INAUGURAL-DISSERTATION

zur
Erlangung der Doktorwürde

der
Naturwissenschaftlich-Mathematischen
Gesamtfakultät

der
RUPRECHT-KARLS-UNIVERSITÄT

HEIDELBERG

vorgelegt von

Dipl-Ing. (FH), M.Sc.: Martin Grössl

aus: Immenstadt / Allgäu

Tag der mündlichen Prüfung: .....

Konzeptioneller Ansatz einer Fehlerprädiktionsumgebung

Betreuer: Professor Dr. Dr. h.c. Andreas Reuter

Zusammenfassung

Die im Rahmen der vorliegenden Arbeit entwickelte Fehlerprädiktionsumgebung basiert auf ausgewählten Methoden der Statistik, des maschinellen Lernens und des Data-Mining. Zur Repräsentation des Systemverhaltens eines beobachteten Informationssystems wurden mathematische Modelle, wie Bayes'sche Netze und Markov-Ketten erstellt, die mittels eines probabilistischen Model-Checking sowohl den Entwicklungsverlauf (Pfad) in einen Fehlerzustand als auch die Zeitdauer bis zum Auftreten dieses möglichen Fehlers analysieren.
Des Weiteren wurde auf der Grundlage eines Kalman-Filters ein Ansatz zur Identifikation von Anomalien bzw. Fehlverhalten in Datenströmen entwickelt. Dieser beinhaltet eine System-Identifikation, welche die Modelle, die die Basis für die Erkennung bilden, aus Messdaten ableitet.

Abstract

In this thesis a failure prediction environment based on selected methods of statistics, machine learning and data mining was developed. For the system behavior representation of an observed information system mathematical models, such as Bayesian networks and Markov chains, were generated. These models were analyzed using a probabilistic model-checking for both the process evolution (path) into a fault condition as well as the time duration until the occurrence of this possible fault.
Furthermore, based on a Kalman filter an approach for identification of anomalies and / or misbehavior in data streams was developed. This also includes a system identification part, which derives the models from measurement data. These models form the basis for the misbehavior detection.

Danksagung

An dieser Stelle möchte ich mich herzlich bei Herrn Professor Dr. Dr. h.c. Andreas Reuter für die Übernahme der Betreuung der Doktorarbeit bedanken. Seine zahlreichen Ratschläge, Hinweise und Diskussionen haben den Verlauf der Arbeit geprägt. Mein Dank gilt auch dem Heidelberger Institut für Theoretische Studien (HITS gGmbh) für die finanzielle Unterstützung und die Bereitstellung der Infrastruktur in der Zeit meiner Promotion sowie der Fakultät für Mathematik und Informatik der Universität Heidelberg für die administrative Begleitung. Schließlich danke ich meiner Familie für die Unterstützung in den arbeitsintensiven Phasen während meiner Promotion.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Es gibt nur wenige Bereiche, in denen keine Daten gesammelt und ausgewertet werden. Computer und ihre entsprechend notwendige, oft ganz spezielle Software übernehmen aktuell diese Aufgaben. Bei vielen Anwendungen, besonders in sicherheitskritischen, aber auch unternehmensrelevanten Bereichen, besteht ein großer Bedarf, den Betriebszustand des Systems zu beobachten. Je differenzierter ein solches System ist, desto komplexer stellen sich die sogenannten Überwachungsmaßnahmen dar. Diese sind mitunter nicht direkt handhabbar, können aber auf verschiedene Arten durchgeführt werden.
In diesem Zusammenhang werden häufig für die Bewertung des Systemverhaltens Messwerte von charakteristischen Systemgrößen (z.B. Sensoren) herangezogen. Das Ergebnis gibt allerdings nur näherungsweise den tatsächlichen Zustand an. Präzisere Aussagen werden üblicherweise mittels mathematischer Modelle, z.B. Transitionssysteme und andere, die das Systemverhalten abbilden, getroffen. Modelle, die das gesamte Verhalten von komplexen IT-Systemen abdecken, sind selten bzw. meist nicht vorhanden. Die Spezifikation solcher Modelle ist speziell bei hoch komplexen Systemen, wie z.B. Flug- bzw. Bahn-Reservierungssystemen, die mitunter aus 10^7 Komponenten bestehen, aufwändig bis gar unmöglich. Jedoch bei genau dieser Kategorie komplexer Systeme hat der Betreiber ein besonderes Interesse, einen fehlerfreien und kontinuierlichen Betrieb zu gewährleisten. Ein besonderes Augenmerk gilt daher der Verhinderung bzw. Vermeidung von Systemausfällen. Häufig ist deshalb aus Sicht der Betreiber die Prädiktion eines möglichen Fehlverhaltens erwünscht. Weil aber die Beobachtung von Betriebsdaten üblicherweise nur eine Aussage über das Verhalten einer Komponente, z.B. Hardware, Software (Betriebssystem, Anwendungen, Dienste), zulässt und selten den kompletten Zustand des unterliegenden Systems repräsentiert, wird für Prädiktionsanwendungen ein modellbasierter Ansatz angestrebt. Eine Möglichkeit, ein solches komplettes Systemmodell zu generieren, sind datengetriebene Ansätze, d.h. aus Betriebsdaten wird mit

Hilfe statistischer Methoden ein entsprechendes Modell abgeleitet. Mit Modellen dieser Kategorie lassen sich unterschiedliche Systemklassen abdecken. Da für die Generierung der Modelle Systemdaten verwendet werden, die sowohl normale Betriebsdaten als auch bekannte Fehlfunktionen und ggf. Anomalien in den Daten enthalten, ist die Prädiktion von bekannten Fehlern mit Hilfe solcher Modelle realisierbar. Eine solche Modellierung begünstigt auch die Detektion von Systemfehlern, da beispielsweise Anomalien in den Messdaten in Form von längeren Antwortzeiten bei Systemanfragen abgebildet werden können. Eine solche Systemabbildung lässt nicht nur Schlüsse über den Systemzustand zu, sondern ermöglicht es auch, Systemausfälle zu verhindern. Hierdurch besteht die Möglichkeit, frühzeitig entsprechende Gegenmaßnahmen zur Systemstabilisierung bzw. zum Umgehen kritischer Zustände zu initiieren. Ein in der Praxis gebräuchlicher Ansatz ist die zustandsbehaftete Modellierung solcher Systeme, bei der neben den normalen Betriebszuständen auch bekannte Fehlerzustände bzw. unerwünschte Betriebszustände im Modell explizit abgebildet werden. Dies gewährleistet eine sehr hohe Abdeckung des Systemverhaltens.

1.2 Problemstellung

In industriellen Prozessen und Informationssystemen muss ein langfristiger unterbrechungsfreier Betrieb gewährleistet werden. Voraussetzungen dafür sind Kenntnisse sowohl des Systemverhaltens als auch in Bezug auf mögliche Ereignisse, die in naher Zukunft auftreten könnten. Bei den Ereignissen liegt das Augenmerk auf Fehlern oder einem Fehlverhalten des Systems. Diese Problematik beeinflusst das Systemverhalten insofern, als es zu Unregelmäßigkeiten oder gar zum Stillstand kommen kann.
In der Praxis versucht man, mit Prädiktionsansätzen diese Probleme zu vermeiden. Meist liegt jedoch keine komplette Spezifikation oder gar ein Modell des Systems vor. Es bietet sich daher an, solche Modelle aus Meßdaten des Systems zu generieren. Für die datengetriebene Generierung der Modelle und die modellbasierte Prädiktion kommen meist statistische oder signalverarbeitende Ansätze zum Einsatz.

1.3 Zielsetzung und Beitrag der Arbeit

Im Rahmen der vorliegenden Arbeit wurde ein Ansatz entwickelt, der je nach den Gegebenheiten automatisiert oder automatisiert mit Vorgaben Verhaltensmodelle von Informationssystemen erlernt. Im Rahmen eines Lernverfahrens wurde sowohl das zustandsbasierte als auch das dynamische Verhalten (Zeit) ermittelt und in einem Modell abgebildet. Eine bekannte Einschränkung von zustandsbasierten Modellen ist, dass diese auf diskrete Daten festgelegt sind. In technischen Systemen sind jedoch häufig Messdaten mit kontinuierlicher Charakteristik vorhanden. Deshalb wurde speziell für das in der vorliegenden Arbeit behandelte Lernverfahren eine angepasste Diskretisierung der kontinuierlichen

Variablen entwickelt. Diese Option ermöglicht neben der Diskretisierung auch eine Reduktion der Daten, so dass sowohl die Durchführung des Lernprozesses als auch die weitere Verarbeitung der erlernten Modelle mit moderaten Hardwareressourcen erfolgen kann.
Des Weiteren werden in vielen technischen Systemen Fehlerereignisse mittels Textnachrichten in einem sog. Log abgespeichert. Mittels Hilfsvariablen wurde nun eine Möglichkeit geschaffen, die vorgenannten Daten in zustandsbasierte Modelle zu integrieren.
Darüber hinaus wurde eine Modellierung des Systemverhaltens eines Informationssystems aus Messdaten durchgeführt unter Einsatz der zuvor genannten Erweiterungen. Solche Modelle können sowohl zur Laufzeit (online) angewandt werden als auch als Grundlage für statische Analysen (offline) dienen. Bei der Anwendung zur Laufzeit sind Aussagen über das Systemverhalten bzw. dessen Klassifikation, z.B. im Normalbetrieb, bei Fehlern oder kritischen Zuständen möglich. Bei einer Offline-Anwendung lassen sich hingegen Analysen durchführen, welche die Ursache eines Fehlers, die Zeit bis zum Auftreten eines Fehlers, aber auch die Wahrscheinlichkeit eines Fehlers offenlegen. Mit statischen Analysen beschäftigt sich u.a. die vorliegende Arbeit.

1.4 Aufbau der Arbeit

Bei industriellen Prozessen und Informationssystemen wird häufig eine Aussage über das Systemverhalten der sich im Betrieb befindlichen Computersysteme benötigt. In der Praxis liegt jedoch meist keine komplette Modellspezifikation des Systems vor. Ein solches Modell wird aber unabdingbar für das Verfolgen bzw. die Vorhersage des Systemverhaltens benötigt. In der vorliegenden Arbeit wird der Ansatz einer Fehlerprädiktionsumgebung behandelt, welche auf statistischen Ansätzen beruht. Die theoretischen Grundlagen, die die Basis hierzu bilden, sind in Kapitel 4 (Formale Grundlagen) aufgeführt. Dabei handelt es sich um Ansätze von Bayes'schen Netzen mit Zeiteigenschaften, nichtlineare Kalman-Filter, Support-Vektor-Maschinen, probabilistisches Model-Checking, datenbasierte Lernverfahren zur Modellbildung und einer unterraumbasierten System-Identifikation.
Die erarbeiteten Ansätze werden in Kapitel 5 (Realisierungsansätze) beschrieben. Es handelt sich dabei um einen Fehler-Instrumentierungsansatz, eine nichtlineare Systemidentifikation und eine statistische Anomalie-Erkennung auf der Basis eines nichtlinearen Kalman-Filters. Des Weiteren wurde eine Systemmodellierung, die sowohl diskrete als auch diskretisierte kontinuierliche Variablen behandelt, durchgeführt. Bekannte Systemfehler wurden explizit in den Datenbestand integriert und somit in das Modell aufgenommen, so dass entsprechende Analysen möglich sind. Die erlernten Modelle lassen Aussagen über das Zeitverhalten der normalen Betriebszustände wie auch über das Auftreten und die Verweildauer des Fehlverhaltens bzw. der Fehler zu. Die vorgenannten Modelle wurden statisch analysiert. Dadurch kann die Entwicklung des Systems in einen Fehlerzustand aufgedeckt werden. Darüber hinaus sind auch Annahmen über

die Wahrscheinlichkeit und die Zeitdauer bis zum Entstehen eines Fehlers ableitbar.
Die in Kapitel 5 vorgestellten Methoden wurden in Kapitel 6 (Evaluierung) in praktischen Beispielen angewandt und die erzielten Ergebnisse präsentiert. Die Basis der Auswertung bildeten Daten eines Server-Clusters, so dass realistische Ergebnisse erreicht wurden.
Die „Zusammenfassung und der Ausblick“ der Arbeit werden in Kapitel 7 dargestellt.

