One way to mitigate the problem of false positives is to provide some form of explanation together with each identified root cause [4, 42]. While this does not reduce the set of possible root causes, a human operator could nonetheless benefit from it by being able to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

exclude false positives as a result of the explanation. In the case of rule-based learners, an explanation could be a list of all rules that fired to reach a certain conclusion [42], while in the case of causal discovery methods it could be a list of paths from the identified root cause to the anomalous services or metrics respectively.

2.3.2 Anomaly Detection And Root Cause Analysis

Although anomaly detection and root cause analysis solve two distinct problems they are often strongly coupled as the output of anomaly detection is in most cases part of the input of the RCA process, which requires some form of compatibility. As a result of this, in a number of cases the two processes form some kind of pipeline instead of being triggered independently of each other [4]. Due to their close relation, the next paragraph covers anomaly detection on a high level, before addressing the process of root cause analysis in more detail.

Anomaly detection techniques can be differentiated according to several criteria including but not limited to the type of data used, the type of anomalies detectable and their overall functioning. Based on a recent survey by Soldani and Brogi (2022) [4] the majority of anomaly detection techniques fall within the scope of either unsupervised or supervised learning with the most commonly used data types being metric data and distributed stack traces. As one could anticipate, the majority of metric-based techniques can only detect performance related anomalies, while for stack trace or log-based methods the ratio between functional and performance anomalies detectable is much more balanced. Despite their differences all the surveyed techniques use data collected in training runs to build a baseline, which then can be compared to new data to detect anomalies [4].

Root cause analysis techniques can be differentiated in a similar way as described above and the findings, based on the same survey, can be summarized as follows: More than 70% of all examined RCA methods use metric data, several methods use distributed stack traces and only one method uses log data with the vast majority of all methods being only able to detect performance anomalies. While the precise functioning of the surveyed root cause methods differ quite substantial, for most methods it involves the automatic construction of a graph representation of the application that is then used for the inference of the root cause. Hereby, the survey authors differentiate between topology graphs, which are constructed using monitored service interactions or distributed traces, and causal graphs, which are constructed using application logs or metrics collected from the individual services of the application [4].
There exist, of course, also numerous other RCA techniques that deviate from the functional principle just described, but listing and describing these techniques is beyond the scope of this work. An overview of various alternative approaches can be found in the works of Ibidunmoye et al. (2015) [41] and Solé et al. (2017) [42].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Metric-based Root Cause Analysis

In the case of metric-based root cause analysis, the nodes of the graph can either model the application services or the metrics itself and directed edges between nodes indicate that a service influences another service or a metric influences another metric respectively. The root cause search itself can then be performed by traversing this graph, either starting from the metric nodes, where anomalies were detected or from business critical metrics like the end-user response time measured at the application front-end. Alternatively the centrality of nodes in the causal graph can be computed and ranked, assuming that the most central nodes are responsible for the observed anomalies [4].

An obvious idea for constructing a graph representation of an application is to use some form of correlation to detect interdependencies different metrics and thereby also between the different components of the application. While this approach is simple to understand and implement, correlation-based methods greatly suffer from spurious correlation and tend favour entities with high correlation, while the actual root cause can be easily overlooked. To avoid these problems, many RCA systems [5–10] resort to causal discovery methods for constructing the causal graph with the most common algorithm being the independence-based PC algorithm [21, 25].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Work

This chapter provides an overview of existing literature that has a considerable thematic overlap with this master thesis or has influenced the creation of it in a meaningful way. While the first section covers literature on the topic of causal discovery with a clear focus on time series data, the second section is concerned with the topic of root cause analysis in the field of application performance monitoring.

3.1 Causal Discovery

Literature on the topic of causal discovery is very diverse as the topic can be considered from both a highly theoretical as well as from a more practical point of view with this section definitely focusing on the latter aspect. Furthermore, since one of the main objectives of this work is the comparison of different types of causal discovery methods, the emphasis in this section is on literature that does the same, i.e. comparing different approaches to causal discovery.

Glymour et al. (2019) [43] give an introduction and brief review of causal discovery, covering independence-based, score-based and noise-based approaches including many illustrations how these methods work. Furthermore they highlight practical issues of causal discovery and offer general guidelines when applying these methods to solve actual problems.

Moraffah et al. (2021) [11] provide a comprehensive review of two causal inference tasks including causal discovery for time series data. Within their work, they focus on independence-based and noise-based methods, Granger Causality and also cover methods based on deep learning. Furthermore they curate a list of commonly used evaluation metrics and datasets used for causal discovery and provide information on how they can serve as a benchmark for future research within the field.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Different to Glymour et al. (2019) [43] and Moraffah et al. (2021) [11], which address different approaches to causal discovery from a more qualitative and theoretical perspective, Lawrence et al. (2021) [12] exclusively focus on practically evaluating and comparing several up-to-date causal discovery approaches for time series data. For this, they design a flexible and simple to use framework for generating synthetic time series data aiming to overcome the limitations of evaluating causal discovery methods under different circumstances using static data sets, which are inflexible by nature.

The most recent review of causal discovery for time series found is composed by Assaad et al. (2022) [13], which extensively examine the topic from both a theoretical and practical perspective. Within their work they give a detailed description of the underlying concepts and assumptions behind independence-based, score-based and noise-based methods, Granger Causality and also cover more exotic approaches to causal discovery not mentioned so far. Furthermore, they compose a summary of the main characteristics of eleven representative algorithms from the different method families just mentioned and carry out an experimental comparison evaluating most of these methods on both an artificial and a real data set.

Although the works of Lawrence et al. (2021) [12] and Assaad et al. (2022) [13] also experimentally evaluate and compare causal discovery algorithms based on different concepts similar to this work, they either use only synthetic data or data from a completely different area of application for their evaluation.

Besides research literature, which consciously focuses on reviewing and discussing causal discovery for time series holistically, there also exist a considerable amount of research which focus on specific approaches to causal discovery. Commonly, the authors of this type of research develop entirely new or enhance existing algorithms and as part of their work compare them to already existing ways of modelling time series data regarding their functioning and performance.

Arnold et al. (2007) [29] examines a host of related algorithms, which extend the original bivariate idea of Granger Causality [28] to multivariate data. For this they compare and characterize the performance of five different algorithms including a simple vector autoregression model from multiple viewpoints using both synthetic and real world data. Runge et al. (2019) [27] propose the novel causal discovery algorithm PCMCI [27] based on the independence-based PC algorithm [21, 25] and compare its performance to other simpler approaches using different data sets. First, they evaluate their algorithm and a simple correlation approach qualitatively on real world data sets from the field of climate science. Second, they compare multiple algorithms including their own algorithm, the PC-algorithm [21, 25], a simple vector autoregression model, classical lasso regression [30] and a simple correlation approach using high dimensional synthetic data. Similarly, in the paper in which PCMCI+ [26], an extension of the PCMCI [27] algorithm, is proposed, it is compared to a version of the PC-algorithm [21, 25], VARLiNGAM [36] and to a hybrid approach combining the notion of Granger Causality with the PC-algorithm [21, 25] using synthetic data. Pamfil et al. (2020) [33] propose the novel score-based causal discovery

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

algorithm DYNOTEARS [33] and compare its performance to three other algorithms, which include one noise-based and two hybrid approaches using synthetic data.

The thematically most similar work to this master thesis is the published work by Arya et al. (2021) [44], in which they compare several causal discovery algorithms for the task of root cause analysis. Nonetheless, their work differs from this thesis regarding several aspects. First, within their work they only examine causal discovery algorithms based on Granger Causality and conditional independence testing. Second, instead of using classical resource and performance metrics, they model log data collected from their running application as error rate time series and use these metrics as input for the to be examined algorithms. Third, within their work they only consider a single type of software failure, in which one of the micro-services of the application is completely shut down, compared to the four different software failures considered in this work.

3.2 Root Cause Analysis in Application Performance Monitoring

This section provides relevant references to literature on the topic of root cause analysis (RCA) within the domain of application performance monitoring and it is structured as follows: First, multiple survey paper covering different aspects of root cause analysis are presented. After this, the focus is on research literature describing RCA systems that use graph structures to perform the root cause analysis and which exclusively or mostly use metric data to operate.

The survey papers listed here offer different perspectives on the subject of root cause analysis within the domain of application performance monitoring.
Ibidunmoye et al. (2015) [41] provide a summary on the topic of anomaly detection and root cause analysis techniques with a clear focus on performance anomalies. For this, they describe how performance anomalies manifest themselves and point out the common sources of them. Furthermore they curate an extensive list of research literature that addresses the problem of automatically identifying the root cause of observed performance anomalies and categorize their findings based on the detection strategies used in the references found.

Solé et al. (2017) [42] present an elaborate survey of models and techniques for root cause analysis with an emphasis on the performance and scalability of the evaluated methods. As part of their work they create a detailed classification of different RCA models as well provide a summary of learning algorithms that automatically construct the RCA model with little or no domain knowledge. Furthermore, they define a list of dimensions to classify the surveyed algorithms in terms of their inference capabilities and compile a summary which uses these dimensions to highlight the individual strengths of weaknesses of all the examined algorithms.

Soldani and Brogi (2022) [4] provide a qualitative analysis of methods and techniques for anomaly detection and root cause analysis focusing on modern multi-service applications

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

exclusively. Within their work, they classify the evaluated algorithms along multiple dimensions including but not limited to the type of data used, the underlying method and the type of anomalies the algorithm can detect.

Apart from survey literature on the subject of root cause analysis, of particular interest for this work is literature on RCA systems, which involve the automatic creation of a graph-structure-based RCA model as an important part of their root cause inference process. Many of the following references can be found in the previously mentioned survey literature and in particular in the work of Soldani and Brogi (2022) [4].

The RCA systems MS-Rank [5, 6], AutoMAP [7] and CloudRanger [8] all use some version of the PC algorithm [21, 25] to build a graph structure for inference, but different to this work, nodes in these graphs correspond to services and not individual metrics. Systems that are closer to the graph structure used in this work, in which graph nodes correspond to individual metrics, are MicroCause [9] and LOUD [10]. These two systems also differ from the previously mentioned systems with regard to the causal discovery algorithm used, as MicroCause uses the PCMCI algorithm [27] and LOUD uses a Granger statistical test to construct the graph structure.

So far, all cited RCA systems exclusively use metric time series data for the construction of the inference graph, but there also exist hybrid approaches utilizing additional data sources. The RCA system Sieve [45] constructs the inference graph structure by first collecting stack traces and based on this information choosing service components that should be compared. After that, representative metrics are selected for each service, which are then used to test the possibility that representative metrics from one service influence representative metrics from other services using a pairwise Granger Causality test [28]. Different to this construction strategy, the RCA system CauseInfer [46, 47] creates a two layered hierarchical graph structure for inference. While, the higher layer connects individual services by analyzing traffic lag correlation between services, the lower layer connects individual metrics within a service by utilizing a version of the PC-algorithm [21, 25].