Kapitel 2

Stand der Technik

Computersysteme werden in den unterschiedlichsten Bereichen eingesetzt, die u.a. sowohl sicherheitskritische als auch geschäftskritische Applikationen umfassen. Ihr Betrieb kann somit einen wichtigen Einfluss auf Bereiche des menschlichen Lebens, der Ökonomie eines Unternehmens, der Umwelt, aber auch der eigentlichen, auszuführenden Aufgabe (Auftrag / Mission) einer Anlage (Industrieanlage, Flugzeug, Rakete) ausüben. Für die Gewährleistung eines geordneten Betriebsablaufs solcher Systeme werden Fehlererkennungsmethoden und Fehlerprädiktionsansätze angewandt. Eine Möglichkeit, ereignisbasierte Systemausfälle vorherzusagen, wird in [118] vorgestellt. Dabei wird eine spezielle Erweiterung der zeitbasierten Hidden-Markov-Modelle verwendet, die sogenannten Hidden-Semi-Markov-Modelle (HSMM). Die Konstruktion der HSM-Modelle erfolgt auf der Basis von Fehlersequenzen, die mit maschinellen Lerntechniken (Clustering) aus aufgezeichneten Systemereignissen extrahiert werden. Der Hauptfokus der HSMM-Anwendung liegt auf der Fehlervorhersage zur Systemlaufzeit, die mit realen Systemdaten evaluiert wird. Eine andere Möglichkeit ist, Fehlerprädiktion mit statistischen Ansätzen [50] zu behandeln. Hierbei kommen Methoden des maschinellen Lernens, die radialen Basisfunktions-Kernel (RBF-Kernel), zur Modellierung der Systemdynamiken zum Einsatz. Des Weiteren wird mittels identischer statistischer Techniken eine Variablenselektion durchgeführt.
Entsprechende Prädiktionsmethoden werden nachfolgend beispielhaft aufgeführt: Im Bereich der medizinischen Diagnose bzw. der Medizintechnik [39] kommen sowohl zur Entwicklung von realistischen Diagnose-Szenarien als auch um den Grund für kardiogene Herzinsuffizienz (akuter Herzinfarkt) zu diagnostizieren und die voraussichtliche Entwicklung zu antizipieren, Bayes'sche Netze mit kontinuierlicher Zeit zum Einsatz.
Ein weiteres, sehr beachtliches Anwendungsfeld der Fehlererkennung und Prognose liegt in der Luft- und Raumfahrt. Von der National Aeronautics and Space Administration (NASA) wurden im Bereich des Software Health-Management entsprechende Werkzeuge und Ansätze [121] entwickelt und evaluiert. Diese Ansätze wurden mittels einer Software (ISWHM) in verschiedenen Fallstudien auf

Daten des Betriebssystems, der Anwendungssoftware und auf Messwerte der Hardware-Sensoren angewandt. Damit sollten in einer integrierten Art und Weise Fehler in Hard- und Software- Komponenten auch während des Betriebs detektiert werden. Die ISWHM-Software nutzt für die interne Modellierung ebenfalls Bayes'sche Netze. In diesem Kontext entstand auch eine weitere komplette Software [53] [54] zur Systemüberwachung, die Rückschlüsse auf den „Gesundheitszustand" des beobachteten Systems zulässt. Diese Software wurde speziell dazu entwickelt, komplexe und in Echtzeit schwer überwachbare Systeme abzudecken. Hierbei kamen in einer Trainingsphase Data-Mining-Techniken, wie z.B. Clustering, zum Einsatz, um typisches Systemverhalten zu charakterisieren. In der Betriebsphase werden die vorab gebildeten Gruppen mit weiteren Monitoring-Daten verglichen und mit Abstandsmetriken die Distanzen ermittelt. Neue Daten, die eine große Distanzabweichung zu den Gruppen aufweisen, werden als Fehlfunktion identifiziert. Auf der Basis dieser Methoden wurde u.a. auch ein integriertes Fahrzeugmanagementprojekt [139] [140] durchgeführt. In diesem Rahmen fand eine Anwendung von Data-Mining-Software auf Daten verschiedener Flugzeuge, Technologieträger und Demonstratoren statt.
Des Weiteren hat die NASA in einem Projekt [137] Werkzeuge und Technologien entwickelt, die die automatische Erkennung, Diagnose und Prädiktion von Software-Anomalien ermöglichen. Diese adressieren überwiegend komplexe Softwaresysteme, bei denen Fehler während des Betriebs u.a. durch Interaktion mit der Umgebung auftreten können. Diese Fehler können durch unvorhergesehene Umgebungsveränderungen hervorgerufen werden, die nicht während des Entwicklungsprozesses berücksichtigt wurden bzw. werden konnten. In diesem Rahmen wird auch ein konkreter Forschungsvorschlag [122] für den Einsatz Bayes'scher Netze im Bereich des integrierten Software-Gesundheits-Management (Integrated Software Health Management) aufgezeigt. Dieser richtet sich speziell an sicherheits-kritische, -relevante, hochverfügbare hochsichere Softwaresysteme, welche während des Betriebes kontinuierlich überwacht werden und schnellstmöglich Anomalien erkennen, so dass automatisiert eine Fehler-Ursachen-Analyse und Migrationsstrategien durchgeführt werden können. Die Thematik der Systemzustandsmodellierung mit Bayes'schen Netzen wird auch in [25] aufgegriffen. In einer Fallstudie [120] werden die Fähigkeiten eines Sensor- und Systemüberwachungsmanagements bei einem kleinen Raumfahrzeug präsentiert. Dabei findet die entsprechende Auswertung während des Betriebs eines kleinen Satelliten statt.
Im vorangegangenen Absatz wurde die Anwendung der Bayes'schen Netze für unterschiedliche Szenarien beschrieben. Der Bayes'sche Netzansatz wurde in [12] für den Bereich der Zuverlässigkeits-Modellierung und -Analyse aufgegriffen. In diesem Artikel wurden gebräuchliche Analysetechniken, wie Fehlerbäume und Markovketten mit korrespondierenden Teilen der Bayes'schen Netzstruktur dargestellt und verglichen.
Eine Erweiterung der Bayes'schen Netze mit kontinuierlicher Zeit wurde [156] [155] zur Netzwerk-Einbruchserkennung (Intrusion Detection) angewandt. In diesen Arbeiten wurden mittels maschinellen Lernens sämtliche Anomalien und suspekte Muster identifiziert und aus dem Datenbestand eliminiert, so dass aus-

schließlich das Normalverhalten, welches keine Angriffsmuster enthält, vorhanden blieb. Auf dieser Datenbasis wurde ein Normalverhaltensmodell gelernt, das online gegen reale Netzwerksverkehrsdaten verglichen wurde. Dadurch wird eine Signalisierung von Netzwerkangriffen möglich. Ein ähnlicher Ansatz fand bei der Analyse sozialer Netzwerke [33] [32] Anwendung. Hier wurde die Dynamik der Netze mit einem speziellen Lernverfahren ermittelt, das die Beziehungen gewichtet, erfasst und als Modell abbildet.
Dagegen werden industrielle Anwendungen, wie z.B. eine Prozessüberwachung, [85] mit hybriden Bayes'schen Netzen modelliert. Dies dient der Systemzustandsverfolgung von komplexen physikalischen Systemen, die sowohl diskrete als auch kontinuierliche Größen haben. Damit sollen Rückschlüsse bei der Diagnose ermöglicht werden, wenn Komponenten ausfallen.
Aber auch im Anwendungsbereich der Robotik [98] [31] wird ein hybrides System, nämlich die funktionale Darstellung eines K9 Marsrovers, mittels Kalman-Filter bzw. Rao-Blackwellized Partikel-Filter und Bayes'scher Netze mit kontinuierlicher Zeit (CTBN) repräsentiert. Dies soll eine Erweiterung der per definitionem diskreten CTBN um kontinuierliche Variablen zum Zweck der Systemdiagnose darstellen.
Auch Fehler, die während des Betriebs einer Serverfarm auftreten, lassen sich [49] mit CTBN prognostizieren. In diesem Beitrag werden, auf der Basis von aufgezeichneten Daten eines großen Netzwerks, die Relationen zwischen den Fehlern gelernt und ermöglichen somit eine strukturelle Fehleranalyse und Abschätzung des Betriebszeitverhaltens.
Neben den bisher genannten praktischen Anwendungsfällen wurden im Umfeld der Fehlertoleranz theoretische Arbeiten durchgeführt. Formale Grundlagen für die Fehlermodellierung in verteilten reaktiven Systemen werden in [13] behandelt. Es finden formale Methoden Anwendung, die üblicherweise ein fehlerfreies System repräsentieren. Es sollte im Rahmen der Systementwicklung jedoch möglich sein, Fehler als unerwünschte, aber dennoch unvermeidbare Teile eines Systems explizit aufzunehmen. Daher wird z.B. die formale Methodik mit dem Namen „Focus“ um Begriffe und Vorgehensweisen, die einen Umgang mit verschiedenartigsten Fehlern erlauben, erweitert. Die Erweiterung von Fokus umfasst Begriffe und Notationen, mit denen verschiedenste Klassen von Fehlern eines verteilten reaktiven Systems sowohl in abstrakter, eigenschaftsorientierter Black-Box-Sicht als auch im Kontext von Transitionssystemen dargestellt werden können. Diese Methodik unterstützt eine Systementwicklung mit hohen Qualitätsanforderungen, da potentielle Fehler somit auf systematische Weise modellierbar, analysierbar und behandelbar werden. Hierdurch sollen präzise, nachweisbare Aussagen über die Fehlertoleranz eines Systems und die Auswirkungen von Fehlern gemacht werden.
In [38] werden die Grundlagen der Fehlertoleranz verteilter Systeme präsentiert, die wichtigsten Systemmodelle für fehlertolerante verteilte Systeme vorgestellt und miteinander verglichen. Darüber hinaus wird die grundsätzliche Funktionsweise von Fehlertoleranzverfahren untersucht und um eine Redundanzkomponente (Speicherredundanz bzw. Zeitredundanz) erweitert. Es wird gezeigt, welche Art von Redundanz notwendig ist, um im Falle eines Fehlers bestimmte

Eigenschaften zu erfüllen.
Ein weiteres Anwendungsfeld der Fehlererkennung und -vorhersage umfasst den Bereich der Energieversorgung. Auf diesem Gebiet wird [89] [133] mit Hilfe von modellbasierten Prädiktionsansätzen versucht, Ausfälle in Stromnetzen vorauszusagen. Dies erfordert die Bildung entsprechender dynamischer Modelle, die mit Hilfe spezieller Algorithmen Schutzoperationen bei Netzstörungen durchführen sollen.
Dagegen kommen bei industriellen Prozessen und Windanlagen vermehrt datengetriebene Techniken zum Einsatz: Für die Zustandsüberwachung und Performanzprognose von Windenergie-Systemen [72] werden numerische Techniken, wie Singulärwertzerlegung, Hauptkomponentenanalyse und Hankel-Matrizen zur Zerlegung einer Messwert-Zeitreihe in Teilsegmente verwendet. Ein weiterer Ansatz nutzt die Fisher-Diskriminanten-Analyse zur Bestimmung von Fehlern [73].
Häufig sind nicht alle Fehler bekannt bzw. können nur schwer identifiziert werden. Daher befasst sich ein Forschungsbereich mit datengetriebenen Methoden zur Fehlerklassifikation. Diese sogenannten Data-Mining-Techniken zur Fehlerentdeckung wurden in unterschiedlichen Fallstudien für Prozessanlagen [115] [138] untersucht. Darüber hinaus wurde von der NASA auch eine Software (ORCA) zur Anomalie-Erkennung entwickelt, die mit Datenbeständen [1] u.a. von Mondraketen und vom Ares-System evaluiert wurde.
Prädiktion und Prognose finden auch im Batterie- und Akkumulatoren-Bereich Anwendung. Der Lebens- und Ladezustand von Batterien kann mit Hilfe von Diagnosemethoden ermittelt werden. Dies geschieht [117] durch den Einsatz von Kalman-Filtern. Eine vergleichbare Technik findet auch bei der Fehlerdiagnose von Pico-Satelliten [2] Anwendung.
Ein statistischer Fehlerdetektor für das Lastverhalten eines Computer-Systems auf der Basis eines Kalman-Filters wurde in [68] vorgestellt. Dabei wird mittels Statistik die Ausgabe des erweiterten Kalman-Filters (EKF) gegen den Messwert ausgewertet und Abschätzungen über Fehlverhalten getroffen.
In vielen praktischen Anwendungen kommen sowohl diskrete als auch kontinuierliche Variablen vor. Üblicherweise werden für die Verarbeitung die kontinuierlichen Variablen diskretisiert, allerdings ist das nur eine Annäherung an das tatsächliche Verhalten. In [27] werden daher zwei Ansätze für die Darstellung von kontinuierlichen Variablen betrachtet; es handelt sich dabei um bedingte Gauß'sche Verteilungen und Mischungen von trunkierten Exponentialfunktionen.
Eine recht ähnliche Methode wird für die Zuverlässigkeitsmodellierung hybrider Systeme [97] eingesetzt. Dabei wurden Bayes'sche Netze für die Anwendung auf diskrete und kontinuierliche Variablen erweitert, so dass diese hybride Eigenschaften von Systemen abdecken. Für die Behandlung der kontinuierlichen Variablen kommt u.a. eine dynamische Diskretisierung, die eine Mischung aus trunkierten Exponential-Funktionen nutzt, zum Einsatz. Dieser Ansatz ermöglicht die Modellierung der Zeitverteilung bis zu einem Fehler und das Durchführen von Zuverlässigkeitsanalysen in komplexen Systemen.
Viele Anwendungen weisen hybrides Verhalten auf, d.h. beinhalten sowohl diskrete als auch kontinuierliche Variablen. In [91] wurde für die Einschätzung der

Zuverlässigkeit von Systemen ein hybrider Bayes'scher Netz-Ansatz vorgestellt. Dieser genügt nicht nur der Forderung nach hoher Effizienz, sondern erzielt auch präzise Ergebnisse. Dabei kommt eine dynamische Diskretisierung zum Abschätzen der Wahrscheinlichkeitsdichtefunktionen zur Anwendung.
Mitunter ist eine Modellierung des Systemverhaltens nicht ausreichend, speziell dann, wenn Aussagen über Eigenschaften des zu Grunde liegenden Systems gewünscht sind. Eine etablierte Technik, die im Rahmen statischer Analysen eine quantitative Bewertung des Verhaltens zulässt, ist Model-Checking, das auch im Bereich der Zuverlässigkeitsanalyse Anwendung findet. In [48] werden unter diesem Gesichtspunkt Markov-Modelle mit Hilfe der kontinuierlichen stochastischen Logik (CSL) analysiert. Die CSL bietet die Möglichkeit, zustands- wie auch pfadbasierte Zuverlässigkeitseigenschaften zu spezifizieren. Diese werden im Rahmen der Evaluierung auf ein System, das als Markov-Kette mit kontinuierlichen Zeiteigenschaften (CTMC) dargestellt ist, angewandt. Die Auswertung erfolgt auf der Basis von Daten eines Workstation-Clusters.
In [47] wird mit Hilfe der Continuous Stochastic Reward Logic (CSRL) das Modell geprüft. Dies geschieht im Rahmen einer Fallstudie im Bereich des Adhoc-Mobile Computing. Dabei wird die Durchführbarkeit von Aktionen unter dem Gesichtspunkt der Leistungsbeschränkung gemessen. Dies erlaubt eine Beurteilung der Geschwindigkeit, Zuverlässigkeit und Verfügbarkeit.
Ein softwarebasiertes Steuerungssystem wird in [75] hinsichtlich Zuverlässigkeitseigenschaften analysiert. Da das System ein stochastisches Verhalten aufweist, kommt ein wahrscheinlichkeitsbasierter Model-Checker, die Software PRISM, zum Einsatz.
In einer weiteren Studie [78] wird die formale Verifikationstechnik für die Analyse von Computer- und Kommunikationssystemen angewandt. Dabei wird ein weiter Bereich von quantitativen Eigenschaften, wie Performance und Verfügbarkeit, abgedeckt.
Im Bereich der Satellitentechnik findet das wahrscheinlichkeitsbasierte Model-Checking [107] Anwendung. Dabei werden anhand eines formalen Satelliten-Modells die Eigenschaften der Zuverlässigkeit, Verfügbarkeit und Wartbarkeit geprüft. Ein solches Vorgehen soll die Anzahl der Fehler minimieren oder die Zeit zwischen den Fehlern (MTBF) erhöhen.
Für die Validierung der Zuverlässigkeits- und Verfügbarkeitseigenschaften der zuvor genannten Systeme kommen häufig Testansätze oder gar eine gezielte Fehlstimulation bzw. Fehlerinjektion zum Einsatz. Je nach Fehlermodell wird die Fehlererzeugung entweder in Hardware oder Software umgesetzt. Sofern nur die Softwarekomponenten im Interesse liegen, ist die in Software implementierte Fehlerinjektion (SWIFI) sehr gebräuchlich. Eine flexible softwarebasierte Fehler- und Störungs-Injektionslösung, die diese Techniken beinhaltet, ist das Softwarewerkzeug FERRARI [62].
Eine weitere Software mit dem Namen FTAPE (Fault Tolerance And Performance Evaluator) [143] kombiniert einen Fehlerinjektor, einen Systemlastgenerator und Aktivitätsmessungen. Diese Kombination ermöglicht die Injektion von Fehlern unter hohen Stressbedingungen, wie z.B. hoher Systemlast. Dabei werden die Fehler in Abhängigkeit der Systemlast eingebracht, um eine möglichst hohe

Fehlerausbreitung zu erreichen.
Ein weiterer Vertreter ist die Software Xception [15] [16]. Dieser Ansatz adressiert die erweiterten Debugger-Eigenschaften, die bei vielen modernen Prozessoren vorhanden sind, und setzt damit eine Fehlerinjektions- und Überwachungsumgebung um. Die Aktivierung der Fehler geschieht dabei unabhängig vom Prozessor-Modus. Des Weiteren können damit spezifische Speicherbereiche beeinflusst werden, da die Auslösung an bestimmte Ereignisse gekoppelt werden kann und nicht nur zeitgesteuert bzw. an Zeitüberschreitungen gebunden ist.