Most of the cited RCA systems use the same types of causal discovery algorithms, which may indicate that other approaches to causal discovery are either completely unknown or their usefulness for the task of root cause analysis is unknown. Regardless of the reason, it can be seen as confirmation of the importance of this work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

Study Design

This chapter states the research questions of this master thesis as well as describes the course of action taken to answer them. Its structure roughly aligns with the phases of the CRISP-DM process model [14], which was selected to be the framework of choice for answering the research questions presented in the first section. Based on this and subsequently, the second section focuses on understanding the data generating process and what kind of and how the data was collected. The third section covers the whole modelling facet, which includes the data preprocessing as well as the data modelling itself. The fourth section explains all aspects of the evaluation process including how and under which circumstances the implemented modelling approaches were evaluated. Last, potential threats to the validity of the presented study design are discussed to acknowledge factors that might have a negative effect on the research.

4.1 Research Questions

The aim of this work is to compare and evaluate existing statistical algorithms that fall under the category of causal discovery regarding their usefulness for the task of root cause analysis in the domain of application performance monitoring. More specifically, the following research questions (RQ) should be answered:

1. How well can causal discovery algorithms detect connections between the causes and the effects of software failures based on metric time series data alone? How successfully can these methods identify the software component that is causing the failure?
2. How well do these algorithms perform in comparison to two simple baseline approaches – one using the Pearson correlation and one using the graphical lasso – with regard to the tasks stated in RQ1?

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3. How sensitive is the performance of the examined causal discovery algorithms and the baseline approaches with regard to the start and end points of the metric time series data?

4.2 Process and Data Understanding

This section provides information about the application that is monitored, i.e. the data generating process, and about the collected data itself. First the application structure and how the application was deployed is explained. Second it is described how the application was monitored in this particular case and what type of data was collected. Last, the situation under which the operational data was collected is depicted in detail, which will be important later in the evaluation section.

4.2.1 Demo Application

To be able to collect monitoring data of an actual running software system, a demo application – called *easyTravel* – was used, which is developed by the APM company Dynatrace and is freely available online [48]. At its core, it is a multi-tier web application, which represents an imaginary web presence of a travel agency allowing users to search and book journeys to various destinations. In addition to its main functionality, it also provides tools for generating synthetic views and interactions with the web application and for activating problem patterns, which cause failures or performance issues within the web portal.

Application Structure

The web application follows a multi-tier architecture. The presentation tier consists of two web-servers and two front-end services, a classical one and an Angular[1] one. These front-end services communicate with multiple services from the application tier to handle all business logic like authentication, journey searches, bookings and their corresponding payments. The last level is the data tier containing the database, which is connected to several services from the application tier. A semantic view of the application can be seen in Figure 4.1, which however does not disclose how the individual components are actually deployed.

Deployment

The *easyTravel* instance used for collecting the operational data of the individual services was deployed in a distributed way to be able to handle higher synthetic loads compared to a deployment on a single host. The front-end services are deployed on four different hosts, each host containing both front-end versions. Most application tier services are all part of a single java process, expect for the credit card verification service, which is a C++ process. All processes are deployed four times on two separate hosts. Two load balancers are used, the first for equally spreading the user requests between the

[1] TypeScript-based free and open-source web application framework

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

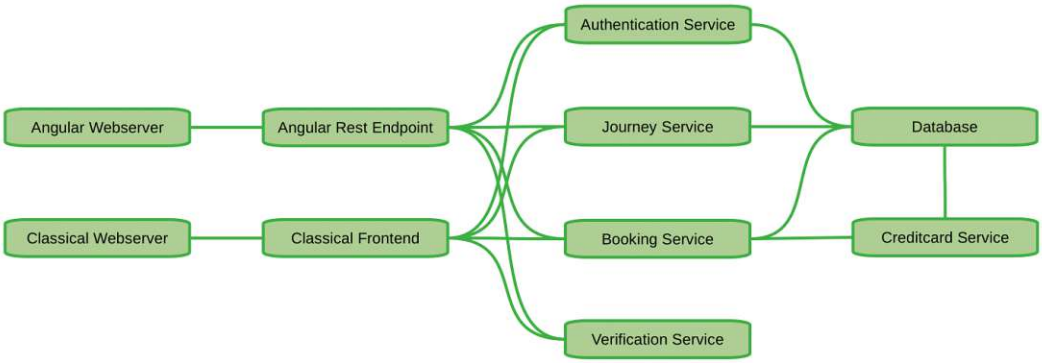


Figure 4.1: Schematic view of the *easyTravel* architecture

different front-end instances and the second for distributing the back-end requests from the front-end instances to the multiple application tier processes. A single Oracle database is deployed separately to handle all requests from the application tier.

4.2.2 System Monitoring

The capturing of operational data was done by utilizing the application monitoring capabilities of the APM solution Dynatrace, which includes automatic detection and grouping of running processes [49]. More precisely, Dynatrace detects processes performing the same function even across multiple hosts and groups them into *process groups*. Moreover, Dynatrace is able to further partition individual processes into distinct *services* if the processes serve multiple different network addressable locations. Similar to the grouping of processes, services that perform the same function are therefore grouped into merged services. This results in a total of ten services: two web-server services, two front-end services, five back-end services and one database service. Due to the grouping of identical functionality, the service based view is equivalent to the semantic view already presented in Figure 4.1.

The application monitoring itself is done by a single software process, called Dynatrace *OneAgent*, which is deployed once per host. After the setup process, it collects metric data from the operating system it runs on and from the processes which run on the deployed host. Furthermore, it is capable of collecting log files and software stack traces, but none of these features were used during the course of this work. All the collected data is sent to the Dynatrace server, where it is processed and made available for the APM users to consume.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.2.3 Metric Data Collection

The metric data for the evaluation was downloaded using the Dynatrace Metrics API on a per service basis. This means that each used metric is aggregated over all the running instances performing the same function. Furthermore, although the *easyTravel* application uses specific technologies like Oracle or Apache Tomcat (Java), only general, non-technology specific metrics were used for the evaluation to make the results as applicable as possible regardless of the actual technologies used. The metrics that were utilized during the course of this work are the CPU time, throughput, error rate and response time.

CPU Time

The CPU time is the amount of time that is used by the CPU to process the computer instructions to fulfill a given task and is measured in microseconds. It differs from elapsed time, which also includes any waiting time. The CPU time is aggregated using the sum across all running instances.

Throughput

The throughput is the number of requests completed and thus the rate of work performed within a time interval. It is aggregated using the sum across all running instances.

Error Rate

The error rate is the percentage of failed requests based on the total amount of requests within a given time interval. It considers all kinds of failed requests and does not differentiate between different types of failures. The error rate is calculated by the fraction of the sum of failed requests and the sum of overall requests across all instances.

Response Time

The response time is the total amount of time it takes to respond to a request and thus represents a measure how quickly individual requests are fulfilled. It is the sum of the time it takes to process the request including any waiting time related to this processing, e.g. loading data from disk, sending of requests to fulfill the original request. The 90% percentile of all response times across all running instances is used during the course of this work.

The throughput, the error rate and the response time are available for all services, while the CPU time is available for all services except the database. This means that in total 39 metrics were used across the ten services described above. Every metric represents an individual time series with a time interval, i.e. the time between two consecutive data points, of one minute.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2.4 Problem Patterns

The metric time series data was collected over periods of time where the demo application undergoes some form of performance degradation due to different software failures. In this work, multiple problem patterns [50], created by Dynatrace during the development of the *easyTravel* demo application, are used to replicate software problems real IT systems could experience in practice. These patterns are enabled through a separated interface and can be activated and deactivated during run time, thereby simulating an error-prone software deployment and a corresponding fix or roll-back.

For this work, four different problem patterns were used, each activated over a time frame of 30 minutes with sufficient time between the different patterns to avoid any kind of mutual interference. Furthermore, the patterns were triggered on all running instances to avoid any kind of imbalance due to the distributed deployment.

CPULoadJourneyService Problem

This problem pattern causes additional high CPU usage by triggering an inefficient and irrelevant computation every time a user searches for a journey [50]. It was designed to resemble problems in which faulty code changes result in much higher computational load for a frequently used functionality of the application. The problem causes an increase in CPU time and response time for the journey service itself and response time increases for the two front-end services and the two web-server services. The root cause service is the journey service.

DatabaseSlowdown Problem

This problem pattern slows down queries on a specific table within the database, which results in a latency increase for depended services [50]. It was designed to resemble problems where only certain tables or queries have performance issues while the rest of the database functions as expected. The problem causes response time increases for the database itself, the three back-end services, the two front-end services and the two web-server services. The root cause service is the database service.

DBSpammingAuthWithAppDeployment Problem

This problem pattern increases the workload of the database in two ways. First, it causes the business back-end to access the database unnecessarily when users are authenticated and second, it reduces the number of cached entries in order to increase the number of database requests [50]. The goal of this pattern is to resemble problems, where the database experiences a higher than normal number of requests, while the overall number of user requests stays the same. Although the root cause in this case could also be the authentication service, the defined root cause service will be the database service as the result of the high number of requests only becomes visible through the throughput metric of the database. The problem pattern causes an increasing response time for the database itself, the authentication service, the two front-end services and the two web-server services.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

AngularFailure Problem

This problem pattern[2] causes the Angular front-end service to return a HTTP 500 Internal Server Error for all requests related to the storage of bookings, thereby increasing the overall failure rate of the service. It tries to resemble problems, in which faulty code changes only affect part of the overall application, while most of the other services remain unaffected. This failure rate increase for the Angular front-end service also effects the failure rate of the Angular web-server service. The root cause service is the Angular front-end service.

The collected metric time series data of a single occurrence of an activated and deactivated problem pattern represents a multivariate time series and will be called **problem instance**. Each problem pattern was observed ten times, resulting in a total of 40 problem instances.

4.3 Modelling

This section explains the modelling process, which includes the data preprocessing and the actual data modelling itself. First, the structure of the time series data is described in detail, followed by a step-by-step explanation of the data preprocessing. After this, the modelling process itself is discussed generally first and then each implemented modelling approach is described in more detail. Last, the process of how parameter selection for the implemented approaches was done is described. For transparency and reproducibility reasons, the versions of all used packages are mentioned throughout this section. All implementation was done using python version 3.8 and the entire modeling and evaluation process was run on a 2019 16-inch MacBook Pro (2.4GHz 8-Core Intel Core i9, 32GB 2667 MHz DDR4).