Kapitel 3

Terminologie und grundlegende Konzepte

In diesem Kapitel werden die grundlegenden Terminologien und Konzepte, die im direkten Bezug zur vorliegenden Arbeit stehen, vorgestellt. Diese umfassen die Definitionen von Dependability, die daraus abgeleitete Definition von Fehlern, die Erläuterung der Fehlerarten und die Auswirkungen von Fehlern. Des Weiteren werden die Entwicklungsrichtungen der Dependability, die u.a. die Fehlerprädiktion und die Fehleranalyse beinhalten, aufgezeigt.

3.1 Dependability

Computer-Systeme werden durch fünf fundamentale Eigenschaften charakterisiert: Funktionalität (functionality), Verlässlichkeit (dependability), Nutzbarkeit (usability), Geschwindigkeit (performance) und Kosten (cost). Die Eigenschaft der verlässlichen Systeme, die Dependability, ist seit Jahren ein im Bereich der Informatik etablierter Begriff. Dennoch existieren dafür verschiedene Definitionen. Eine weitverbreitete Definition wurde von der International Federation for Information Processing Working Group 10.4 erarbeitet und lautet: "The trustworthiness of a computing system which allows reliance to be justifiably placed on the services it delivers". Dies besagt, dass die Zuverlässigkeit eines Rechnersystems es berechtigterweise erlaubt, Vertrauen den von ihm gelieferten Diensten entgegenzubringen. Die Dependability ist ein Konzept, welches Eigenschaften vereint, wie sie in der nachfolgenden Aufzählung aufgelistet und mit kurzen Erläuterungen versehen sind. Dabei können Verfügbarkeit (availability), Verlässlichkeit (reliability), Absicherung (safety) und Sicherheit (security) als Haupteigenschaften angesehen werden.

- Availability - z.B. die Bereitschaft, einen Dienst korrekt auszuführen
- Reliability - z.B. einen Dienst richtig und dauerhaft anzubieten

- Safety - z.B. das Fernhalten von folgenschweren Konsequenzen für Nutzer und Umwelt
- Integrity - z.B. das Ausbleiben unsachgemäßer Systemveränderungen
- Maintainability - z.B. die Möglichkeit von Modifikationen und Reparaturen

Im weiteren Verlauf der vorliegenden Arbeit wird der Begriff der Dependability nach der zuvor aufgeführten Definition [1] verwendet. Diese Auslegung wird auch durch andere Arbeiten [4] gestützt, die ursprünglich Dependability als die Fähigkeit interpretieren, einen Dienst bereitzustellen, dem berechtigterweise vertraut werden kann. Dabei stellt der Dienst, den ein System bereitstellt, dessen Verhalten dar. Dies kann subjektiv durch Nutzer oder andere Systeme, die durch definierte Schnittstellen mit dem System interagieren, festgestellt werden. Üblicherweise wird die Funktionalität eines Systems durch formale oder Anforderungs-Spezifikationen dargestellt. Demzufolge kann ein Fehler als ein Ereignis aufgefasst werden, bei welchem die Funktionalität des Systems (Dienst) von der Spezifikation abweicht. Ein Fehler stellt die schwerste, aber überwachbare Gefahr für ein System dar. Die Begrifflichkeiten der Dependability, wie sie auch in Abbildung 3.1 aufgezeigt werden, werden im nachfolgenden Kapitel detailliert erläutert. Als besonders wichtig wird dabei die Definition des Fehlerbegriffs, in der Abbildung als Threads dargestellt, und seinen Ausprägungen angesehen.

3.2 Definitionen von Fehlern

Der Fehlerbegriff wird nach der Definition von [3] verwendet. Sehr ähnliche Definitionen können in [93], [84] und [4] gefunden werden. Für eine bessere Verständlichkeit und leichtere Zuordnung werden die englischen Fehler-Begriffe (Failure, Bug, Defect, Error) im Rahmen der Erläuterungen verwendet. Ein Fehler (Bug/Defect) im Software-Code, der zu einer Fehlfunktion (Failure) führt, wird als Versagen (Fault) bezeichnet. Sofern ein Versagen (Fault) in der Software zur Ausführungszeit auftritt, kann es zu einem Fehlerzustand (Error) im System kommen. Damit würde dann auch das System von der Spezifikation abweichen, d.h. eine Fehlfunktion ruft ein feststellbares Fehlverhalten hervor. Dabei kann die Feststellung durch einen Nutzer, entweder eine Person oder eine andere Computer-Komponente, erfolgen, was impliziert, dass unterschiedliche Vorfälle auch innerhalb eines Systems auftreten können. Solange diese Vorfälle jedoch keine Störung hervorrufen, handelt es sich nicht um einen Fehler. Daher werden auch Störungen (error) [130] als Feststellung von Fehlern bezeichnet.
Nachstehend soll zunächst ein Überblick über die unterschiedlichen Fehlertypen gegeben werden.

[1] Der wesentliche Teile dieses Kapitels wurden aus [4] entnommen, darunter fallen auch die Abbildungen.

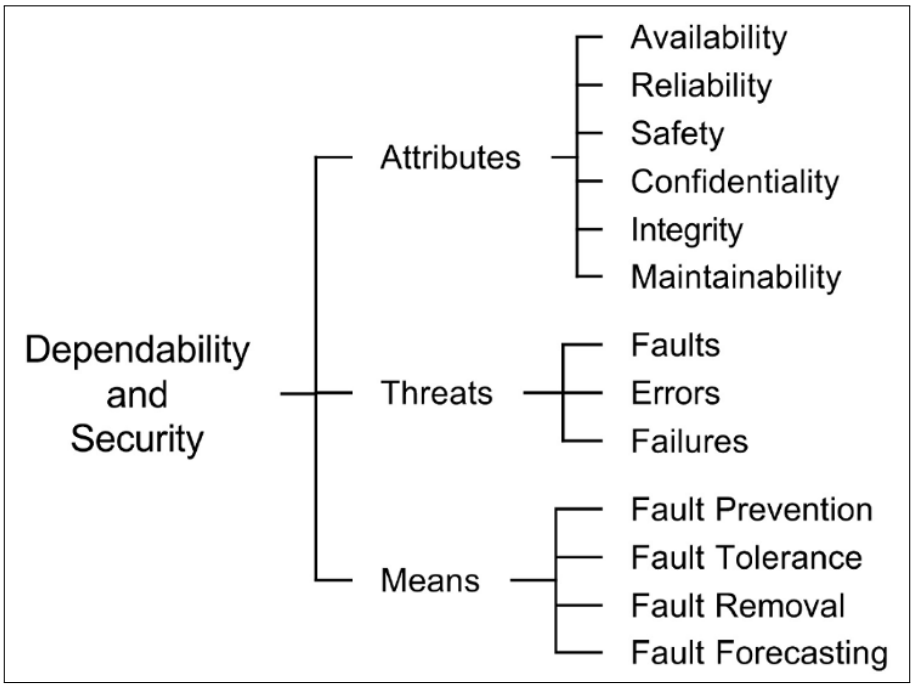


Abbildung 3.1: Dependability Überblick

- Ein System kann auf verschiedene Weise, insbesondere durch eine ständige, aktive Suche nach Fehlern überwacht werden, z.B. durch Testen von Datenstruktur-Prüfsummen u.a.
- Systemgrößen (Parameter), wie beispielsweise die Speichernutzung, die Anzahl von Prozessen, die Auslastung usw., können überwacht werden, um Nebenwirkungen (Symptome) von Fehlern zu identifizieren: z.B. weist eine Abnahme des freien Speichers über die Zeit in der Regel auf ein Speicherleck hin.
- Ein anstehendes Fehlverhalten (fault), das nicht rechtzeitig erkannt wird, kann in einen Fehler (error) übergehen.
- Wenn das Fehlverhalten (fault) nicht detektiert wird (z.B. durch einen entsprechenden Mechanismus), kann es sich in einen Fehler (error) verwandeln, der evtl. von außerhalb des Systems oder der Komponente wahrgenommen werden kann.

3.2.1 Fehlerarten

Es gibt verschiedene Fehlertypen, die aus unterschiedlichen Gründen entstehen können. Um die Definition des Fehlerbegriffs zu vervollständigen, werden

im Folgenden die einzelnen Fehlerarten näher erläutert. In der nachfolgenden Aufzählung sind die gebräuchlichsten Fehlertypen aufgeführt. Eine Darstellung, die aus den Fehlerarten auf die in Kapitel 3.3 beschriebenen Fehlermodelle schließen lässt, ist in Abbildung 3.2 gegeben.

- Permanente Fehler: Dabei handelt es sich um Defekte, die solange anstehen, bis die Ursache, z.B. durch eine Reparatur, behoben wird.
- Intermittierende Fehler: Dies sind temporäre Defekte, die durch systeminterne Mängel hervorgerufen werden.
- Transiente Fehler: Das sind temporäre Fehler, die durch Sonderfälle in den Umgebungsbedingungen hervorgerufen werden.

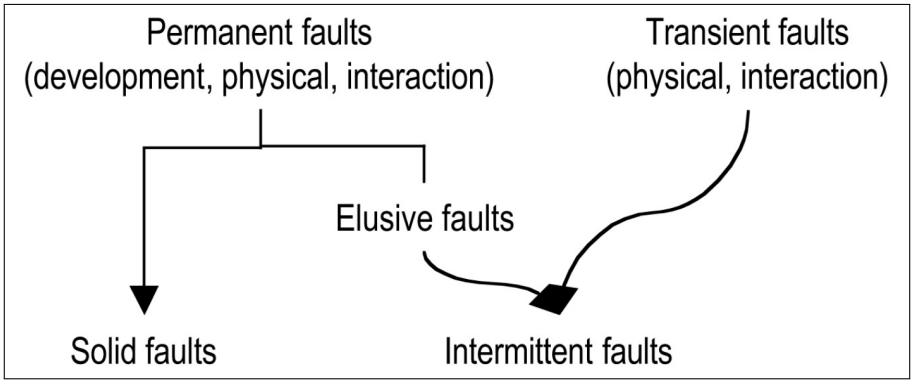


Abbildung 3.2: Zusammenhang der Fehlerarten

3.2.2 Versagen (Failure)

Eine beobachtbare Abweichung des Systemverhaltens, z.B. eine Abweichung von der Spezifikation, wird als Versagen (engl. failure) bezeichnet. Das Vorhandensein eines Versagens wirkt sich verständlicherweise auf die korrekte Funktion eines Systems aus. Prinzipiell kann die Art des Versagens berechnungs- (computation failure) bzw. zeitbezogen (timing failure) sein. Beim zeitbezogenen Versagen werden Zeitbedingungen verletzt, so werden z.B. Ergebnisse zu früh oder zu spät bereitgestellt. Hingegen werden beim wertbezogenen Versagen die Leistungen zwar zum richtigen Zeitpunkt bereitgestellt, jedoch wird ein falscher Wert als Resultat geliefert. Des Weiteren wird der völlige Ausfall eines Systems mit Totalversagen bezeichnet. Das System liefert dabei keinerlei Ergebnisse.
Das Versagen kann nach vier Gesichtspunkten, wie in Grafik 3.3 dargestellt, untergliedert werden.
Daraus leiten sich auch die gebräuchlichen Fehlerklassen, die in der nachfolgenden Aufzählung aufgeführt sind, ab.

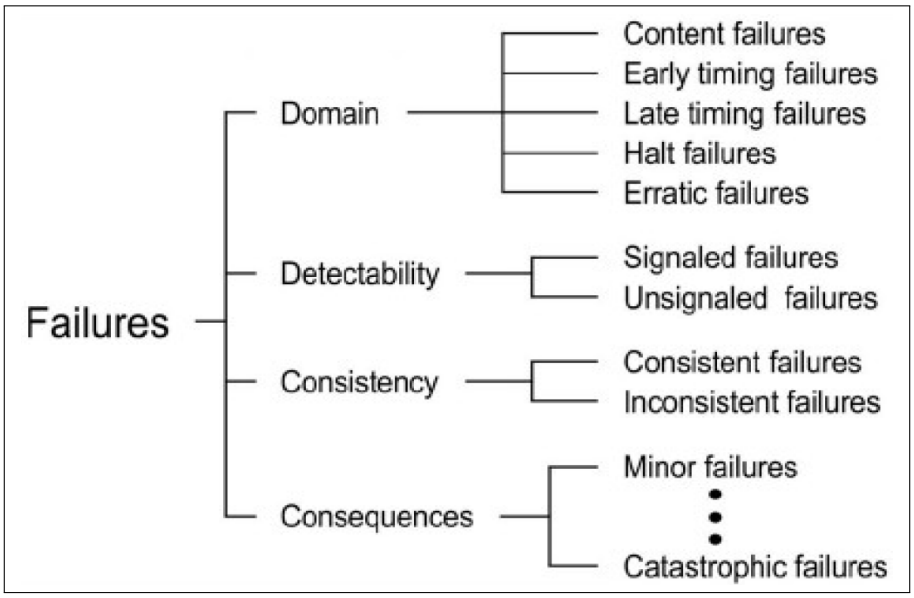


Abbildung 3.3: Fehler-Sichtweise

- Crash Failure: Der Dienst stoppt den Betrieb und setzt diesen erst nach einer entsprechenden Reparatur fort.
- Omission Failure: Der Dienst reagiert nicht bei Anfragen
- Performance Failure: Der Dienst reagiert zu spät (vorausgesetzt ein Schwellwert existiert)
- Timing Failure: Der Dienst reagiert entweder zu früh oder zu spät (vorausgesetzt es existieren zwei Schwellwerte)
- Computation Failure: Die Reaktion des Dienstes liefert ein falsches Ergebnis.
- Arbitrary Failure: Die Reaktion des Dienstes zeigt einen beliebigen Fehler.
- Authenticated Byzantine Fault: Ein willkürlicher oder tückischer Fehler, der keine authentifizierten Nachrichten beschädigen kann. (Der Sender bzw. Empfänger kann die Korruption entdecken.)
- Byzantine Fault: Jeder beliebige Fehler bzw. jede tückische Aktion ist möglich.

3.2.3 Fehlerzustand (Error)

Ein Fehlerzustand (error) wird angenommen, wenn der Zustand eines Systems vom spezifizierten bzw. erwünschten Systemzustand abweicht. Diese Annahme

lässt sich jedoch nur bei den in der Informatik gebräuchlichen zustandsbasierten Systemen definieren. Allerdings muss ein Fehlerzustand nicht zwangsweise zu einem Systemversagen führen. Beispielsweise kann ein falscher Wert einer Variablen durch einen Kontrollmechanismus abgefangen oder durch einen korrekten Wert überschrieben werden, bevor es zu negativen Auswirkungen kommt. In diesem Fall hätte ein temporärer Fehlzustand vorgelegen, der nicht zum Versagen führt.

3.2.4 Fehlerursache (Fault)

Die Einflüsse, auf Grund derer ein System in einen Fehlerzustand übergeht, werden als Fehlerursache (fault) bezeichnet. Sobald eine Fehlerursache (fault) in Erscheinung tritt, ruft sie einen Fehlerzustand (error) hervor. Ein solcher Defekt ist überall im System (alle Bestandteile / Komponenten und Ebenen) möglich. Es gibt keine Beschränkung auf Hardware, Software oder die Umgebung. Eine Übersicht der unterschiedlichen Fehlerarten und deren Entstehung wird in Abbildung 3.4 aufgezeigt.

3.3 Fehlermodelle

Die unterschiedlichen Fehler-Modelle geben Aufschluss über die Entstehung eines Fehlers und zeigen auf, wann bzw. wie dieser eingeführt wurde. Für die Entwicklung bzw. Verbesserung von Fehlertoleranz-Mechanismen muss ein entsprechendes Fehlermodell vorausgesetzt werden, das alle möglichen Arten von Fehlern in einem System spezifiziert. Prinzipiell gibt es für eine Fehlerursache nur zwei Möglichkeiten: Entweder entstand der Fehler durch menschliches Versagen oder durch die technische / physikalische Umgebung. Fehler, die durch menschliche Einwirkung entstehen, können zu unterschiedlichen Zeitpunkten in das System gelangen. Dies kann sowohl zur Entwicklungszeit als auch zur Laufzeit des Systems geschehen (z.B. durch Fehlbedienung). Andererseits können Fehler auch durch schädliche Umwelteinflüsse, wie beispielsweise: elektromagnetische oder radioaktive Strahlungen, Hitze, Kälte oder Feuchtigkeitseinflüsse entstehen.
Dies impliziert, dass die Reproduzierbarkeit von Fehlern Teil des jeweiligen Fehlermodells ist. [4] definiert eine Terminologie, bei der in Abhängigkeit der Reproduzierbarkeit zwischen soliden (harten), d.h. wenn die Aktivierung nachvollzogen werden kann, und flüchtigen (weichen) Fehlern, d.h. wenn keine systematische Reproduzierbarkeit besteht, unterschieden wird. Eine weiterreichende gebräuchliche Darstellung von Fehlermodellen [43] ist in der nachfolgenden Aufzählung aufgeführt:

- Bohrbugs - Dabei handelt es sich um deterministische Fehler, die meist recht einfach reproduziert werden können. Es sind häufig permanente Designfehler, die durch Testen aufgedeckt und im Rahmen des Entwicklungsprozesses beseitigt werden.

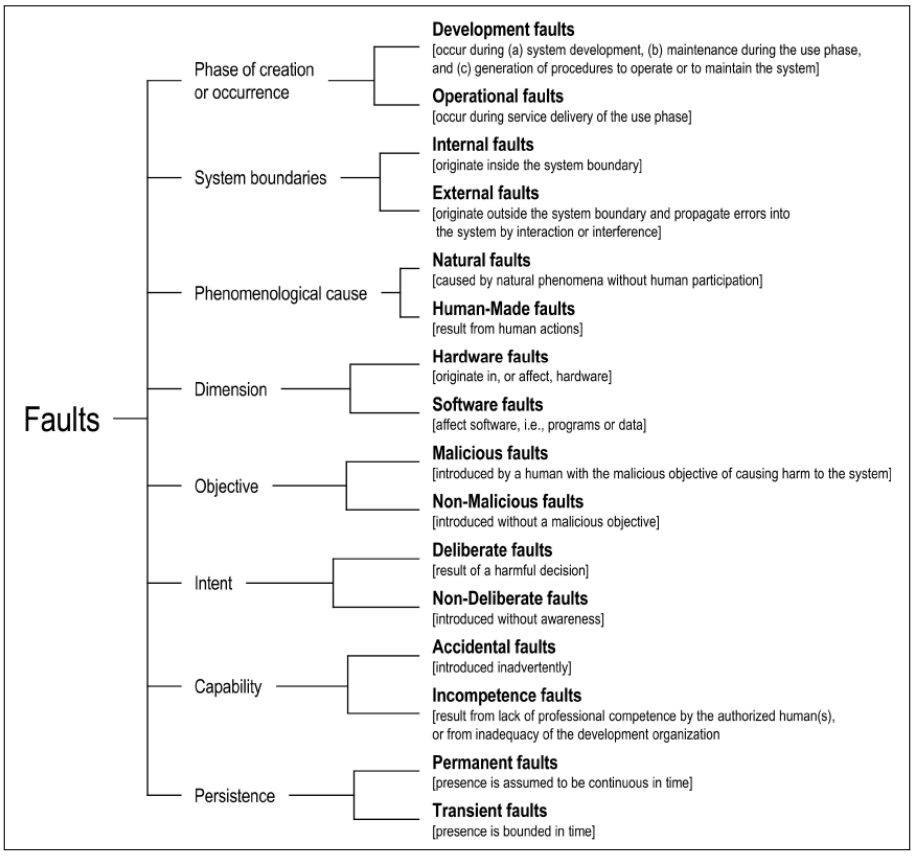


Abbildung 3.4: Fehlerarten

- Mandelbugs - Fehler dieser Kategorie treten eher chaotisch auf und sind auf vielfältige und komplexe Abhängigkeiten zurückzuführen, so dass das Auftreten nicht-deterministisch wirkt.
- Heisenbugs - Diese Fehler treten im Rahmen von Analysen bzw. gezielten Überprüfungen auf. Häufige Vertreter sind Race-Conditions, die auf Grund eines geänderten Zeitverhaltens, z.B. durch einen Debugger, vorkommen können. Solche Fehler resultieren in transientem, nicht-deterministischem Fehlverhalten. Daher sind Heisenbugs schwer zu identifizieren.
- Schrödingbugs - Diese Defekte treten eigentlich nicht in Erscheinung. Es handelt sich meist um Implementierungen, die einen Logikfehler aufweisen und nur auffallen, wenn der Quellcode analysiert wird. Häufig kommen solche Fehler bei Sicherheitsanwendungen vor, bei denen nach Bekanntwerden der Schwachstellen diese ausgenutzt werden.

3.4 Entwicklungsrichtungen der Dependability

Auf der Basis der Zusammenhänge zwischen Versagen (Failure), Fehlerzustand (Error), Fehlerursache (Fault) und den Fehlermodellen lassen sich Kategorien für die Entwicklungen im Bereich der Dependability aufzeigen. Diese sollen dazu beitragen, Maßnahmen zu entwickeln, um die Dependability bzw. deren Eigenschaften zu erreichen. Die nachfolgende Aufzählung umfaßt vier Kategorien:

- Fehler-Vermeidung: Das Auftreten oder Einführen von Fehlern soll durch vorbeugende Maßnahmen verhindert werden. Dabei ist es notwendig, Fehler zu entdecken und entsprechend darauf zu reagieren.
- Fehler-Toleranz: Hier werden Mechanismen zum Entdecken und Behandeln (Tolerieren) von Fehlern genutzt, die den Betrieb eines Systems auch bei Vorhandensein eines Fehlers gewährleisten sollen.
- Fehler-Beseitigung: Dadurch soll die Anzahl an Fehlern und deren Einfluss reduziert werden. Abbildung 3.5 [2] stellt gebräuchliche Ansätze für die Fehlerbeseitigung vor.
- Fehler-Vorhersage: Ansätze dieser Kategorie schätzen sowohl das Aufkommen von Fehlern als auch die Entwicklung zukünftiger Ereignisse und die möglichen Konsequenzen, die durch einen Fehler entstehen, ab.

Mit den genannten Mechanismen wird es möglich, ein fehlertolerantes System zu entwickeln, das trotz enthaltener und auftretender Fehler ein akzeptables Verhalten zeigt.

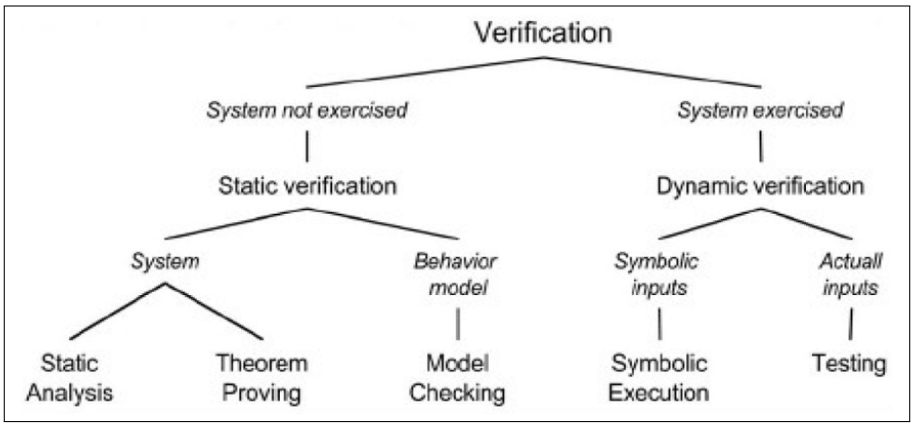


Abbildung 3.5: Fehlerüberprüfung

[2] Entnommen aus [4].

Kapitel 4

Formale Grundlagen

Für den Entwurf einer Fehlerprädiktionsumgebung war die Verwendung diverser statistischer bzw. mathematischer Methoden erforderlich. In diesem Kapitel wird eine Einführung der im Rahmen der vorliegenden Arbeit eingesetzten Methoden gegeben. Die Methoden für die Systemmodellierung, auf deren Basis die Prädiktion stattfindet, umfassen Bayes'sche Netze, Markov Ketten für kontinuierliche Zeit und Model-Checking-Ansätze. Die Fehler-Prädiktion zur Systemlaufzeit wurde auf der Basis von Kalman-Filter, System-Identifikation, Support-Vektor-Maschinen und Kernel-Methoden realisiert. Diese soll zum Verständnis der nachfolgenden Kapitel beitragen.