4.3.1 Time Series Structure

Time series data of observed problem patterns was stored as tabular data. More specifically, each problem instance was saved in a *comma-separated values* (CSV) format with 39 columns representing the individual metrics of the ten services and an additional column containing minute resolution time information. The observation time frame for each problem instance spans two and a half hours – 150 points in time, which includes the 30 minutes, when the problem was activated and an additional hour before and after the problem itself. An illustration of the introduced structure is shown in Figure 4.2.

4.3.2 Time Series Preprocessing

Data preprocessing of the raw time series data is done in several steps. First, the metrics are grouped based on the service they belong to. Second, constant or almost constant columns are dropped due to the fact that the causal discovery method PCMCI+ throws

[2]The AngularFailure Problem is not part of the list of available problem patterns [50].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

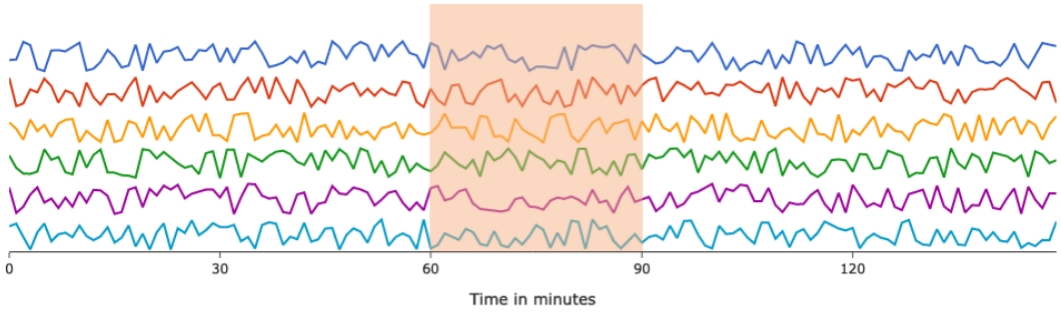


Figure 4.2: Illustration of time series data – shaded area highlighting the time frame during which problem patterns are activated

an error when dealing with constant time series data. This exclusively effects the error rate metric, which is only different from zero during the AngularFailure Problem. Third, each individual metric is scaled using the *scikit-learn StandardScaler* function to remove the mean and scale to unit variance. After this, the grouping per service is used to identify metrics with almost identical behavior, which is done by calculating the correlation among time series data of the same service. If the coefficient between two metrics is above 0.99, one of them is dropped as such a high correlation indicates that these metrics resemble redundant information, which contributes little to the process of causal inference and can even make the results unstable [47]. This happens repeatedly between the CPU time and the response time metric, which is plausible as the process time often can represent the majority of the time to process a request. Although the dropped metric is not used for the modeling step anymore, the information of its high correlation with the kept metric is saved as it is needed for the evaluation. The last step reverses the metric grouping per service, such that the output format of the preprocessing is identical to the format of the raw time series data. Besides the usage of the python package *scikit-learn* (version 0.24.2), all preprocessing was done using the python package *pandas* (version 1.4.0).

4.3.3 Causal Graph Modeling

In this work, causal graph modeling refers to the process of converting the metric time series data into a graph representation in which individual metrics represent nodes and edges between nodes should indicate linear causal influence or independence depending on their presence or absence. This modeling is done by applying algorithms, which take the preprocessed multivariate time series data, i.e. a single problem instance, as input and return a graph structure as their output.

While the structure of the input data was already explained in Section 4.3.1, the output structure is as follows: The output graphs are represented using the *DiGraph* class from the python package *networkx* (version 2.6.3) and consist of nodes and edges. Each metric, which is present after the preprocessing, represents an individual node. Depending on the results of the algorithm applied, three options are possible for every node pair. First, the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

algorithm found no connection, indicating independence, which means no edges are added to the graph. Second, the algorithm decided on a connection with a distinct direction, in which case a single edge is added to the graph. Last, the algorithm detected a connection, but the direction of influence remains uncertain. In this case, two differently oriented edges are added to the graph. A simplified output graph is depicted in Figure 4.3.

In this work, six different approaches for causal graph modeling are examined. Four of these approaches are the causal discovery methods, which were already introduced in Subsection 2.1.3 from a theoretical point of view. Following this, practical details about their implementation as modeling algorithms are presented now. In addition to the four causal discovery methods, two naive approaches – one using the Pearson correlation and the second one using the graphical lasso – are explained in more detail as well. These two approaches act as baseline methods as both the Pearson correlation and the graphical lasso are regularly used to measure linear relationships and are in general well known.

PCMCI+

The first method is the PCMCI+ algorithm from the python package *tigramite* (version 4.2.2.1). The modeling is done by using the PCMCI object method *run_pcmciplus* with the conditional independence test *ParCorr*, which uses a *Student's t* test internally. The two relevant outputs for this work are a string array, which indicates directed causal links using $-->$ and undirected connections using $o-o$ or $x-x$ and a matrix quantifying the strength (not the directionality) of the found connections. This output is converted to the output structure explained earlier during which links with an absolute link weight below a certain threshold are removed. The implemented approach has three parameters: the maximum data lag considered, the threshold and a significance parameter for the independence test.

Granger Lasso

The second method is the Granger Lasso, which uses the implementation of the identically named *granger_lasso* method from the python package *causal-learn* (version 0.1.2.8) with only a single adjustment. Instead of using the *LassoCV* function, which uses cross validation to find the best model, the reimplementation uses the standard *Lasso* function for the estimation of coefficients. Both functions are part of the python package *scikit-learn* (version 0.24.2). These coefficients are interpreted as causal influences between the time series and lagged versions of all time series and are converted to the required graph structure accordingly. Edges with an absolute coefficient below a specified threshold are removed. The implemented approach also has three parameters: the maximum data lag considered, the threshold and the regularization parameter for the lasso function. Different to the other causal discovery methods, the minimum data lag is one, not zero, meaning that no contemporaneous links can be detected.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

DYNOTEARS

The third method is the DYNOTEARS algorithm from the python package *causalnex* (version 0.11.0). The modeling is done by using the function *from_pandas_dynamic*, which returns a *StructureModel* – a graph object used throughout the mentioned python package. This output is then converted to the needed output structure explained before. The implemented approach has four parameters, all of which are directly used within the modeling function of the python package. $\lambda_{\boldsymbol{W}}$ and $\lambda_{\boldsymbol{A}}$ are the L1 regularization parameters for the contemporaneous and lagged edges respectively. The other two parameters are the threshold to remove edges and the maximum data lag considered, both equivalent to the same parameters in the already explained methods.

VARLiNGAM

The fourth and last causal discovery method is the VARLiNGAM algorithm from the python package *lingam* (version 1.6.0). The modeling is done by using the VARLiNGAM object method *fit*. In addition, two parameters were set during the initialisation of the object itself. First, the criterion parameter was set to *None* such that results across all lags are returned. Second, the prune parameter was set to *True*, which results in a sparser output by the VARLiNGAM method itself. The output itself is a numeric matrix quantifying the strength of the found connections. This matrix is converted to the required output structure during which links with an absolute link weight below a specified threshold are removed. The implemented approach has two parameters: the maximum data lag considered and the threshold.

Pearson Correlation

The first naive method uses the Pearson correlation to derive connections among the individual time series of the input data, which is done in three separate steps. First, the original input data is expanded by shifted versions of itself, each shifted version representing a different time lag up to a maximum number of lags. Second, the pairwise correlation between every time series of this expanded data frame is calculated. Last, the resulting numeric matrix containing the correlation coefficients for every time series pair is converted to the required graph structure during which links with an absolute link weight below a specified threshold are removed. For both, the shifting and the calculating of the Pearson correlation, the python package *pandas* (version 1.4.0) is used. The implemented approach has two parameters: the maximum data lag considered and the threshold.

Graphical Lasso

The second naive method uses the graphical lasso to derive connections among the input time series data. Its implementation is almost identical with the one using the Pearson correlation. After the original input data is expanded, the *GraphicalLasso* function from the python package *scikit-learn* (version 0.24.2) is used to estimate the precision matrix and subsequently calculate the partial correlation coefficients, which measure the association between two variables while controlling for all the other variables [51]. This

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

differs from the correlation approach, which does not control for other variables. The last step, the conversion from the partial correlation coefficient matrix to the required output structure, is done in the same way as for the first naive method. The implemented approach has three parameters: the maximum data lag considered, the threshold and the regularization parameter for the graphical lasso function.

4.3.4 Parameter Selection

The parameters for the six approaches were selected using hyperparameter tuning. For this purpose the 40 problem instances were split into two groups, each group containing the same amount of instances from each of the different problem patterns. The ratio between the two sets was 30/70 meaning 12 instances were used for the hyperparameter tuning and 28 instances for the final evaluation. For the hyperparameter tuning itself the python package *optuna* (version 2.10.0) was used, which works by specifying the objectives and their optimization direction, i.e. if the objective should be maximized or minimized. In this case, *recall* – the number of relevant causal links in the graph – was maximized and *density* – the percentage of overall links within the graph – was minimized. Both criteria are explained in more detail in Section 4.4. This multi-objective optimization lead to a Pareto frontier for each approach with higher densities, i.e. a higher number of overall links, resulting in higher recall values. Because all approaches should be compared based on a similar density level and not all methods were able to produce results for higher densities, the method parameters of each approach were determined by the hyperparameter tuning run with the highest recall with a density between 4% and 7%. The general method parameter *maximum data lag considered* was set to 2 regardless of the hyperparameter tuning results to further increase comparability among the different approaches and because the hyperparameter tuning results showed that its importance for both objectives was lower than 1% for all methods and even much lower than that for most methods.

4.4 Evaluation

This section discusses the process of evaluating the implemented modelling approaches from various angles and under different circumstances. The first part explains in detail how each individual problem instance is evaluated, while the second part of this section states the different evaluation scenarios investigated to be able to examine the performance of the modelling algorithms under different circumstances.

4.4.1 Evaluation Criteria

To evaluate the presented causal graph modelling approaches from various angles, several different evaluation criteria were defined. While the first criteria evaluates the implemented algorithms, the four other criteria assess the output graphs generated by the modelling approaches.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

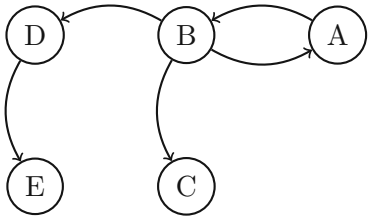


Figure 4.3: Simplified example of an output graph

From these four criteria, the first two criteria can be determined without knowledge of the problem patterns while the other two rely on information about the root cause service and the affected metrics, which can be found in Section 4.2.4. Furthermore, for all graph-based evaluation criteria an example on the basis of a simplified output graph is given.