4.1 Bayes'sche Netze

Eine häufig verwendete Methode für die Modellierung von statischen Systemen sind Bayes'sche Netze (BN) [105]. Ein Bayes'sches Netz ist ein graphisches Modell zur Darstellung wahrscheinlichkeitstheoretischer Zusammenhänge über eine Menge von Variablen. BN sind in wissensbasierten Systemen, z.B. Experten-Systemen [108], bei denen Unsicherheiten eine wesentliche Rolle spielen, etabliert. BN werden u.a. in einem weiten Bereich von Diagnose-Systemen und der Betrugserkennung (Fraud Detection) eingesetzt.
Weiterhin ermöglichen es BN-Modelle, kausale Relationen zwischen Variablen zu modellieren. Dies kann durch eingebrachtes Expertenwissen beeinflusst werden.
Bayes'sche Netze sind ein mathematisches Konstrukt, das die vereinigte Wahrscheinlichkeitsverteilung P über eine Menge von Variablen kompakt repräsentiert. BN bestehen aus einer qualitativen und einer quantitativen Komponente. Während die qualitative Komponente die Netzwerkstruktur ausdrückt, wird die quantitative Komponente durch die Zuweisung der bedingten Wahrscheinlichkeitsverteilungen repräsentiert. Der qualitative Teil, der die Struktur bildet, ist ein azyklisch gerichteter Graph (DAG), dessen Knoten die Zufallsvariablen und dessen Kanten die Abhängigkeiten der Variablen darstellen. Ein Beispiel einer sol-

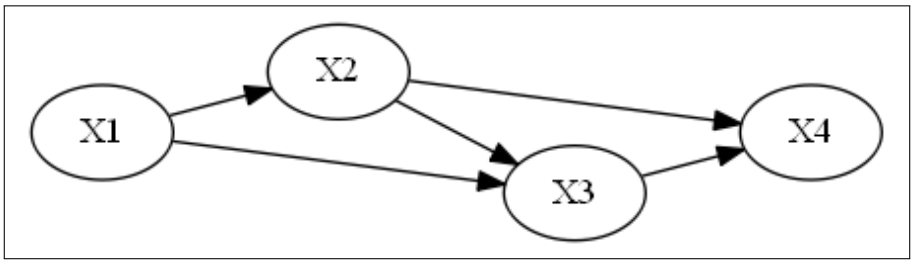


Abbildung 4.1: Beispiel einer Bayes'schen Netzwerkstruktur

chen Struktur (siehe Abbildung 4.1) wird von (bedingten) Abhängigkeiten zwischen den Variablen bestimmt.
Dem Graphen sind an den Kanten Wahrscheinlichkeiten annotiert, deren Kombination eine Verteilung definiert. Diese bedingte Wahrscheinlichkeitsverteilung (CPD) wird durch die dem Knoten zugeordnete Zufallsvariable vorgegeben. Je nach BN-Typ können die Knoten entweder nur diskrete oder sowohl diskrete als auch kontinuierliche Zufallsvariablen abbilden. Im Fall von diskreten Variablen werden die CPDs durch Wahrscheinlichkeitstabellen beschrieben, während bei kontinuierlichen Variablen üblicherweise Wahrscheinlichkeitsdichtefunktionen zum Einsatz kommen, aus denen auch die Abhängigkeiten der Zufallsvariablen, z.B. zu den Elternknoten, erkennbar sind.
Als Eltern eines Knotens V werden diejenigen Knoten bezeichnet, von denen eine Kante zu V führt. Mit Hilfe eines Bayes'schen Netzes lässt sich die gemeinsame Wahrscheinlichkeitsverteilung aller beteiligten Variablen möglichst kompakt repräsentieren. Hierbei werden bekannte, bedingte Abhängigkeiten, die durch eine Kante im Graphen dargestellt werden, ausgenutzt. Die Konstruktion des DAG und die Berechnung der zugehörigen Wahrscheinlichkeiten kann manuell durch Spezifikationen bzw. Domänenwissen oder automatisiert mittels Lernverfahren erfolgen. Sofern im DAG eine Kante vom Knoten $X1$ zum Knoten $X2$ zeigt, wird $X1$ als ein Elternteil von $X3$ und $X3$ als ein Kind von $X1$ bezeichnet. Ein Elternteil beeinflusst direkt seine Kinder. Die Eltern eines generischen Knotens X werden mit Pa(X) angegeben. Weiterhin hat jeder Knoten seine eigene lokale Wahrscheinlichkeitsverteilung. Die Vereinigung all dieser lokalen Komponenten ergibt die vereinigte Wahrscheinlichkeitsverteilung (JPD) des BN, die sogenannte quantitative Komponente. In der Praxis werden BN häufig bei großen Systemen mit einer Vielzahl an Variablen angewandt. Für die Anwendung der BN ist sowohl die Struktur (DAG) als auch die Wahrscheinlichkeitsverteilung erforderlich. Beides kann vorab spezifiziert oder von aufgezeichneten Daten abgeleitet (gelernt) werden.
Die Kombination einer Vielzahl von Variablen kann zu sehr großen Zustandsbereichen führen. Dies macht eine Repräsentation von großen Systemen und die Berechnung der Wahrscheinlichkeitsverteilung schwierig. Eine mögliche Lösung ist die Faktorisierung des Zustandsraumes. Dabei werden nur direkte Abhängigkeiten von Variablen, die durch eine Kante im Graphen ausgedrückt werden,

berücksichtigt. Dies ermöglicht, basierend auf der jeweiligen Variablen, die Bildung von Subprozessen, die nur die Variablen, die direkte Abhängigkeiten mit der ausgewählten aufweisen, enthalten. Hierbei handelt es sich um eine Erweiterung der BN zur effizienten Darstellung einer großen Anzahl interagierender Variablen welche auch eine dynamische Komponente zur Modellierung von Zeitinformationen [30] beinhaltet. Mitunter ist das Strukturmodell aber auch nicht bekannt bzw. nicht spezifiziert. Daher muss die Ermittlung der Struktur häufig mittels Lernverfahren durchgeführt werden. Das Lernen von Bayes'schen Netzen ist ein Problem der Kategorie NP-Hard [22] [21]. Generell gibt es zwei etablierte Ansätze zum Strukturlernen von Daten:

- Zum einen den „Search-and-Score"-Ansatz [20] [28]. Hier wird versucht, ein Netzwerk zu identifizieren, das eine Auswertefunktion maximiert, die anzeigt, wie gut das Netzwerk zu den Daten passt.
- Der zweite Ansatz basiert auf Bedingungen (constraints), die z.B. Relationen zwischen den Variablen ausdrücken. Algorithmen dieser Kategorie [136] [106] schätzen anhand der Daten ab, ob bestimmte bedingte Unabhängigkeiten zwischen den Variablen zutreffen. Dies wird typischerweise durch statistische oder informationstheoretische Messungen erreicht.

Beide Methoden sind mit Vor- bzw. Nachteilen behaftet und liefern daher je nach Rahmenbedingungen unterschiedliche Ergebnisse. Eine Kombination der Verfahren kann diese Einschränkungen bzw. Nachteile teilweise kompensieren. Die quantitative Komponente des Bayes'schen Netzes, die Parameter, kann ebenfalls durch Lernstrategien ermittelt werden. Dabei wird die Wahrscheinlichkeitsdichteverteilung abgeschätzt und somit die Parameter bestimmt. Es existieren dafür etablierte Ansätze, die sowohl auf vollständige als auch auf partielle Daten angewandt werden können. Die gebräuchlichsten Ansätze sind Maximum-Likelihood (ML) Estimation, Bayesian Estimation, Discriminative Parameter Learning und Generative Parameter Learning für vollständige Daten. Bei partieller Datenlage kommt der Expectation Maximization Ansatz zum Einsatz.

4.2 Verfahren zum Lernen der Struktur eines BN-Graphen Max-Min Hill-Climbing (MMHC)

Ein spezieller Algorithmus unter dem Namen Max-Min Hill-Climbing (MMHC)-Algorithmus [144] basiert auf einer Kombination der in Kapitel 4.1 erwähnten Strukturlernverfahren. Bei diesem Lernverfahren handelt es sich um einen hybriden Ansatz, bei dem Ideen des lokalen Lernens, Constraint-basierte Ansätze und Search-and-Score-Techniken in einer effektiven Weise vereint werden. In praktischen Experimenten [144] lieferte MMHC deutlich bessere Ergebnisse als State-of-the-art Algorithmen-Vertreter (PC [136], Three Phase Dependency Analysis [18], Sparse Candidate [37], Optimal Reinsertion [94], Greedy Equivalent Search [19] and Greedy Search) der zuvor angesprochenen Kategorien. Das Prinzip der hybriden Lern-Methode MMHC, die sich in zwei Schritte gliedern lässt, stellt

sich wie folgt dar:
• Im ersten Schritt wird die Struktur (Skeleton) eines Bayes'schen Netzes ohne Orientierung der Kanten rekonstruiert.
• Im Anschluss wird ein Bayesian-Scoring Greedy Hill-Climbing Search (Auswahlverfahren) durchgeführt, um die Kanten zu richten (orientieren). Nachfolgend eine detailliertere Erläuterung der MMHC Schritte:
• In der ersten Phase ermittelt MMHC die Menge von Eltern-Kandidaten, wobei ein Elternteil-Kandidat von X jede andere Variable Y ist, die einen Bezug (Kante) zu X hat. Eine besondere Eigenschaft von MMHC ist es, die maximale Anzahl an Eltern automatisch zu erkunden und diese für jede Variable individuell zu setzen. Dies impliziert, dass MMHC bzgl. der Anzahl keine Vorgaben benötigt und somit im Gegensatz zu anderen gebräuchlichen Algorithmen autark funktioniert. Zur Identifikation des Gerüsts (Gerippes) kommen neben der Heuristik [144] auch bedingte Unabhängigkeitstests zum Einsatz, insbesondere ein lokaler Erkundungsalgorithmus mit dem Namen Max-Min Parents and Children (MMPC). Der Max-Min-Teil im Namen bezieht sich auf die Heuristik, die der Algorithmus nutzt, um das Set von Eltern und Kindern jeder Variablen V zu identifizieren. Des Weiteren steht PC für die Ausgabe Eltern (Parents) und Kinder (Children). MMPC geht bei der Erkundung in einem zweistufigen Ansatz vor:
◦ In der ersten Stufe werden mittels der Max-Min Heuristik [144] sequentiell potentielle PC-Kandidaten der Variablen eines Sets mit dem Namen PC-Kandidaten (CandidatesParentsChildren / CPC) hinzugefügt. Das CPC wird, wie nachstehend beschrieben, in jeder Iteration erweitert. Die Max-Min Heuristik selektiert Variablen, die die minimale Verbindung mit der Zielvariablen T (Target) relativ zu CPC maximieren. Diese Heuristik ist akzeptabel, da alle Variablen mit einer Kante von oder nach T und einige mehr (ohne Kante) CPC hinzugefügt werden. Diese Phase endet, wenn alle übrigen Variablen unabhängig von T sind.
◦ In der zweiten Stufe versucht MMPC die falsch-positiven Ergebnisse, die in der ersten Phase hinzugefügt wurden, zu entfernen. Dies geschieht mittels eines statistischen Unabhängigkeitstests, indem die einzelnen Kandidaten des CPC gegen T mittels $Ind(X;T|S)$ überprüft werden. Hierbei kommt der bedingte Unabhängigkeitstest, die G^2 Statistik, zum Einsatz, eine Alternative zum Goodness-of-fit Test.
• In der zweiten Phase des MMHC wird mit Hilfe des zuvor ermittelten Gerüsts (Gerippes) eine Greedy-Hill-climbing Suche im Raum des Bayes'schen Netzwerks durchgeführt, um die zuvor ermittelten Kanten auszurichten. Die Greedy-Suche ist mit einer Randbedingung versehen, d.h. es werden nur Kanten berücksichtigt, bei denen bereits Übergänge im Gerüst vorhanden sind. Die Suche beginnt mit einem leeren Graphen, wobei im Rahmen der Suche Operationen für Hinzufügen, Löschen und die Umkehr der Kanten angewandt werden. Es wird immer diejenige Operation gewählt, die zur größten Erhöhung bei der Auswertung führt. Anschließend wird die Suche in ähnlicher Weise rekursiv fortgesetzt.
Mit Hilfe von MMHC wird die Struktur von BN auf der Basis von Daten (Data-

Driven) gelernt. Diese kann auch direkt für Erweiterungen der Bayes'sche Netze, welche Zeitbedingungen berücksichtigen, wie z.B. Dynamische Bayes'sche Netze [96] [30] (DBN), verwendet werden. DBN ermöglichen neben der Repräsentation des statischen Verhaltens auch die Modellierung von Dynamiken.

4.3 Bayes'sche Netze mit kontinuierlicher Zeit

4.3.1 Grundlagen

Bayes'sche Netze mit kontinuierlicher Zeit (CTBN) [102] [99] wurden entwickelt, um die Einschränkungen der dynamischen Bayes'schen Netze (DBN) [96] zu überwinden. Die wesentlichste Einschränkung ist, dass DBN die Zeit per definitionem ausschließlich zu diskreten Zeitpunkten berücksichtigen. Dies impliziert, dass Ereignisse, die zwischen zwei solchen Zeitpunkten liegen, entweder nicht berücksichtigt werden oder einem der beiden Zeitpunkte zugeordnet werden müssen. Des Weiteren müssen bei DBN alle Variablen mit der gleichen Zeitbasis und auch in identischer Geschwindigkeit überwacht werden. Dies ergibt bei einer Vielzahl an Variablen sehr große Zustandsräume. Ein solches Vorgehen gestaltet eine Auswertung des tatsächlichen Verhaltens meist sehr schwierig. Im Gegensatz dazu stellen CTBN explizit temporale Dynamiken dar und ermöglichen es somit, die Wahrscheinlichkeitsverteilung über die Zeit beim Auftreten entsprechender Vorgänge zu berechnen. Die Basis der CTBN bilden homogene Markovprozesse [99], wobei BN benutzt werden, um komplexe dynamische Systeme zu beschreiben. Hierzu wird eine strukturierte Darstellung von Markovprozessen mit kontinuierlicher Zeit und endlichen Zuständen verwendet und der Gesamt-Markovprozess durch das CTBN faktorisiert. Bei der Faktorisierung wird für jede Systemvariable ein inhomogener Markovprozess gebildet, der die jeweilige Dynamik abbildet. Die Dynamik hängt dabei von anderen Systemvariablen ab, die mit der jeweiligen Variablen in Relation stehen. Man nutzt hierzu eine graphische Darstellung ähnlich der Abbildung 4.1, die mehrere System-Variablen als lokale Variablen repräsentiert. Systemvariablen, die Einfluß auf eine Variable ausüben, werden innerhalb dieser Darstellung als Eltern-Variablen (Parents) bezeichnet. Somit wird die Dynamik einer Variablen von der auftretenden Instantiierung der Eltern-Variablen innerhalb des Satzes beeinflusst. Die Abhängigkeiten werden als Übergänge im Graphen dargestellt. Das CTBN stellt die temporale Dynamik explizit als kontinuierliche Zeit mittels einer Exponentialfunktion dar und nutzt die strukturierte Darstellung zur Abbildung der Abhängigkeiten der stochastischen Variablen. Diese Exponentialverteilungsdarstellung verhindert die exponentielle Explosion des Zustandsraumes und ermöglicht die Verarbeitung von Ereignissen bei kontinuierlicher Zeit. Auf Grund der genannten Vorteile der CTBN werden diese in sehr vielfältigen Bereichen eingesetzt, wie Mensch-Maschine-Interaktion [100], Serverfarm Fehler [49], Überwachung eines Roboters [98], Netzwerk Einbruchserkennnung [155], Analyse sozialer Netzwerke [33] [32], Entwicklung von realistischen Diagnose-Szenarien für kardiogene Herzinsuffizienz [39].