Run Time

The first criteria is the run time of the implemented causal graph modelling approach in seconds. It represents the total CPU time from the call of the implemented approach until the final output graph is returned and thus does not include the time for preprocessing the time series data nor the time for evaluating the output graph. Although all approaches are implemented in python, they all use various low-level routines for more computational intensive calculations. As it is unclear to what extent the developers of the used libraries utilize these more efficient routines, the run time should not be taken at face value, but instead be seen more as a rough estimate of their computational effort. All else being equal, short algorithmic run times are preferred over longer run times as faster graph construction results in an overall faster RCA process.

Density

The second criteria is the density, which is defined as the percentage of all directed edges E in the output graph $\mathcal{G} = (V, E)$ compared to a fully connected directed graph with the same number of nodes V.

$$density = \frac{E}{|V| \cdot (|V| - 1)}$$

To give a concrete example, Figure 4.3 depicts a simplified output graph, which has a density of 25%. Similar to the run time, low densities are preferred over high densities all else being equal as less dense graphs can be analyzed faster, which results in an overall faster RCA process.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Unambiguously Directed Edges

The third criteria is the percentage of unambiguously directed edges among all directed edges E in the output graph $\mathcal{G} = (V, E)$. Here, unambiguously directed edge refers to an edge (u, v) with $(u, v) \in E$, but $(v, u) \notin E$. To give another concrete example, the graph depicted in Figure 4.3 has 60% unambiguously directed edges, because three out of the five edges in the output graph do not have an oppositely facing equivalent. While this criteria indicates the graph modelling algorithm's ability to orient edges, it does not provide any information about the correctness of the orientation.

Rootcause Rank

The fourth criteria is the placement of the root cause service, i.e. the service that actually caused the failure, along a ranking of all the ten services with a placement of one indicating that the graph modelling algorithm identified the actual root cause service as the most likely root cause and ten as the least likely root cause. The ranking itself is created by sorting the individual services according to the fraction of outgoing edges, which connect metrics of the service of interest to metrics from all the other services. The idea behind this is that in the case of a severe problem, the metrics of the root cause service affect many other metrics – indicated by outgoing edges – and therefore the service is ranked higher. It is calculated by the *group_out_degree_centrality* function from the python package *networkx* (version 2.6.3). Referring to the graph in Figure 4.3 again and assuming that every metric from A to E represents its own service would yield the following ranking: 1. (B), 2. (A,D), 3. (C,E). Under the additional assumption that A is the root cause service, the root cause rank would be 2.

Recall

The last criteria is the recall, which measures the percentage of predefined causal paths found in the output graph for a specific problem instance. Here, a causal path refers to a path that starts at any metric of the root cause service and reaches an anomalously behaving metric not belonging to the root cause service with a maximum of one unspecified node in between. This means that the number of relevant paths which can be potentially found for each problem instance depends on the problem pattern which was active. The root cause service and the affected metrics that are the basis of the predefined causal paths are explicitly mentioned in Section 4.2.4, but in summary the CPULoadJourneyService problem has four causal paths, the DatabaseSlowdown problem has seven causal paths, the DBSpammingAuthWithAppDeployment problem has five causal paths and the AngularFailure problem has one causal path. Referring to the graph in Figure 4.3 and assuming that the metric A belongs to the root cause service and C, D and E are anomalously behaving metrics not belonging to the root cause service, the recall would be 67%, as the path A to E has two unspecified nodes in between and is therefore considered not valid. Last, while the recall criteria is primarily meant for determining the presence of predefined causal paths in the original directed graph produced by the implemented approaches, it can also be determined for two modified versions of the original output graph. These versions are called undirected graph type

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and edge reversed graph type and are determined as follows: The recall for the undirected graph type refers to the percentage of predefined causal paths found in the undirected version of the original output graph. Likewise, the recall for the edge reversed graph type refers to the percentage of predefined causal paths found in the original output graph, where the direction of all edges was reversed.

4.4.2 Evaluation Scenarios

Each problem instance was modelled by each of the implemented approaches and then evaluated based on the above defined criteria. This just described procedure was carried out three times under different circumstances to emulate different scenarios under which a potential RCA algorithm using one of the causal graph modelling approaches could be used.

Broad Symmetric Time Window

The first scenario considers a time window of 90 minutes of the collected time series under which the demo application functions normally for the first 30 minutes, then operates 30 minutes during which a problem pattern is active and then functions normally again for another 30 minutes. Referring to the illustration of the time series data in Figure 4.2 the analysis time frame starts at minute 30 and ends at minute 120. This broad time window around the actual problem period is used to resemble a *post-mortem analysis* in which the start and the end of a detected problem is not clear and thus the analysis time frame is chosen wider as actually needed.

Shrinking Symmetric Time Window

The second scenario also uses a symmetric time window around the actual problem period similar to the first scenario, but here the total time window is progressively reduced from initially 90 minutes to 30 minutes. This means that in case of a total time window of 50 minutes the normal operating time window only spans the first and last ten minutes and in the end the total time window only consists of the 30 minutes during which a problem pattern is active. With this procedure a *post-mortem analysis* should be imitated in which the start and end of a detected problem is known more precisely and thus the analysis time window can be chosen narrower.

Expanding Time Window

The last scenario uses a expanding time window that starts with a time frame of 30 minutes in which the demo application functions normally and is progressively expanded to include more and more of the data points during which a problem pattern is active. More specifically and referring to the illustration of the time series data in Figure 4.2, the analysis time frame starts at minute 30 and is gradually extended until it spans 90 minutes from minute 30 to minute 120. This situation should resemble a *real-time analysis* in which a problem just started and is slowly unfolding.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.5 Threats to Validity

The study design explained in this chapter has been based on surveyed literature and was adjusted several times to make the results and the conclusions drawn from it as valid and generally applicable as possible. Nonetheless, not every aspect of one's work can be controlled for and thus in this section multiple concerns that could pose a potential threat to the validity of this work should be addressed. Thereby, three different types of validity threats – internal validity, construct validity and external validity – are considered and differentiated [52].

Internal validity: The collected time series data is aggregated on an one minute basis, which means that any changes of the operational state of the application will become visible only with a certain time delay in the captured metric data. Furthermore, all considered methods are only able to detect linear relationships and therefore any highly non-linear connections will most likely not be detected. While both aspects certainly have an effect on the detection performance of all implemented causal graph modelling approaches, in case that the impact on them differs, this could shift the obtained results.

Construct validity: With the proposed evaluation criteria and scenarios it is tried to answer the posed research questions as comprehensively as possible. Nonetheless, it can not be ruled out that a specific aspect for utilizing one of the implemented methods for the use case of root cause analysis was not covered or potentially was interpreted incorrectly, and thus the drawn conclusions would need to be adjusted.

External validity: Although the here used multi-tier web application represents a commonly used type of software system, software applications in general can serve all kinds of purposes and therefore can exhibit wildly different behaviour compared to the ones observed in this work. Furthermore, even between software applications that serve the same type of purpose, aspects like their internal realization and their way of deployment can result in differences. This means that the results and conclusions reached in this work may only be valid for certain types of software systems.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Study Results

This chapter presents the results obtained during the execution of the study design described in Chapter 4 and is structured according to the three evaluation scenarios outlined there. The first section shows the evaluation results of all implemented approaches assessed across all predefined criteria in order to provide the most comprehensive overview of their individual strengths and weaknesses. After that, sections two and three focus exclusively on the results obtained during the evaluation of all the methods regarding the two most important performance criteria – recall and rootcause rank – under different circumstances. The results are presented in the form of plots and tables and are accompanied by written descriptions of what is presented. Within this chapter, when speaking of all problem instances, it refers to the 28 problem instances – seven instances per problem type – used for the final evaluation.

5.1 Scenario 1: Broad Symmetric Time Window

The results presented in this section show the performance of the implemented approaches for the first examined evaluation scenario, which should resemble a *post-mortem analysis* in which the start and the end of a detected problem is not exactly known beforehand.

Plot 5.1 incorporates information on three out of the five evaluation criteria. The y-axis shows the average recall across all problem instances with a high recall value indicating that, on average, a high percentage of predefined causal paths were found within the output graphs produced by the implemented approaches. The x-axis shows the average density across all problem instances with a low density value indicating that on average the output graphs produced are sparse compared to a fully connected graph. The scale of the average run time of each of the implemented approaches is represented by the size of the data points in order to be able to put all these three criteria into perspective.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

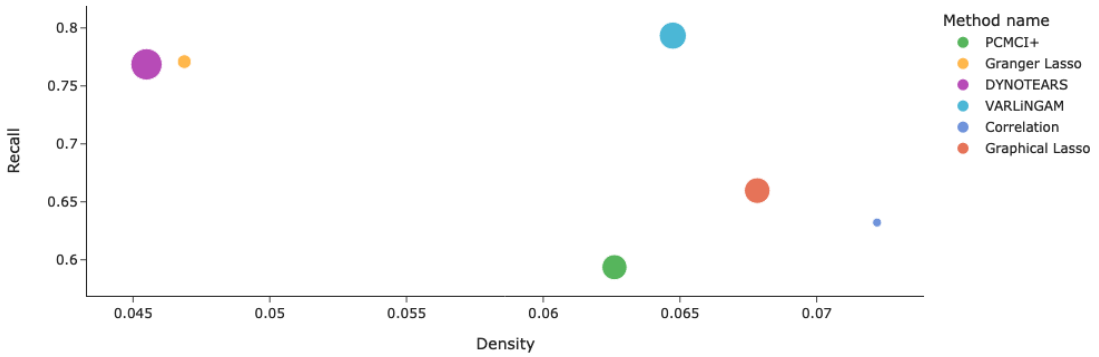


Figure 5.1: Average recall and density – size reflecting run time

One can see that the three causal discovery algorithms DYNOTEARS, Granger Lasso and VARLiNGAM outperform the two naive approaches both in terms of higher recall and lower density values. The fourth causal discovery algorithm PCMCI+ has a lower average density, but also a lower average recall compared to the naive approaches. The run time of all implemented approaches will be discussed in more detail below, but it can already be noted that the best performing method DYNOTEARS also has the highest run time and the most naive approach using the Pearson correlation has the lowest run time among all examined methods. The exact values for the three criteria can be found in Table 5.1.

Method	Recall	Density	Run time (sec)
PCMCI+	0.594	0.063	3.136
Granger Lasso	0.771	0.047	0.016
DYNOTEARS	0.769	0.045	217.145
VARLiNGAM	0.793	0.065	11.179
Correlation	0.632	0.072	0.004
Graphical Lasso	0.660	0.069	4.554

Table 5.1: Average results for the evaluation criteria recall, density and run time

As already explained in Section 4.4, the recall measures the percentage of causal paths found in the output graph and since the number of causal paths that can be potentially found depends on the problem pattern which was active, Plot 5.2 shows the average recall and density for each problem pattern separately.