Die Semantik von CTBN kann unter zwei Blickwinkeln betrachtet werden:
Zum einen kann das gesamte System als ein zusammengesetzter homogener Markov-Prozess beschrieben werden, bei dem mittels Kombinationsoperationen (Amalgamation) die faktorisierte Darstellung des CTBN (s. Kap 4.3.1 weiter oben) vereinigt wird. Zum anderen kann man das CTBN als generatives Modell betrachten, das über eine Menge von Ereignissen verfügt, die mit Variablen im System korrespondieren und bestimmte Werte zu gewissen Zeiten annehmen [102].
Bayes'sche Netze mit kontinuierlicher Zeit (CTBN) sind graphische Modelle, deren Knoten mit Zufallsvariablen assoziiert werden, wobei sich die jeweiligen Zustände kontinuierlich über die Zeit entwickeln. Die Entwicklung jeder Variablen hängt allerdings nicht von der Zeit, sondern vom Zustand ihrer Eltern im Graphen ab. Dies erfordert jedoch eine Erweiterung des Markov-Prozesses durch die Einführung der Notation des bedingten Markov-Prozesses. Dabei handelt es sich um eine spezielle Ausprägung des inhomogenen Markov-Prozesses [101], die sich wie folgt darstellt: Für jede gegebene Zufallsvariable (X) wird eine Funktion gebildet, deren Parameter durch Intensitäten (Raten) ausgedrückt werden. Der aktuelle Wert der Intensität wird von einer bestimmten Menge, den Eltern-Variablen (konditionierende Variablen), die ebenfalls in den Prozess involviert sind, bestimmt. Daher verändern sich die Intensitäten zwar über die Zeit, jedoch nicht als eine Funktion von dieser. Die Dynamik der Zufallsvariablen ($X(t)$) lässt sich bei bedingten Markov-Prozessen vollständig durch eine konditionierte Intensitäts-Matrix (CIM) beschreiben. Mit dieser ist es möglich, lokale Abhängigkeiten zwischen den Zufallsvariablen zu modellieren. Dabei werden nur direkte Abhängigkeiten, also zu den Eltern $PA(X)$ der Variablen, abgedeckt. Dies kann auch als Abhängigkeit 1. Ordnung bezeichnet werden. Die Fusion der dadurch entstandenen Menge von CIMs bzw. einer Teilmenge von dieser kann mittels Amalgamation erfolgen. Daraus ergibt sich ein einzelnes strukturiertes Modell des Gesamt- bzw. Teilsystems, welches den vollständigen Entwicklungsaspekt einer multivariaten Wahrscheinlichkeitsverteilung beschreibt. Eine Darstellung bringt Einschränkungen bei der Analyse komplexerer Abhängigkeiten mit sich.

Definition 1: Zur näheren Erläuterung darf darauf hingewiesen werden, dass ein CTBN nach [99] aus zwei Hauptkomponenten besteht. Angenommen X ist die Menge der lokalen Variablen $X_1, \ldots, X_n$, dann hat jedes X_i eine endliche Domäne, bestehend aus Werten $Val(X_i)$. Die beiden Komponenten des CTBN über X sind wie folgt definiert:

- Es gibt eine initiale Wahrscheinlichkeitsverteilung (P_X^0), welche ein Bayes'sches Netz B über X spezifiziert
- Es gibt Dynamiken, die als kontinuierliches Transitionsmodell dargestellt werden. Dieses definiert einen gerichteten Graphen (G) mit den Knoten $X_1, \ldots, X_n$, wobei $Pa(X_i)$ die Eltern von X_i in G beschreibt. Die Übergänge des Transitionssystems (Kanten) werden durch die Werte einer konditionierten Intensitätsmatrix (CIM), $Q_{X_{ij}|pa(X_i)}$, für jede Varia-

ble X_i beschrieben. Die CIM bestimmt die Entwicklung der vereinigten Wahrscheinlichkeitsverteilung über die Zeit.

Bayes'sche Netze bestehen per definitionem aus azyklischen Graphen. Dies ist im Lernprozess der Struktur begründet [102]. Hingegen werden bei CTBN die Übergänge, die den Effekt der Wirkung eines Variablenwertes auf den Wert einer anderen Variablen darstellen, über die Zeit ermittelt. Daher gibt es keine azyklische Einschränkung, d.h. es sind bei CTBN Zyklen im Graphen G möglich. Ein gerichteter Übergang vom Knoten X zum Knoten Y im Graphen (G) impliziert, dass die dynamische Entwicklung der Zufallsvariablen Y über die Zeit vom Zustand der Zufallsvariablen X abhängt. Bei zyklischen Graphen kann die Dynamik der Entwicklung der Zufallsvariablen X auch gleichzeitig vom Zustand der Zufallsvariablen Y abhängen. Graphische Modelle wie CTBN bieten neben der Beschreibung von Unabhängigkeiten bzw. Abhängigkeiten zwischen Variablen auch eine strukturierte Darstellungssprache für Markov-Prozesse bestehend aus mehreren Variablen. Dies wird auch dadurch unterstützt, dass bei CTBN die Unabhängigkeiten zwischen Verteilungen über die vollständigen Trajektorien der Variablen abgebildet werden.
Sofern ein Markov-Prozess aus mehreren Variablen besteht, wird in einem CTBN ähnlich wie beim Bayes'schen Netz jede Variable X_i von allen anderen Variablen durch ihre Markovdecke (Markov-Blanket) getrennt. Die Markovdecke der Variablen X_i wird definiert als die Eltern von X_i, die Kinder von X_i und die Eltern der Kinder. In einem CTBN ist X_i unabhängig von allen anderen Variablen dargestellt. Daher ist die komplette Trajektorie der Markov-Kette von der Variablen X_i im CTBN nur durch Fusion darstellbar.

4.3.2 CTBN-Modell

Bayes'sche Netze mit kontinuierlicher Zeit (CTBN) basieren auf homogenen Markov-Prozessen mit kontinuierlicher Zeit und endlichen Zuständen. Es wird angenommen dass X eine Prozessvariable ist, deren Zustand sich kontinuierlich über die Zeit ändert. Die Domäne von X_i, die mittels $Val(X_i) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_n}\}$ beschrieben wird, repräsentiert dann die Menge an Instanzen von X, wobei die Anzahl der Elemente (Kardinalität) mittels $Card(X_i)$ angegeben wird. Das Verhalten von X wird durch einen homogenen Markov-Prozess mit kontinuierlichem Zeitverhalten und n-endlichen Zuständen $\{x_1, x_2, \ldots, x_n\}$ abgebildet und mittels eines Transitionsmodells beschrieben. Dabei erfolgt die Repräsentation durch eine zeitinvariante Intensitätsmatrix Q und eine initiale Verteilung P_x^0 über $Val(X)$, d.h. bei gegebener Matrix Q können die Verweil- bzw. Transitions-Zeiten ermittelt werden. Die Elemente von Q $(q_{x_i x_j})$ geben die Intensität (Rate) an, mit der X von x_i nach x_j wechselt. Hierbei stellen die Diagonalelemente $(q_{x_i x_i})$ einen Sonderfall dar: Sie enthalten den negativen Wert der Wahrscheinlichkeit für das Verlassen des Zustandes x_i. Diese Wahrscheinlichkeit kann wie folgt berechnet werden:

$$q_{x_i} = \sum_{i \neq j} q_{ij} \tag{4.1}$$

Die Zeit (t) folgt einer Exponentialverteilung mit dem Parameter q_{x_i}. Somit ergibt sich die Wahrscheinlichkeitsdichtefunktion f (siehe Gleichung 4.2). Die korrespondierende Verteilungsfunktion ist Gleichung 4.3 zu entnehmen.

$$f(t) = q_{x_i} \exp(-q_{x_i} t) \text{ für } t > 0 \tag{4.2}$$

$$F(t) = 1 - \exp(-q_{x_i} t) \text{ für } t > 0 \tag{4.3}$$

Des Weiteren kann der Erwartungswert für ein Verlassen des Zustandes x, mit $1/q_{x_i}$ bestimmt werden. Die Wahrscheinlichkeit dafür, dass X in den Zustand x_0 wechselt, ist durch Gleichung 4.4 gegeben.

$$\Theta_{x_0} = \frac{q_{xx_0}}{q_x} \tag{4.4}$$

Die Verteilung $P_x(t)$ über den Wert von X zur Zeit t mit der initialen Verteilung P_x^0 wird durch

$$P_x(t) = P_x^0 exp(Q_x(t)) \tag{4.5}$$

angegeben. Um komplexere Prozesse darstellen zu können, muss die bisher genutzte Notation durch Verwendung sogenannter bedingter Markov-Prozesse erweitert werden. Hierbei modelliert ein CTBN die vereinigte Dynamik von verschiedenen lokalen Variablen unter der Annahme, dass das Transitionsmodell aller Variablen in X ein Markovprozess ist, dessen Parameter von einer Untermenge anderer Variablen $C \subset X$ abhängen, so dass gilt $S \backslash C$. Daher müssen sich beim bedingten Markov-Prozess die Intensitätsmatrizen wie eine Funktion der konditionierenden Variablen C verhalten. Dies erfordert eine Erweiterung der Intensitätsmatrix zu einer konditionierten Intensitätsmatrix (CIM). Die CIM ist als eine Menge von Intensitätsmatrizen definiert, die für jede Werteinstanz c aus dem Variablenset C eine Intensitätsmatrix enthält. Daher kann die CIM auch als dreidimensionaler Tensor angesehen werden, bei dem die Tiefe der dritten Dimension von der Anzahl der Werteinstanzen c abhängt. Die Entwicklung der konditionierten Variablen X wird daher direkt von den Instanzwerten der Variablen in C vorgegeben. Dadurch variieren die Intensitäten über die Zeit, jedoch nicht als eine direkte Funktion der Zeit. Da die BN-Terminologie zugrunde gelegt ist, werden die Variablen, die zum Set C gehören, auch als die Eltern ($pa(X)$) der Zufallsvariablen X bezeichnet. Für den Fall, dass $pa(X)$ leer ist, ist die CIM eine einfache Standard-Intensitäts-Matrix. Häufig ist das Verhalten der Prozesse nicht vorab spezifiziert, daher müssen die CIM Parameter q_x gelernt werden. Dies kann sowohl mit kompletten als auch mit partiellen Messdaten erfolgen. Hierzu muss jedoch die graphische Struktur des CTBN bekannt sein. Durch Lernen können neben den konditionierten Elementen von $Q(q_x|c)$ auch die Multinominalverteilungen Θ_{x,x_0} wie folgt bestimmt werden:

$$q_x = \frac{M[x]}{T[x]} \tag{4.6}$$

$$\Theta_{x,x_0} = \frac{M[x;x']}{M[x]} \tag{4.7}$$

Die Statistik des Hidden-Markov-Prozess-Modells (HMP) ist hierbei mit $M[x; x']$, $M[x]$, $T[x]$ definiert. Die Anzahl der Wechsel, in denen X vom Zustand x nach x' übergeht, wird mit $M[x; x']$ angegeben. Die gesamte Anzahl von Zustandswechseln ($M[x]$), in denen das System den Zustand x verlässt, wird mit Gleichung 4.8 berechnet. $T[x]$ ist die komplette Zeit, in der X im Zustand x verweilt.

$$M[x] = \sum_{x'} M[x; x'] \tag{4.8}$$

Ein CTBN mit der Struktur aus Abbildung 4.1 wurde zur Veranschaulichung mit künstlichen Daten gelernt. Dabei haben die vier Variablen (X_1, X_2, X_3, X_4) die Kardinalitäten $(2, 3, 3, 2)$. Die auf der Basis der angeführten Wertigkeiten ermittelten Intensitäts-Matrizen des CTBN werden nachfolgend dargestellt. Es ist zu erkennen, dass die Kardinalität einer Variablen die Dimension der korrespondierenden Intensitäts-Matrizen bestimmt. Des Weiteren wird die Anzahl der Intensitäts-Matrizen, welche einer Variablen zugeordnet ist, durch die Kardinalitäten der beschränkenden Variablen vorgegeben. Deutlich ist dies bei den Matrizen der Variablen X_2, X_3 und X_4 festzustellen. Hierbei gilt, dass sich die Anzahl der Matrizen durch $\prod_j Card(PA_j(X_i))$ ergibt, d.h. im Falle von X_3 berechnet sich die „dritte Dimension“ aus $Card(X_1) = 2$ und $Card(X_2) = 3$ zu $2 * 3 = 6$. Somit existieren 6 konditionierende Intensitäts-Matrizen für X_3. Wenn die Variable nicht konditioniert ist, d.h. keine einwirkenden Variablen ($pa(x) = \{\}$) existieren, ist trotzdem eine Intensitäts-Matrix vorhanden wie bei X_1 zu erkennen ist.