For the AngularFailure problem, both naive methods have the highest recall, but not the lowest density values. The two causal discovery algorithms DYNOTEARS and Granger Lasso only have a slightly lower recall, but similar density values compared to the Pearson correlation approach. For the CPULoadJourneyService prob-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

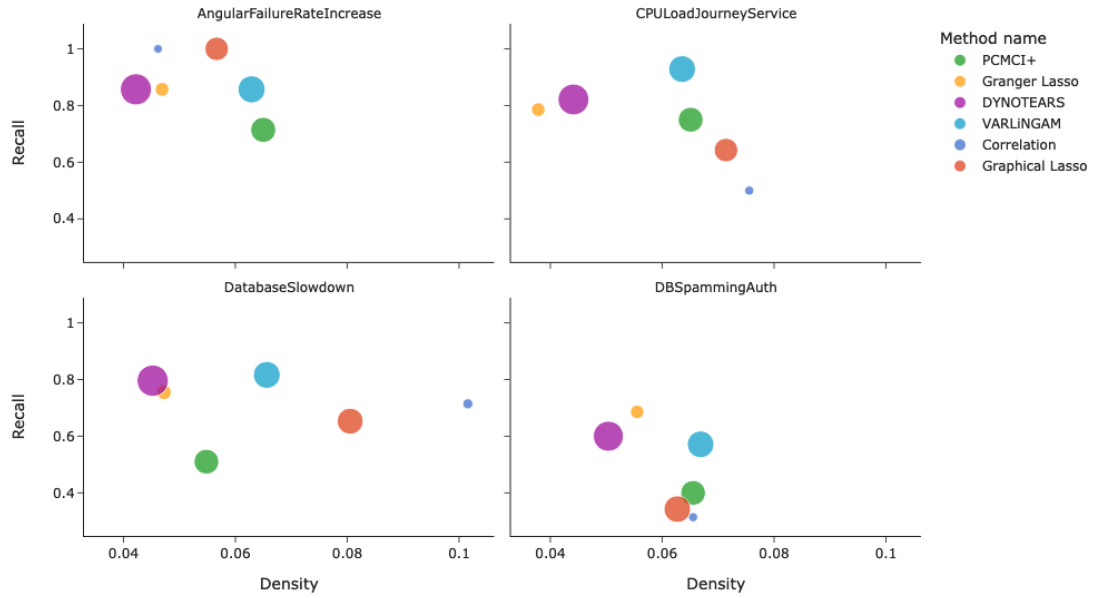


Figure 5.2: Average recall and density for each problem type – size reflecting run time

lem, both naive approaches are outperformed by all of the causal discovery algorithms with regard to recall and density. For the DatabaseSlowdown problem, all the causal discovery algorithms have a lower density compared to the naive approaches, while DYNOTEARS, Granger Lasso and VARLiNGAM also have higher recall values. For the DBSpammingAuthWithAppDeployment problem, all causal discovery algorithms have a higher recall compared to the naive approaches, but only DYNOTEARS and Granger Lasso have clearly lower density values.

While the recall criteria only considers the affected metrics and their connections to the root cause service, the rootcause rank criteria is influenced by all edges in the output graph. This works by ranking each service compared to all other services based on the fraction of outgoing edges that connect metrics of the service of interest to metrics from all the other services. Plot 5.3 shows how often the actual root cause service is placed on a specific rank from one to ten based on the output graph produced by the implemented approaches. In this case, rank one means that the metrics of the root cause service have the most outgoing edges and therefore the actual root cause is also considered to be the service most likely to have caused the failure according to the algorithm. For reasons of clarity, the ranks from five to ten are presented as one group.

One can see that the causal discovery algorithm DYNOTEARS and the naive approach using the graphical lasso are able to rank the actual root cause as the most likely root cause more often than all other approaches. In case one not only considers the first rank, but the first three ranks, DYNOTEARS would perform best, followed by the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

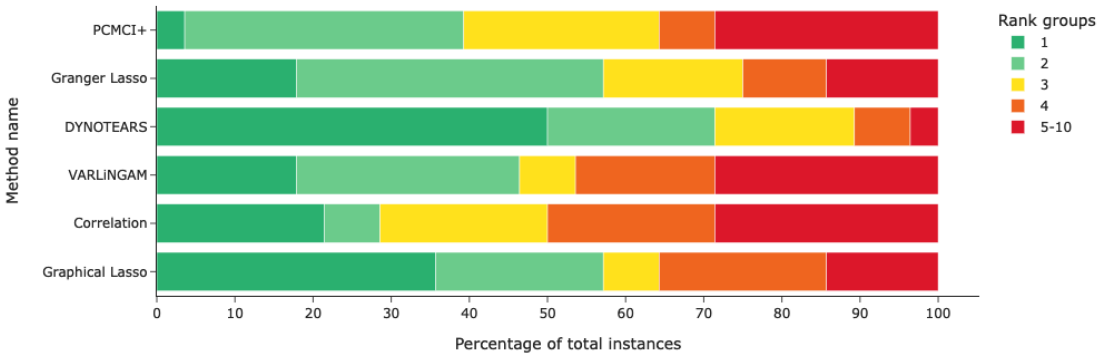


Figure 5.3: Ranking of the root cause service based on out degree centrality

Granger Lasso. The naive approach using the Pearson correlation is never among the top performing approaches regardless which ranks are considered to be relevant.

After presenting the results based on the criteria recall, density and rootcause rank, the run time results of the implemented approaches are discussed in greater detail with Table 5.1 showing the average run time for each method across all 28 problem instances. As already mentioned in Section 4.4, it is not sensible to refer to the exact values measured, but the differences in magnitude between the different approaches can still provide valuable insights.

By far the fastest method is the naive approach using the Pearson correlation, with the next faster method being the Granger Lasso, which is on average four time slower than the Pearson correlation approach. The causal discovery method PCMCI+ and the naive approach using the graphical lasso have similar run times with around three to four seconds, but are already much slower than the two fastest methods. Next is the causal discovery algorithm VARLiNGAM, which is around three to four times slower than the just mentioned methods. By far the slowest approach is the DYNOTEARS algorithm with an average run time twenty times slower than the second slowest method and several magnitudes slower than the fastest approach using the Pearson correlation.

In addition to the average run times presented in Table 5.1, Plot 5.4 shows the run time for each problem instance and implemented approach plotted against the density of the corresponding output graphs produced.

Here, one can see that the run time of each of the implemented approaches seems to be independent from the number of connections detected by the respective algorithm, i.e. the density of the output graph. In general, the variability between individual problem instance run times for the same approach is small compared to the run time differences between different methods. Only the naive approach using the graphical lasso showed deviating run times for some problem instances due to convergence issues.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

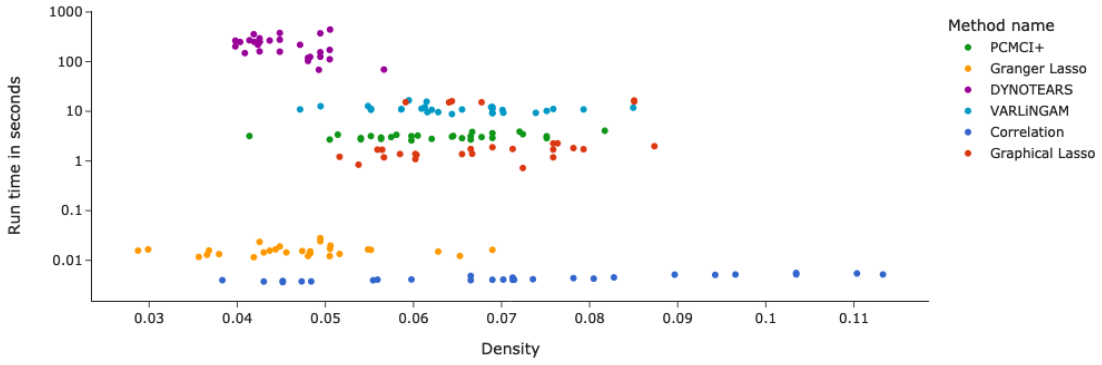


Figure 5.4: Method run time and corresponding output graph density for each problem instance

The next results to be presented concerns the number of unambiguously directed edges in the output graphs produced by the implemented approaches. For this, Plot 5.5 shows the average percentage of unambiguously directed edges across all problem instances with a high value indicating that only for a small fraction of the overall edges the corresponding algorithm could not decide on an orientation.

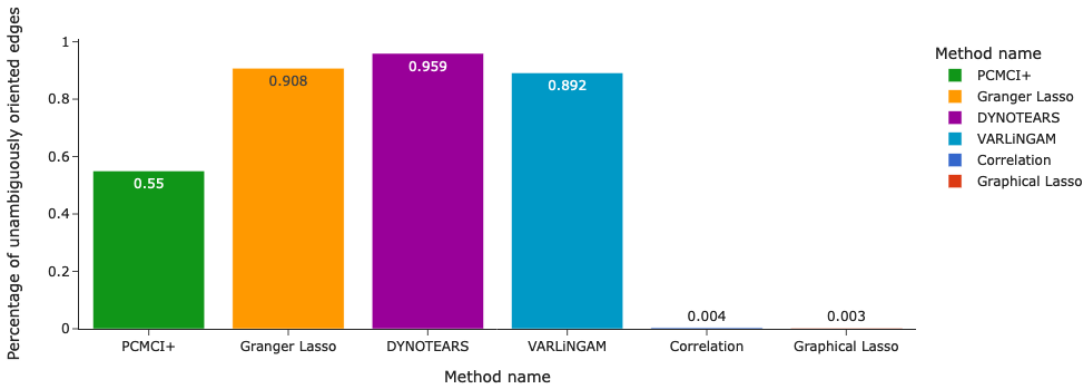


Figure 5.5: Average percentage of unambiguously directed edges

It shows that all causal discovery algorithms oriented the majority of all edges in their respective output graphs unambiguously with DYNOTEARS, Granger Lasso and VARLiNGAM even orienting almost all edges. Both naive approaches have nearly no uniquely oriented edges as the vast majority of their edges connect metrics with no time lag between them for which they are not able to decide on a distinct direction.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Nonetheless, these high percentages of oriented edges do not mean that the connections indicated in this way actually represent causal dependencies. Yet and despite the fact there is no way to assess the correctness of the orientation for all edges based on the implemented study design, it might be possible to estimate it for a small fraction of the overall edges. One way is to calculate the recall, which measures the presence of paths who are likely to be causal, for two different versions of the original output graphs. These versions are called undirected graph type and edge reversed graph type and are determined as follows: The recall for the undirected graph type refers to the percentage of predefined causal paths found in the undirected versions of the original output graphs. Likewise, the recall for the edge reversed graph type refers to the percentage of predefined causal paths found in the original output graphs where the direction of all edges was reversed.