Intensitätsmatrizen für das oben eingeführte Beispiel:

$$\begin{bmatrix} -13.1012 & 13.1012 \\ 17.8237 & -17.8237 \end{bmatrix}$$

Intensitäts-Matrix X1

$$\begin{bmatrix} -21.17 & 16.4656 & 4.70444 \\ 13.0507 & -23.2013 & 10.1506 \\ 8.8219 & 7.56163 & -16.3835 \end{bmatrix}$$

Intensitäts-Matrix X2,0

$$\begin{bmatrix} -19.4949 & 5.31679 & 14.1781 \\ 3.27951 & -9.83852 & 6.55901 \\ 11.5996 & 34.7987 & -46.3983 \end{bmatrix}$$

Intensitäts-Matrix X2,1

$$\begin{bmatrix} -34.8068 & 34.8068 & 1e-06 \\ 7.46157 & -22.3847 & 14.9231 \\ 4.27991 & 12.8397 & -17.1196 \end{bmatrix}$$

Intensitäts-Matrix X3,0

$$\begin{bmatrix} -26.6821 & 22.2351 & 4.44701 \\ 1e-06 & -35.9411 & 35.9411 \\ 11.7229 & 7.81525 & -19.5381 \end{bmatrix}$$

Intensitäts-Matrix X3,1

$$\begin{bmatrix} -22.7256 & 15.1504 & 7.57518 \\ 10.2559 & -35.8956 & 25.6397 \\ 17.3471 & 4.33678 & -21.6839 \end{bmatrix}$$

Intensitäts-Matrix X3,2

$$\begin{bmatrix} -9.90055 & 4.95027 & 4.95027 \\ 16.7851 & -22.3801 & 5.59503 \\ 1e-06 & 1e-06 & -2e-06 \end{bmatrix}$$

Intensitäts-Matrix X3,3

$$\begin{bmatrix} -29.6626 & 22.2469 & 7.41565 \\ 10.0408 & -13.3878 & 3.34694 \\ 13.8947 & 2.77894 & -16.6736 \end{bmatrix}$$

Intensitäts-Matrix X3,4

$$\begin{bmatrix} -139.86 & 139.86 & 1e-06 \\ 14.3699 & -35.9247 & 21.5548 \\ 1e-06 & 52.7983 & -52.7983 \end{bmatrix}$$

MIntensitäts-Matrix X3,5

$$\begin{bmatrix} -19.5973 & 19.5973 \\ 38.3534 & -38.3534 \end{bmatrix}$$

Intensitäts-Matrix X4,0

$$\begin{bmatrix} -7.28654 & 7.28654 \\ 15.6055 & -15.6055 \end{bmatrix}$$

Intensitäts-Matrix X4,1

$$\begin{bmatrix} -17.4368 & 17.4368 \\ 58.0552 & -58.0552 \end{bmatrix}$$

Intensitäts-Matrix X4,2

$$\begin{bmatrix} -35.3607 & 35.3607 \\ 13.1423 & -13.1423 \end{bmatrix}$$

Intensitäts-Matrix X4,3

$$\begin{bmatrix} -15.0983 & 15.0983 \\ 27.571 & -27.571 \end{bmatrix}$$

Intensitäts-Matrix X4,4

$$\begin{bmatrix} -24.2895 & 24.2895 \\ 10.9778 & -10.9778 \end{bmatrix}$$

Intensitäts-Matrix X4,5

$$\begin{bmatrix} -19.7291 & 19.7291 \\ 11.8519 & -11.8519 \end{bmatrix}$$

Intensitäts-Matrix X4,6

$$\begin{bmatrix} -11.7415 & 11.7415 \\ 22.8964 & -22.8964 \end{bmatrix}$$

Intensitäts-Matrix X4,7

$$\begin{bmatrix} -1e-06 & 1e-06 \\ 35.5391 & -35.5391 \end{bmatrix}$$

Intensitäts-Matrix X4,8

4.3.3 Amalgamation (Fusion der Teilprozesse)

Um das Verhalten des CTBN beschreiben zu können, ist es notwendig, einen einzelnen Markov-Prozess zu konstruieren. Hierfür wird eine Operation auf den konditionierten Intensitäts-Matrizen (CIM), eine Amalgamation [99], benötigt, die mehrere CIM kombiniert, um eine größere CIM zu generieren. Diese neue konditionierte Intensitäts-Matrix beinhaltet die Intensitäten der Variablen in S, konditioniert durch die aus C. Wie bereits zuvor erwähnt, besteht eine wesentliche Annahme darin, dass Variablen nicht zeitgleich den Zustand wechseln. Daher sind alle Intensitäten Null, wenn sie aus zwei simultanen Veränderungen hervorgehen. Für die Diskussion der Struktur einer einzelnen Matrix über mehrere Variable wird eine Funktion benötigt, welche die Reihen der größeren Matrix auf die Ausprägungen (Instanzen) der Variablen abbildet. Hierfür wird eine Ordnung genutzt, die sowohl für die Variablen wie auch für deren Instanzen eine eindeutige Sequenz definiert. Auf dieser Basis kann zunächst die Expansion einer CIM in eine einzelne Matrix über ein größeres Set von Variablen und danach die Amalgamationsoperation, welche zur vereinigten Intensitätsmatrix führt, definiert werden. Diese stellt das Verhalten des CTBN als einen homogenen Markov-Prozess dar. Die Anzahl der Zustände des vereinigten Markov-Prozesses ergibt sich aus der Gleichung 4.9, hierbei gibt n_i die Anzahl der Zustände jeder Variablen im CTBN an. Daraus ergibt sich die Dimension der vereinigten Intensitätsmatrix Q als eine $n \times n$ Matrix.

$$n = \prod n_i \tag{4.9}$$

Die vereinigte Intensitätsmatrix kann in drei Schritten erzeugt werden: Es wird angenommen, dass i und j ein beliebiges Zustandspaar im vereinten Prozess bezeichnen, wobei sie sich nur in einem Wert unterscheiden. X soll die differierende Variable sein und V die Menge der Eltern von X. Somit ist v die Instanz von V im Zustand i und j. Der Wert des nicht diagonalen Elements q_{ij} in Q wird als korrespondierende Intensität von i und j in der CIM gesetzt. Alle anderen nicht diagonalen Elemente sind Null, da per definitionem im CTBN zum gleichen Zeitpunkt nicht mehr als eine Variable den Zustand wechseln kann. Die Diagonalelemente werden als Ausgleich berechnet, um jede Zeilensumme auf Null zu bringen. Die Abbildung zwischen Zeilen- bzw. Spalten-Nummern und Instanzen der Variablen wird durch eine Variablenordnung [102] bereitgestellt. Bevor die Amalgamation durchgeführt werden kann, muss die gewünschte CIM in einen erweiterten Zustandsraum, den vereinigten Zustandsraum, überführt werden. Dies erfordert eine Erweiterung der CIM. Die Amalgamation kann nun basierend auf der erweiterten CIM wie folgt definiert werden:

Definition 2. [99] Die Amalgamation ist eine Operation, welche aus zwei CIMs $Q_{S1|C1}$ und $Q_{S2|C2}$ eine neue CIM $Q_{S|C}$ berechnet, wobei $S = S1 \cup S2$ und $C = (C1 \cup C2) \backslash S$ gilt. Gegeben ist eine feste und strikte Ordnung $\delta_{S,C}$ (der Variablen und deren Ausprägungen /Instanzen), welche zwingend eingehalten werden muss. Die Expansion von $Q_{S1|C1}$ und $Q_{S2|C2}$ in eine einzelne Matrix über $S \times C$ definiert die vereinigte Matrix als Summe $Q_{S|C} = Q_{S1|C1} + Q_{S2|C2}$.

Definition 3. [102] Die Inverse der Amalgamation wird mit Hilfe der Matrix-Subtraktion definiert.
Anstelle des ursprünglich beschriebenen Verfahrens der Amalgamation [102] wird eine konzeptionell einfachere und verbesserte Version [126] angewandt, die für die Berechnung der vereinigten Intensitätsmatrix die Summe von Kronecker-Produkten (siehe Gleichung 4.10) verwendet. Hierzu muss die Ordnung ($\delta_{S,C}$), die sowohl die Variablen wie auch deren Ausprägungen abdeckt, definiert und strikt eingehalten werden.
Kronecker-Produkt-Beispiel mit beliebig gewählten Matrizen Q_1 und Q_2:

$$Q_1 \otimes Q_2 = \begin{bmatrix} q_{11}Q_2 & \cdots & q_{1n}Q_2 \\ \vdots & \ddots & \vdots \\ q_{m1}Q_2 & \cdots & q_{mn}Q_2 \end{bmatrix} \tag{4.10}$$

Für die Amalgamation muss ein in der Schreibweise umständlicher, aber konzeptionell einfacher Begriff definiert werden. Dazu wird eine Hilfsmatrix Δ eingeführt. Bei Δ sind alle Elemente Null mit Ausnahme einer einzelnen 1 an der Stelle i,j. Sofern die zu bearbeitende Variable (X_i) keine Abhängigkeiten aufweist, d.h. zu anderen in der Ordnung aufgeführten Variablen keine Abhängigkeit hat, wird anstelle von Δ die Einheitsmatrix (I) verwendet. Das Resultat der Kronecker-Produktberechnung für eine individuelle Intensitätsmatrix wird mit $Q_{X_{ij}}^{par_i}$ bezeichnet. Die Amalgamation mittels Kroneckerprodukt muss für alle Intensitätsmatrizen ($Q_{X|C}$) im Set von jeder Variablen des CTBN durchgeführt werden, wobei die jeweiligen Ergebnisse pro Variable X_i aufsummiert werden. Die resultierende Matrix für die Variable X_i wird mit $\tilde{Q}_{X_{ij}|par_i}$ bezeichnet. Im Falle, dass die Variable durch Eltern konditioniert wird, hat par_i den Wert x_k und die Matrix wird mit Δ bezeichnet. Sofern es keine Konditionierung von X_i gibt, entspricht diese der Identitätsmatrix (I). Die Kronecker-Produktberechnung platziert die Elemente von $Q_{X_{ij}}^{par_i}$ in die jeweils relevanten Einträge der vereinigten Intensitätsmatrix. Die vereinigte Intensitätsmatrix wird entsprechend Gleichung 4.11 definiert, wobei $\tilde{Q}_{X_i}$ aus Gleichung 4.12 entnommen wurde.

$$Q = \sum_i \tilde{Q}_{X_i} \tag{4.11}$$

$$\tilde{Q}_{X_i} = \sum_{par_i} \tilde{Q}_{X_i|par_i} \tag{4.12}$$

Das folgende kurze Beispiel soll die Details der Amalgamation anhand eines exemplarischen CTBN-Netzwerks (siehe Abbildung 4.1), bestehend aus vier Va-

riablen (X_1, X_2, X_3, X_4), aufzeigen.

$$\tilde{Q}_{X_1} = Q_{X_1} \otimes I \otimes I \otimes I \tag{4.13}$$

$$\tilde{Q}_{X_2} = \sum_{x_1} \Delta_{X1,X1} \otimes Q_{X_2|x_1} \otimes I \otimes I \tag{4.14}$$

$$\tilde{Q}_{X_3} = \sum_{x_1,x_2} \Delta_{X1,X1} \otimes \Delta_{x2,x2} \otimes Q_{X_3|x_1,x_2} \otimes I \tag{4.15}$$

$$\tilde{Q}_{X_4} = \sum_{x_2,x_3} I \otimes \Delta_{x2,x2} \otimes \Delta_{x3,x3} \otimes Q_{X_4|x_2,x_3} \tag{4.16}$$

$$Q = \tilde{Q}_{X_1} + \tilde{Q}_{X_2} + \tilde{Q}_{X_3} + \tilde{Q}_{X_4} \tag{4.17}$$

Der so fusionierte Markov-Prozess (Q) entspricht einer Markov-Kette mit kontinuierlicher Zeit (CTMC). Nachfolgend werden exemplarisch Intensitätsmatrizen dargestellt, bei denen die Amalgamation auf der Basis der in Kapitel 4.3.2 aufgeführten Intensitäts-Matrizen durchgeführt wurde. Die für die Berechnung der Gesamtmatrix benötigten Matrizen sind dem Anhang zu entnehmen. Die Dimension der Intensitätsmatrizen ergibt sich aus Gleichung 4.9. Durch Einsetzen der Kardinalitäten $(2, 3, 3, 2)$ der vier Variablen (X_1, X_2, X_3, X_4) resultiert diese zu 36. Es ist zu beachten, dass bei der Darstellung der Intensitätsmatrizen eine Ordnung [127] [34] eingehalten werden muss, die in Abhängigkeit der Variablenbelegungen die Bedeutung der Matrixeinträge in der Darstellung beeinflusst. Die Ordnung der Variablenbelegung ist bei der ersten Beispiel-Matrix und der fusionierten Gesamtmatrix veranschaulicht dargestellt. Bei allen anderen zum Beispiel gehörenden Intensitätsmatrizen findet diese Ordnung ebenfalls Anwendung. Für eine übersichtliche (kompakte) Darstellung und zur besseren Lesbarkeit wurden die Variablenbezeichner (X_1, X_2, X_3, X_4) in den nachfolgenden Beispielen durch Buchstaben (a, b, c, d) ersetzt. Der jeweilige Index gibt die Belegung (Instanz) der Variablen an. So bedeutet beispielsweise a_0, dass die Variable X_1 den Wert 0 und c_2, dass X_3 den Wert 2 enthält.