Plot 5.6 shows the average recall values across all problem instances for different versions of the output graph. The two main aspects related to this depiction are the following: First, a small difference between the recall values of the undirected graph type and the original directed recall values indicates that most predefined causal paths are already contained in the directed output graph versions. Second, low recall values for the edge reversed graph type indicate that the predefined causal paths can not be found in an edge reversed version of the output graph.

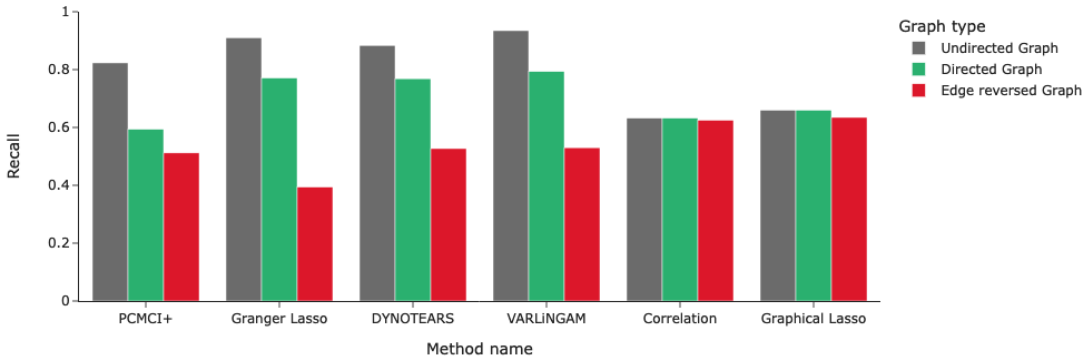


Figure 5.6: Average recall for different output graph types

One can see that all causal discovery algorithms show a clear difference between the recall for the undirected, the directed and the edge reversed version of the output graphs, while the two naive approaches show almost no difference. Furthermore, while the methods DYNOTEARS, Granger Lasso and VARLiNGAM have similar recall for the directed graph type, the recall for the edge reversed version differs with the Granger Lasso having the lowest recall in this regard. The exact values can be found in Table 5.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Method	UG Recall	DG Recall	ERG Recall
PCMCI+	0.823	0.594	0.512
Granger Lasso	0.920	0.771	0.394
DYNOTEARS	0.884	0.769	0.527
VARLiNGAM	0.935	0.793	0.530
Correlation	0.632	0.632	0.625
Graphical Lasso	0.660	0.660	0.635

Table 5.2: Average recall for different output graph types – UG denoting undirected graph, DG denoting directed graph, ERG denoting edge reversed graph

5.2 Scenario 2: Shrinking Symmetric Time Window

The results presented in this section show the performance of the implemented approaches for the second examined evaluation scenario, which should resemble a *post-mortem analysis* in which the start and the end of a detected problem is known more precisely, and thus the analysis time window can be chosen narrower.

Plot 5.7 shows the changes in average recall across all problem instances as the total time windows changes from having 30 data points before and after the problem time frame to having zero data points. This means that the total time window at the left-most position of the x-axis consists of the 90 minutes, while at the right-most position the total time windows only spans 30 minutes during which a problem pattern was active.

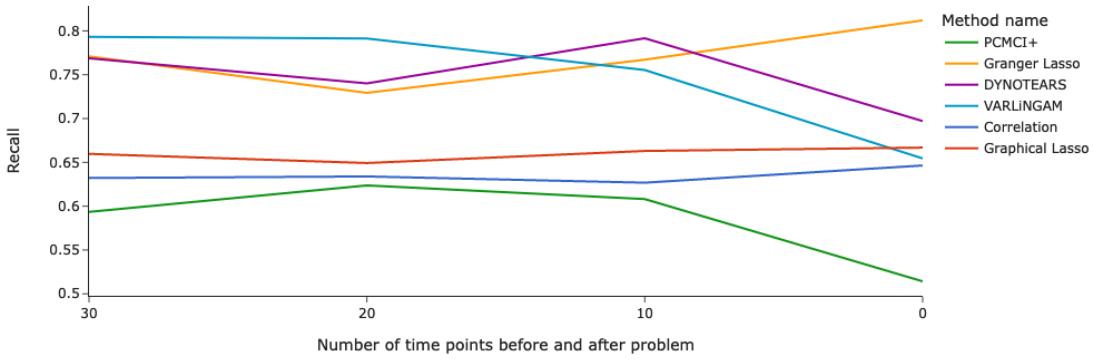


Figure 5.7: Average recall for different numbers of time points before and after the problem period

One can see that the average recall of each of the implemented approaches seems fairly steady regardless how accurately the analysis time frame was selected to cover the period during which a problem pattern was active. In fact, the three causal discovery methods PCMCI+, VARLiNGAM and DYNOTEARS even have their worst result in the case

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

where the problem time frame is selected exactly. Furthermore, apart from the method VARLiNGAM, which shows a slight downward trend, no clear pattern is visible. In general, the results regarding the recall are in line with the results seen in the first scenario with the three causal discovery methods DYNOTEARS, Granger Lasso and VARLiNGAM outperforming the two naive approaches.

The second criteria evaluated over a changing total time window is the rootcause rank. Plot 5.8 shows these changes as the time window decreases from a total of 90 minutes, which includes 30 minutes before and after the problem time period, to a total time window length of only 30 minutes during which a problem pattern was active.

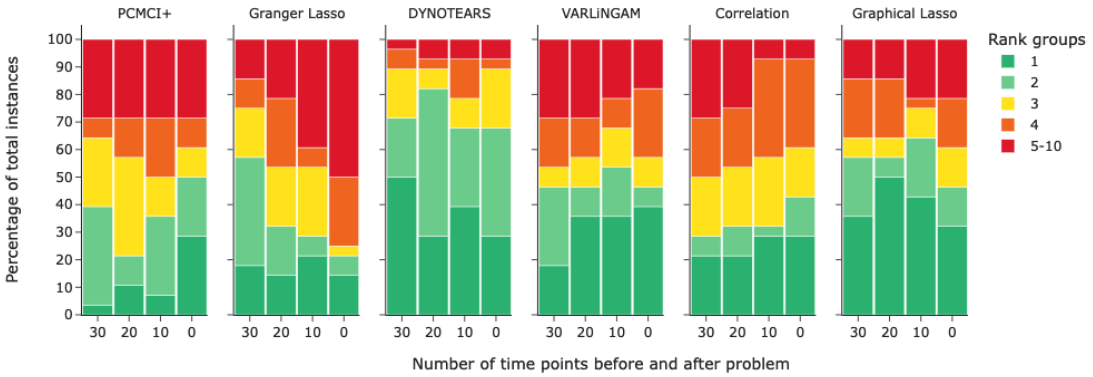


Figure 5.8: Ranking of the root cause service based on their out degree centrality for different numbers of time points before and after the problem period

Similar to the steady performance regarding the recall shown before, one can see that also the performance regarding the root cause rank criteria seems to be fairly steady for most of the implemented methods, namely the causal discovery methods PCMCI+, DYNOTEARS and VARLiNGAM and the naive approach using the graphical lasso. The exception to this is the Granger Lasso method, which shows much better ranking results for the 90 minute total time window compared to the exact 30 minute time window around the problem. The naive approach using the Pearson correlation shows a less pronounced and somewhat different trend with slightly worse results as the total time window increases. Compared to the results seen in the first scenario, the results are less clear here. Nonetheless, one can see that the methods DYNOTEARS and graphical lasso, which performed well in the first scenario also perform well here and that the naive approach using the Pearson correlation is again not among the top performing approaches.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.3 Scenario 3: Expanding Time Window

The results presented in this section show the performance of the implemented approaches for the third examined evaluation scenario, which should resemble a *real-time analysis* in which a problem just started and is slowly unfolding and thus the analysis time window is expanding.

The Plot 5.9 shows the changes in average recall across all problem instances as the total time window progressively expands to include more and more data points during which a problem pattern is active. This means that the total time window at the left-most position of the x-axis is 30 minutes long and does not contain any data points during which a problem pattern is active, while at the right-most position the total time window spans 90 minutes and contains the 30 minute problem period.

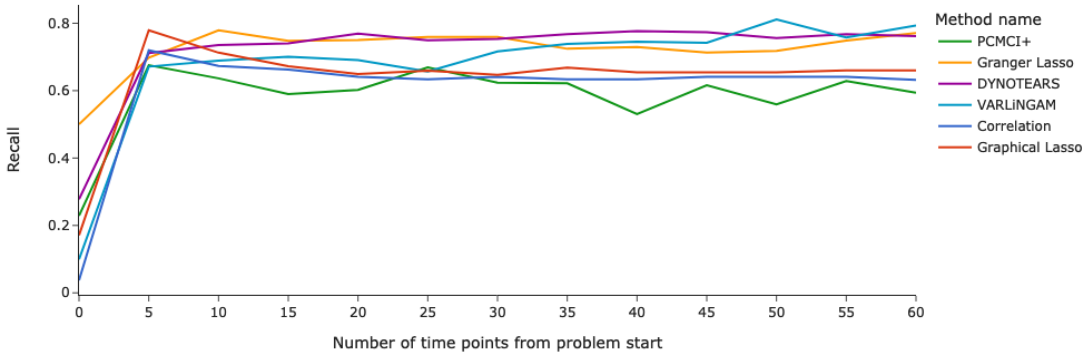


Figure 5.9: Average recall for an expanding total time window

One can see that the average recall for the left-most position is extremely low for most of the implemented approaches and dramatically improves as soon as a small part of the problem period is included in the analysis time frame. Interestingly, the Granger Lasso method was able to detect about half of the predefined causal paths on average even before the problem started, which differs substantially from the results of all the other approaches. After a small part of the problem period is covered, the performance of all methods is fairly stable and in line with the results observed in the first scenario as the three causal discovery methods DYNOTEARS, Granger Lasso and VARLiNGAM show higher average recall values than the two naive approaches for the majority of the time. Furthermore, after the initial increase, no method shows a clear trend with regard to the expanding time frame.

The results for the root cause ranking criteria, which are shown in Plot 5.10, are less clear. The causal discovery method DYNOTEARS and the naive approach using the graphical lasso show a clear improvement as soon as a small part of the problem period is included in the analysis time frame and a steady performance after that, similar to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

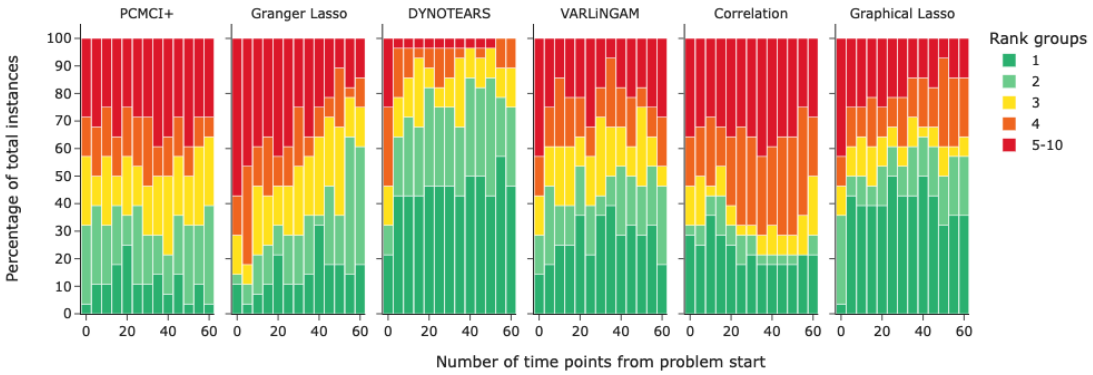


Figure 5.10: Ranking of the root cause service based on their out degree centrality for an expanding total time window

the results regarding the recall. This differs substantially from the results of the causal discovery methods PCMCI+ and VARLiNGAM and the naive approach using the Pearson correlation, which do not show such a clear difference. Similar to the second scenario, the Granger Lasso method shows a steady trend with better performance as the length of analysis time window increases.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Discussion

This chapter answers the initial research questions by discussing and interpreting the study results presented in Chapter 5. It is structured according to the three research questions.

Research Question One: *How well can causal discovery algorithms detect connections between the causes and the effects of software failures based on metric time series data alone? How successfully can these methods identify the software component that is causing the failure?*

All of the evaluated causal discovery algorithms, namely PCMCI+, Granger Lasso, DYNOTEARS and VARLiNGAM, were able to detect short, consistently oriented paths from the causes to the effect of the examined software failures based on metric time series data alone for the majority of the predefined cause-effect pairs with all methods expect for PCMCI+ even achieving an average directed recall of over 75%. In the case one is mainly interested in the detected relationships between the causes and effects and not their direction of influence, i.e. the skeleton of the generated output graph, the performance – the undirected recall – is even better. Both findings are all the more remarkable as the output graphs, which indicate metric interactions using edges, can be considered fairly sparse. Furthermore, while the recall is not as high for all four examined problem types, the results still suggest that causal discovery algorithms are capable of detecting directed connections from cause to effect for different problem types without explicit optimization of these algorithms for any of them.

For identifying the software component that is causing the failure, the evaluated causal discovery algorithms also perform quite well in the majority of cases by ranking the actual root cause service higher than most other services with regard to the likelihood that they have caused the software failure. Since this ranking is entirely based on the output graph produced by the evaluated methods, where a good ranking of a service

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

means that the metrics of that service influence many other services, this, on the one hand, confirms the assumption that the root cause service is an influential factor of application performance and, on the other hand, suggests that the evaluated algorithms can detect these influences in many cases.

A more general question with regard to the more practical research questions outlined above is if the orientation of edges between metrics in the output graphs really coincide with the actual (causal) direction of influence between them. While this can not be answered unambiguously for all the detected metric-metrics connections with the here implemented study design, at least for the edges, where the direction of influence is known, it seems that the direction of influence is often correctly determined. This can be seen by the fact that for the three best performing causal discovery approaches, namely DYNOTEARS, VARLiNGAM and Granger Lasso, the vast majority of all detected connections between cause and effect consistently point from cause to effect, suggesting that a correct orientation by accident is unlikely. Additionally, in case the original orientation of all detected connections is reversed, far fewer connections from cause to effect can be found, further strengthening the assumption that the orientation is unlikely to be random and that the original orientation coincides with the actual direction of influence. Last and as already briefly addressed with regard to the identification of the failure causing software component, in most cases the evaluated causal discovery algorithms detect many outgoing, influencing, connections from the actual root cause of the software failures, which, once more, suggests that the estimated direction of influence actually coincides with the actual (causal) direction of influence between the metrics.

Research Question Two: *How well do these algorithms perform in comparison to two simple baseline approaches – one using the Pearson correlation and one using the graphical lasso – with regard to the tasks stated in RQ1?*

The two baseline approaches are also able to detect short connections between the causes and effects of the examined software failures based on metric time series data alone, but they do perform worse than most of evaluated causal discovery algorithms in most cases. An exception to this is the AngularFailureRate problem, the simplest of the four examined software failures, where both naive approaches have a perfect detection rate of 100%. This outstanding performance can however most likely be attributed to the fact that for this problem pattern the change in error rate of the root cause service almost one-to-one corresponds to the change in error rate for the affected service, which represents the best case scenario for detecting a metric-metric connection based on high (partial) correlation coefficients. All the other problem patterns represent more complex failure scenarios and for these problem patterns, causal discovery based approaches in general have a higher detection rate. Another disadvantage of the naive approaches besides their inferior performance is their almost complete inability to decide on the direction of influence between variables, while several evaluated causal discovery algorithms were able to orient up to 90% of their detected relationships. While these high percentages of oriented edges does not mean that the estimated directions of influence are always correct, the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

arguments put forward in the discussion of research question one nonetheless suggest that the orientation seems to be correct more often than not. Although the value of this benefit can not be quantified here, it is not the less a feature of causal discovery, which the naive approaches can not provide.

For identifying the software component that is causing the failure, the performance of the two naive approaches have to be considered separately. While the Pearson correlation approach is always among the worst performing approaches overall, the graphical lasso approach performs quite well, even outperforming some causal discovery methods in some cases. One potential explanation of this poor performance of the correlation approach is that it may detect many incorrect (spurious) metric-metric interactions between services where neither service is the actual root cause service, thereby diluting the correctly detected influences of the actual root cause service. In comparison, the graphical lasso approach should not be negatively affected by this effect, as the partial correlation coefficients calculated based on the method's result represent the association between time series while taking the associations between all other time series into account. This explanation would also explain why the density of the output graphs produced by the graphical lasso approach is always sparser, i.e. contains fewer detected influences, than that of the Pearson correlation approach. Nonetheless, several of the evaluated causal discovery methods perform as well as the graphical lasso approach or even better.

Another important criteria for using any of the evaluated approaches as part of an actual root cause analysis system besides their detection performance is their run time. With regard to this aspect, the Pearson correlation approach is as expected the fastest method, but surprisingly the causal discovery method Granger Lasso, which offers considerably better performance than the naive approach, is only slightly slower in absolute terms. This means that existing root cause analysis systems that currently utilize the Pearson correlation or even slower methods for estimating associations between different metrics could benefit from this better detection performance while adding little to no overhead to the overall run time of the RCA system.
However, apart from the surprisingly low run time of the Granger Lasso approach, the overall evaluation largely confirmed the general assumption that the more complex causal discovery methods have better detection performance but worse run times.

Research Question Three: *How sensitive is the performance of the examined causal discovery algorithms and the baseline approaches with regard to the start and end points of the metric time series data?*

Based on the different evaluation scenarios carried out during this work, one can conclude that the performance of all examined approaches regarding the tasks mentioned in RQ1 can be said to be not very sensitive with respect to the start and end points of the metric time-series data. While the absolute values for different evaluation criteria vary depending on the selected time frame, which is to be expected when using data from an actual running software application, the relative performance between the different approaches changes very little. One unexpected finding was that when the time frame

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

exactly covers the actual problem time frame, the number of causal paths found – the recall – decreases for many causal discovery methods. One possible explanation could be the resulting overall shorter time window that gives the methods less data to work with, resulting in poorer performance. While this certainly can have an effect, based on the results of the third evaluation scenario, it may also be the case that this difference in performance can be partly attributed to the fact that the transition period from a normal operating state to a malfunctioning operating state contains essential information and therefore the relevant connections are harder to detect, if this transition period is not part of the input data.

In general it can be stated that the conclusions made with regard to RQ1 and RQ2 seem to be valid regardless of the exact start and end points of the time series data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion

The goal of this master thesis was to better understand how useful and applicable different causal discovery methods are for the task of root cause analysis and how they perform compared to simpler techniques with regard to multiple evaluation criteria. Therefore, within the course of this thesis, four existing causal discovery algorithms – each based on a different concept – and two simpler techniques were compared and evaluated with regard to the just stated task. For this purpose, 40 metric time series data sets were collected from a multi-tier demo web application – called *easyTravel* [48] – over periods of time where the application underwent some form of performance degradation due to four different, intentionally evoked, problems. These metric time series data sets were then used as input for the to be evaluated algorithms to identify connections between individual metrics. Last, the graph structures resulting from this modelling step were then evaluated according to four different evaluation criteria. The two most important of which were how many predefined connections between the causes and effects of the different software failures could be detected and how easily the error causing software components could be identified. In addition, the algorithms were also compared in terms of their run time.

Both, the modelling and the evaluation step were carried out under three different scenarios to make the results and conclusion drawn as generally applicable as possible. The in depth comparison showed that the detection performance of the evaluated causal discovery algorithms can differ quite substantially between the different approaches, but overall most of them outperform simpler techniques for identifying relevant connections between individual metrics like the Pearson correlation. These results suggest that many existing RCA systems who rely on a graph like structure for their root cause search could greatly benefit from using one of the evaluated causal discovery algorithms.

While the evaluation that was carried out revealed many interesting insights, there are still numerous questions that could be answered in future work. First, the software application used within this work was a multi-tier web application, which represents a

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

commonly used type of software system. Nonetheless, there exist numerous other kinds of applications with potentially completely different workloads like Internet of things (IoT) or batch processing applications. Further evaluation of one of these systems could provide valuable insights about the generality of the conclusions reached in this thesis. Second, while the evaluation of this work only examined algorithms which can detect linear relationships, future work could also consider methods for detecting non-linear connections and examine how common highly non-linear dependencies between metrics are. Last, the metric time series data used in this work had a time interval, i.e. the time between two consecutive data points, of one minute. Future work could thus experiment with different sampling frequencies of the input time series data to investigate the effect this has on the performance of the evaluated methods.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] G. Linden, "Marissa Mayer at Web 2.0," http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html, Nov. 2006, last visited 21.10.2022.

[2] A. Lerner, "The Cost of Downtime," https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/, 2014, last visited 21.10.2022.

[3] IBM Cloud Education, "What is Application Performance Management (APM)?" https://www.ibm.com/cloud/learn/application-performance-management, Jun. 2019, last visited 21.10.2022.

[4] J. Soldani and A. Brogi, "Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey," ACM Computing Surveys, vol. 55, no. 3, pp. 1–39, 2022. [Online]. Available: https://doi.org/10.1145/3501297

[5] M. Ma, W. Lin, D. Pan, and P. Wang, "MS-Rank: Multi-Metric and Self-Adaptive Root Cause Diagnosis for Microservice Applications," in 2019 IEEE International Conference on Web Services (ICWS). IEEE, 2019, pp. 60–67. [Online]. Available: https://doi.org/10.1109/ICWS.2019.00022

[6] ——, "Self-Adaptive Root Cause Diagnosis for Large-Scale Microservice Architecture," IEEE Transactions on Services Computing, vol. 15, no. 3, pp. 1399–1410, 2022. [Online]. Available: https://doi.org/10.1109/TSC.2020.2993251

[7] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "AutoMAP: Diagnose Your Microservice-based Web Applications Automatically," in Proceedings of The Web Conference 2020. Association for Computing Machinery, 2020, pp. 246–258. [Online]. Available: https://doi.org/10.1145/3366423.3380111

[8] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "CloudRanger: Root Cause Identification for Cloud Native Systems," in 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE, 2018, pp. 492–502. [Online]. Available: https://doi.org/10.1109/CCGRID.2018.00076

[9] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing Failure Root Causes in a Microservice through

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Causality Inference," in 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS). IEEE, 2020, pp. 1–10. [Online]. Available: https://doi.org/10.1109/IWQoS49365.2020.9213058

[10] L. Mariani, C. Monni, M. Pezze, O. Riganelli, and R. Xin, "Localizing Faults in Cloud Systems," in 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2018, pp. 262–273. [Online]. Available: https://doi.org/10.1109/ICST.2018.00034

[11] R. Moraffah, P. Sheth, M. Karami, A. Bhattacharya, Q. Wang, A. Tahir, A. Raglin, and H. Liu, "Causal inference for time series analysis: problems, methods and evaluation," Knowledge and Information Systems, vol. 63, no. 12, pp. 3041–3085, 2021. [Online]. Available: https://doi.org/10.1007/s10115-021-01621-0

[12] A. R. Lawrence, M. Kaiser, R. Sampaio, and M. Sipos, "Data Generating Process to Evaluate Causal Discovery Techniques for Time Series Data," 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2104.08043

[13] C. K. Assaad, E. Devijver, and E. Gaussier, "Survey and Evaluation of Causal Discovery Methods for Time Series," Journal of Artificial Intelligence Research, vol. 73, pp. 767–819, 2022. [Online]. Available: https://doi.org/10.1613/jair.1.13428

[14] R. Wirth and J. Hipp, "CRISP-DM: Towards a standard process model for data mining," in Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining, vol. 1, 2000, pp. 29–39.

[15] E. Hariton and J. J. Locascio, "Randomised controlled trials - the gold standard for effectiveness research," vol. 125, no. 13, p. 1716, 2018. [Online]. Available: https://doi.org/10.1111%2F1471-0528.15199

[16] J. Pearl and D. Mackenzie, The Book of Why: The New Science of Cause and Effect. Penguin Books, 2019.

[17] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., 1988.

[18] J. Peters, D. Janzing, and B. Schölkopf, Elements of Causal Inference: Foundations and Learning Algorithms. MIT Press, 2017.

[19] S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H.-G. Leimer, "Independence properties of directed markov fields," Networks, vol. 20, no. 5, pp. 491–505, 1990. [Online]. Available: https://doi.org/10.1002/net.3230200503

[20] D. M. Hausman and J. Woodward, "Independence, invariance and the causal markov condition," British Journal for the Philosophy of Science, vol. 50, no. 4, pp. 521–583, 1999. [Online]. Available: https://doi.org/10.1093/bjps/50.4.521

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[21] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, Causation, Prediction, and Search. MIT press, 2000.

[22] J. Runge, "Causal network reconstruction from time series: From theoretical assumptions to practical estimation," Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 28, no. 7, 2018. [Online]. Available: https://doi.org/10.1063/1.5025050

[23] M. Gong, K. Zhang, B. Schölkopf, C. Glymour, and D. Tao, "Causal discovery from temporally aggregated time series," in Uncertainty in Artificial Intelligence - Proceedings of the 33rd Conference, vol. 2017. PubMed Central, 2017.

[24] T. Verma and J. Pearl, "Equivalence and synthesis of causal models," in UAI '90: Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, MIT, Cambridge, MA, USA, July 27-29, 1990, 1991, pp. 220–227.

[25] P. Spirtes and C. Glymour, "An Algorithm for Fast Recovery of Sparse Causal Graphs," Social Science Computer Review, vol. 9, no. 1, pp. 62–72, 1991. [Online]. Available: https://doi.org/10.1177/089443939100900106

[26] J. Runge, "Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets," in Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), vol. 124. PMLR, 2020, pp. 1388–1397. [Online]. Available: https://proceedings.mlr.press/v124/runge20a.html

[27] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," Science Advances, vol. 5, no. 11, p. eaau4996, 2019. [Online]. Available: https://doi.org/10.1126/sciadv.aau4996

[28] C. W. J. Granger, "Investigating Causal Relations by Econometric Models and Cross-spectral Methods," Econometrica, vol. 37, no. 3, pp. 424–438, 1969. [Online]. Available: https://doi.org/10.2307/1912791

[29] A. Arnold, Y. Liu, and N. Abe, "Temporal causal modeling with graphical granger methods," in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Association for Computing Machinery, 2007, p. 66–75. [Online]. Available: https://doi.org/10.1145/1281192.1281203

[30] R. Tibshirani, "Regression shrinkage and selection via the lasso," Journal of the Royal Statistical Society: Series B (Methodological), vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

[31] The OEIS Foundation Inc., "Number of acyclic digraphs (or dags) with n labeled nodes," http://oeis.org/A003024, last visited 21.10.2022.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[32] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing, "DAGs with NO TEARS: Continuous optimization for structure learning," in NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems. Curran Associates Inc., 2018, p. 9492–9503.

[33] R. Pamfil, N. Sriwattanaworachai, S. Desai, P. Pilgerstorfer, P. Beaumont, K. Georgatzis, and B. Aragam, "DYNOTEARS: Structure learning from time-series data," in Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, PMLR, vol. 108. PMLR, 2020, pp. 1595–1605. [Online]. Available: http://proceedings.mlr.press/v108/pamfil20a.html

[34] S. Shimizu, P. Hoyer, A. Hyvärinen, and A. Kerminen, "A linear non-gaussian acyclic model for causal discovery," Journal of Machine Learning Research, vol. 7, pp. 2003–2030, 2006.

[35] E. Oja, A. Hyvarinen, and J. Karhunen, Independent Component Analysis. John Wiley & Sons, 2001.

[36] A. Hyvärinen, K. Zhang, S. Shimizu, and P. O. Hoyer, "Estimation of a structural vector autoregression model using non-gaussianity," The Journal of Machine Learning Research, vol. 11, p. 1709–1731, 2010.

[37] H. Krasner, "The Cost of Poor Quality Software in the US: A 2018 Report," https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report, Sep. 2018, last visited 21.10.2022.

[38] C. Heger, A. van Hoorn, M. Mann, and D. Okanović, "Application performance management: State of the art and challenges for the future," in Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering. Association for Computing Machinery, 2017, p. 429–432. [Online]. Available: http://dx.doi.org/10.1145/3030207.3053674

[39] F. De Silva, P. Byrne, and J. Chessman, "Gartner Magic Quadrant for Application Performance Monitoring," https://www.gartner.com/en/documents/4000354, Apr. 2021, last visited 21.10.2022.

[40] R. Rabiser, K. Schmid, H. Eichelberger, M. Vierhauser, S. Guinea, and P. Grünbacher, "A domain analysis of resource and requirements monitoring: Towards a comprehensive model of the software monitoring domain," Information and Software Technology, vol. 111, pp. 86–109, 2019. [Online]. Available: https://doi.org/10.1016/j.infsof.2019.03.013

[41] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," ACM Computing Surveys, vol. 48, no. 1, 2015. [Online]. Available: https://doi.org/10.1145/2791120

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[42] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," 2017. [Online]. Available: http://arxiv.org/abs/1701.08546

[43] C. Glymour, K. Zhang, and P. Spirtes, "Review of Causal Discovery Methods Based on Graphical Models," Frontiers in Genetics, vol. 10, 2019. [Online]. Available: https://doi.org/10.3389/fgene.2019.00524

[44] V. Arya, K. Shanmugam, P. Aggarwal, Q. Wang, P. Mohapatra, and S. Nagar, "Evaluation of Causal Inference Techniques for AIOps," in 8th ACM IKDD CODS and 26th COMAD. Association for Computing Machinery, 2021, pp. 188–192. [Online]. Available: https://doi.org/10.1145/3430984.3431027

[45] J. Thalheim, A. Rodrigues, I. E. Akkus, P. Bhatotia, R. Chen, B. Viswanath, L. Jiao, and C. Fetzer, "Sieve: actionable insights from monitored metrics in distributed systems," in Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. Association for Computing Machinery, 2017, p. 14–27. [Online]. Available: https://doi.org/10.1145/3135974.3135977

[46] P. Chen, Y. Qi, P. Zheng, and D. Hou, "CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in IEEE INFOCOM 2014 - IEEE Conference on Computer Communications. IEEE, 2014, pp. 1887–1895. [Online]. Available: https://doi.org/10.1109/INFOCOM.2014.6848128

[47] P. Chen, Y. Qi, and D. Hou, "*CauseInfer*: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment," IEEE Transactions on Services Computing, vol. 12, no. 2, pp. 214–230, 2019. [Online]. Available: https://doi.org/10.1109/TSC.2016.2607739

[48] L. Karolina, "easyTravel Documentation and Download," https://community.dynatrace.com/t5/Getting-started/easyTravel-Documentation-and-Download/m-p/181271, Feb. 2022, last visited 21.10.2022.

[49] Dynatrace Support Center, "Service detection and naming," https://www.dynatrace.com/support/help/how-to-use-dynatrace/services/service-detection-and-naming, last visited 21.10.2022.

[50] L. Karolina, "Available easyTravel problem patterns," https://community.dynatrace.com/t5/Getting-started/Available-easyTravel-problem-patterns/m-p/181674, Mar. 2022, last visited 21.10.2022.

[51] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," Biostatistics, vol. 9, no. 3, pp. 432–441, 2007. [Online]. Available: https://doi.org/10.1093/biostatistics/kxm045

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[52] R. Feldt and A. Magazinius, "Validity Threats in Empirical Software Engineering Research - An Initial Survey," in Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE), 2010, p. 374–379.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.