Amalgemations-Matrix $X1$

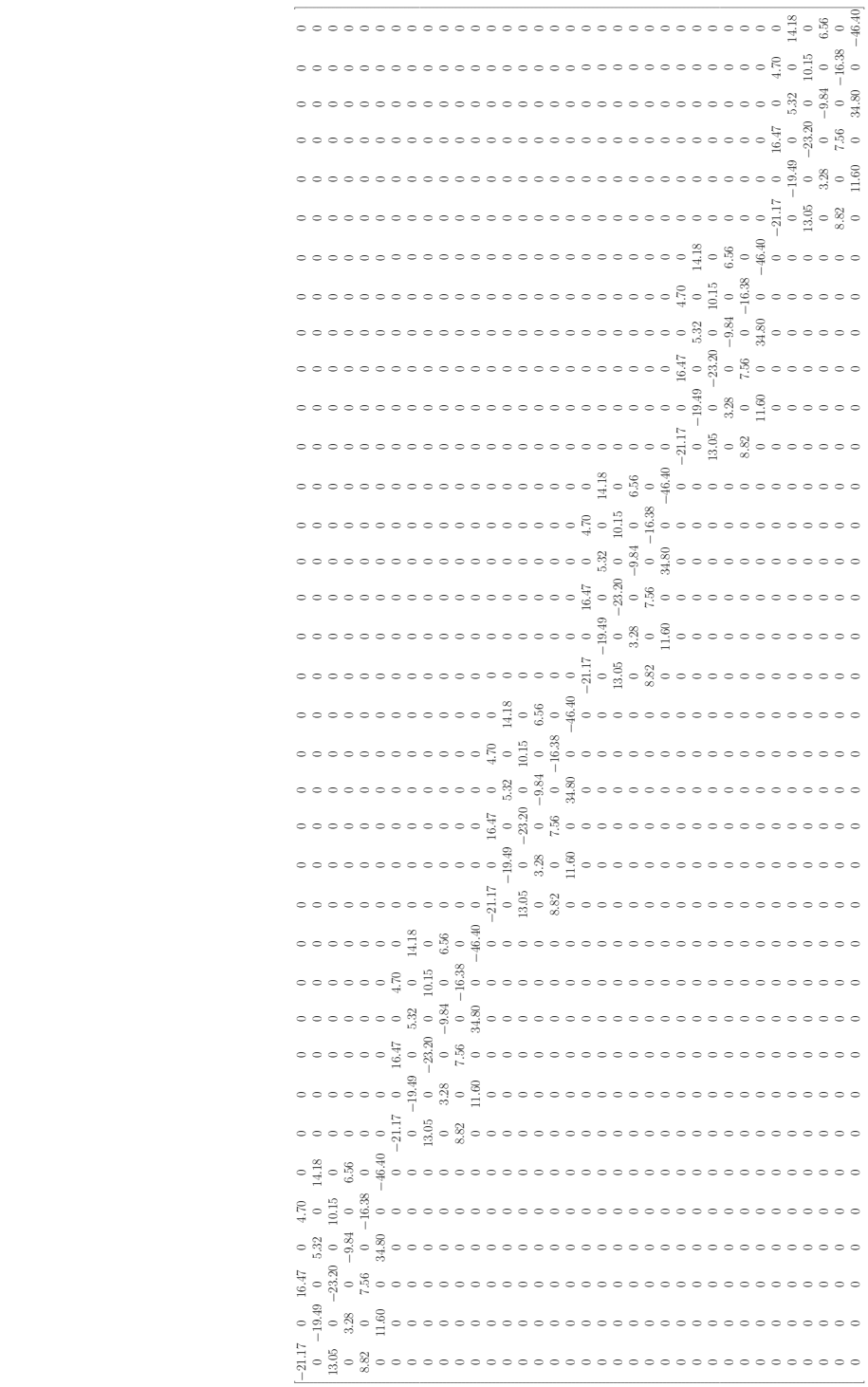
Var.-belegung	$a_0b_0c_0d_0$	$a_1b_0c_0d_0$	$a_0b_1c_0d_0$	$a_1b_1c_0d_0$	$a_0b_2c_0d_0$	$a_1b_2c_0d_0$	$a_0b_0c_1d_0$	$a_1b_0c_1d_0$	$a_0b_1c_1d_0$	$a_1b_1c_1d_0$	$a_0b_2c_1d_0$	$a_1b_2c_1d_0$	$a_0b_0c_2d_0$	$a_1b_0c_2d_0$	$a_0b_1c_2d_0$	$a_1b_1c_2d_0$	$a_0b_2c_2d_0$	$a_1b_2c_2d_0$	$a_0b_0c_0d_1$	$a_1b_0c_0d_1$	$a_0b_1c_0d_1$	$a_1b_1c_0d_1$	$a_0b_2c_0d_1$	$a_1b_2c_0d_1$	$a_0b_0c_1d_1$	$a_1b_0c_1d_1$	$a_0b_1c_1d_1$	$a_1b_1c_1d_1$	$a_0b_2c_1d_1$	$a_1b_2c_1d_1$	$a_0b_0c_2d_1$	$a_1b_0c_2d_1$	$a_0b_1c_2d_1$	$a_1b_1c_2d_1$	$a_0b_2c_2d_1$	$a_1b_2c_2d_1$
$a_0b_0c_0d_0$	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_0d_0$	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_0d_0$	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_1c_0d_0$	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_2c_0d_0$	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_2c_0d_0$	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_0c_1d_0$	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_1d_0$	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_1d_0$	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_1c_1d_0$	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_2c_1d_0$	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_2c_1d_0$	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_0c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_1c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_2c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_2c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_0c_0d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_0d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_0d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_1c_0d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_2c_0d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_2c_0d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_0c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0	0	0
$a_1b_1c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0	0	0
$a_0b_2c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0	0	0
$a_1b_2c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0	0	0
$a_0b_0c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0	0	0
$a_1b_0c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0	0	0
$a_0b_1c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10	0	0
$a_1b_1c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82	0	0
$a_0b_2c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.10	13.10
$a_1b_2c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−17.82

−21.17	0	16.47	0	4.70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13.05	0	−23.20	0	10.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8.82	0	7.56	0	−16.38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	−21.17	0	16.47	0	4.70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	13.05	0	−23.20	0	10.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	8.82	0	7.56	0	−16.38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	−21.17	0	16.47	0	4.70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	13.05	0	−23.20	0	10.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	8.82	0	7.56	0	−16.38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−21.17	0	16.47	0	4.70	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13.05	0	−23.20	0	10.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8.82	0	7.56	0	−16.38	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−21.17	0	16.47	0	4.70	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13.05	0	−23.20	0	10.15	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8.82	0	7.56	0	−16.38	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−21.17	0	16.47	0	4.70	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13.05	0	−23.20	0	10.15	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8.82	0	7.56	0	−16.38	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Amalgemations-Matrix $X2,1$

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -19.49 & 0 & 5.32 & 0 & 14.18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3.28 & 0 & -9.84 & 0 & 6.56 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 11.60 & 0 & 34.80 & 0 & -46.40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.49 & 0 & 5.32 & 0 & 14.18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.28 & 0 & -9.84 & 0 & 6.56 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 11.60 & 0 & 34.80 & 0 & -46.40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.49 & 0 & 5.32 & 0 & 14.18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.28 & 0 & -9.84 & 0 & 6.56 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11.60 & 0 & 34.80 & 0 & -46.40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.49 & 0 & 5.32 & 0 & 14.18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.28 & 0 & -9.84 & 0 & 6.56 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11.60 & 0 & 34.80 & 0 & -46.40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.49 & 0 & 5.32 & 0 & 14.18 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.28 & 0 & -9.84 & 0 & 6.56 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11.60 & 0 & 34.80 & 0 & -46.40 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.49 & 0 & 5.32 & 0 & 14.18 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.28 & 0 & -9.84 & 0 & 6.56 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11.60 & 0 & 34.80 & 0 & -46.40
\end{bmatrix}
$$

Amalgemations-Matrix $X2,2$



Fusion der Amalgemations-Matrizen $X2$

Gesamtsystem Matrix Q

Var.-belegung	$a_0b_0c_0d_0$	$a_1b_0c_0d_0$	$a_0b_1c_0d_0$	$a_1b_1c_0d_0$	$a_0b_2c_0d_0$	$a_1b_2c_0d_0$	$a_0b_0c_1d_0$	$a_1b_0c_1d_0$	$a_0b_1c_1d_0$	$a_1b_1c_1d_0$	$a_0b_2c_1d_0$	$a_1b_2c_1d_0$	$a_0b_0c_2d_0$	$a_1b_0c_2d_0$	$a_0b_1c_2d_0$	$a_1b_1c_2d_0$	$a_0b_2c_2d_0$	$a_1b_2c_2d_0$	$a_0b_0c_0d_1$	$a_1b_0c_0d_1$	$a_0b_1c_0d_1$	$a_1b_1c_0d_1$	$a_0b_2c_0d_1$	$a_1b_2c_0d_1$	$a_0b_0c_1d_1$	$a_1b_0c_1d_1$	$a_0b_1c_1d_1$	$a_1b_1c_1d_1$	$a_0b_2c_1d_1$	$a_1b_2c_1d_1$	$a_0b_0c_2d_1$	$a_1b_0c_2d_1$	$a_0b_1c_2d_1$	$a_1b_1c_2d_1$	$a_0b_2c_2d_1$	$a_1b_2c_2d_1$
$a_0b_0c_0d_0$	−88.68	13.10	16.47	0	4.70	0	34.81	0	0	0	0	0	0	0	0	0	0	0	19.60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_0d_0$	17.82	−83.60	0	5.32	0	14.18	0	22.24	0	0	0	0	0	4.45	0	0	0	0	0	19.60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_0d_0$	13.05	0	−66.31	13.10	10.15	0	0	0	15.15	0	0	0	0	0	7.58	0	0	0	0	0	7.29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_1c_0d_0$	0	3.28	17.82	−44.85	0	6.56	0	0	0	4.95	0	0	0	0	0	4.95	0	0	0	0	7.29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_2c_0d_0$	8.82	0	7.56	0	−76.58	13.10	0	0	0	0	22.25	0	0	0	0	0	7.42	0	0	0	0	0	17.44	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_2c_0d_0$	0	11.60	0	34.80	17.82	−221.52	0	0	0	0	0	139.86	0	0	0	0	0	0	0	0	0	0	0	17.44	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_0c_1d_0$	7.46	0	0	0	0	0	−92.02	13.10	16.47	0	4.70	0	14.92	0	0	0	0	0	0	0	0	0	0	0	35.36	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_1d_0$	0	0	0	0	0	0	17.82	−108.62	0	5.32	0	14.18	0	35.94	0	0	0	0	0	0	0	0	0	0	0	35.36	0	0	0	0	0	0	0	0	0	0
$a_0b_1c_1d_0$	0	0	10.26	0	0	0	13.05	0	−87.30	13.10	10.15	0	0	0	25.64	0	0	0	0	0	0	0	0	0	0	0	15.10	0	0	0	0	0	0	0	0	0
$a_1b_1c_1d_0$	0	0	0	16.79	0	0	0	3.28	17.82	−65.14	0	6.56	0	0	0	5.60	0	0	0	0	0	0	0	0	0	0	0	15.10	0	0	0	0	0	0	0	0
$a_0b_2c_1d_0$	0	0	0	0	10.04	0	8.82	0	7.56	0	−67.16	13.10	0	0	0	0	3.35	0	0	0	0	0	0	0	0	0	0	0	24.29	0	0	0	0	0	0	0
$a_1b_2c_1d_0$	0	0	0	0	0	14.37	0	11.60	0	34.80	17.82	−124.44	0	0	0	0	0	21.55	0	0	0	0	0	0	0	0	0	0	0	24.29	0	0	0	0	0	0
$a_0b_0c_2d_0$	4.28	0	0	0	0	0	12.84	0	0	0	0	0	−71.12	13.10	16.47	0	4.70	0	0	0	0	0	0	0	0	0	0	0	0	0	19.73	0	0	0	0	0
$a_1b_0c_2d_0$	0	11.72	0	0	0	0	0	7.82	0	0	0	0	17.82	−76.59	0	5.32	0	14.18	0	0	0	0	0	0	0	0	0	0	0	0	0	19.73	0	0	0	0
$a_0b_1c_2d_0$	0	0	17.35	0	0	0	0	0	4.34	0	0	0	13.05	0	−69.73	13.10	10.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11.74	0	0	0
$a_1b_1c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	3.28	17.82	−39.40	0	6.56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11.74	0	0
$a_0b_2c_2d_0$	0	0	0	0	13.89	0	0	0	0	0	2.78	0	8.82	0	7.56	0	−46.16	13.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_2c_2d_0$	0	0	0	0	0	0	0	0	0	0	0	52.80	0	11.60	0	34.80	17.82	−117.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_0b_0c_0d_1$	38.35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−107.43	13.10	16.47	0	4.70	0	34.81	0	0	0	0	0	0	0	0	0	0	0	0
$a_1b_0c_0d_1$	0	38.35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−102.35	0	5.32	0	14.18	0	22.24	0	0	0	0	0	4.45	0	0	0	0	0
$a_0b_1c_0d_1$	0	0	15.61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13.05	0	−74.63	13.10	10.15	0	0	0	15.15	0	0	0	0	0	7.58	0	0	0	0
$a_1b_1c_0d_1$	0	0	0	15.61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3.28	17.82	−53.17	0	6.56	0	0	0	4.95	0	0	0	0	0	4.95	0	0	0
$a_0b_2c_0d_1$	0	0	0	0	58.06	0	0	0	0	0	0	0	0	0	0	0	0	0	8.82	0	7.56	0	−117.20	13.10	0	0	0	0	22.25	0	0	0	0	0	7.42	0	0
$a_1b_2c_0d_1$	0	0	0	0	0	58.06	0	0	0	0	0	0	0	0	0	0	0	0	0	11.60	0	34.80	17.82	−262.14	0	0	0	0	0	139.86	0	0	0	0	0	0	0
$a_0b_0c_1d_1$	0	0	0	0	0	0	13.14	0	0	0	0	0	0	0	0	0	0	0	7.46	0	0	0	0	0	−69.80	13.10	16.47	0	4.70	0	14.92	0	0	0	0	0	0
$a_1b_0c_1d_1$	0	0	0	0	0	0	0	13.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.82	−86.40	0	5.32	0	14.18	0	35.94	0	0	0	0	0
$a_0b_1c_1d_1$	0	0	0	0	0	0	0	0	27.57	0	0	0	0	0	0	0	0	0	0	0	10.26	0	0	0	13.05	0	−99.77	13.10	10.15	0	0	0	25.64	0	0	0	0
$a_1b_1c_1d_1$	0	0	0	0	0	0	0	0	0	27.57	0	0	0	0	0	0	0	0	0	0	0	16.79	0	0	0	3.28	17.82	−77.61	0	6.56	0	0	0	5.60	0	0
$a_0b_2c_1d_1$	0	0	0	0	0	0	0	0	0	0	10.98	0	0	0	0	0	0	0	0	0	0	0	10.04	0	8.82	0	7.56	0	−53.85	13.10	0	0	0	0	3.35	0
$a_1b_2c_1d_1$	0	0	0	0	0	0	0	0	0	0	0	10.98	0	0	0	0	0	0	0	0	0	0	0	14.37	0	11.60	0	34.80	17.82	−111.12	0	0	0	0	0	21.55
$a_0b_0c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	11.85	0	0	0	0	0	4.28	0	0	0	0	0	12.84	0	0	0	0	0	−63.24	13.10	16.47	0	4.70	0
$a_1b_0c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	11.85	0	0	0	0	0	11.72	0	0	0	0	0	7.82	0	0	0	0	17.82	−68.71	0	5.32	0	14.18
$a_0b_1c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22.90	0	0	0	0	0	17.35	0	0	0	0	0	4.34	0	0	0	13.05	0	−80.88	13.10	10.15	0
$a_1b_1c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22.90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3.28	17.82	−50.56	0	6.56
$a_0b_2c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35.54	0	0	0	0	0	13.89	0	0	0	0	0	2.78	0	8.82	0	7.56	0	−81.70	13.10
$a_1b_2c_2d_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35.54	0	0	0	0	0	0	0	0	0	0	0	52.80	0	11.60	0	34.80	17.82	−152.56

4.4 Markov-Ketten mit kontinuierlicher Zeit

4.4.1 Allgemeine Definition

Die Markov-Kette [103] ist eine spezielle Klasse von stochastischen Prozessen, die sich sehr gut eignet, um zufällige Zustandsänderungen eines Systems zu modellieren. Markov-Ketten verhalten sich wie Transitionssysteme [7], jedoch mit dem einzigen Unterschied, dass eine nicht-deterministische Auswahl des Nachfolgezustands durch eine wahrscheinlichkeitstheoretische ersetzt wird. Vereinfacht gesagt: der Nachfolgezustand s wird gemäß einer Wahrscheinlichkeitsverteilung ausgewählt. Die Übergangswahrscheinlichkeiten hängen nur vom aktuellen Zustand und nicht von der Vergangenheit ab. Dies wird auch als gedächnislose Eigenschaft bzw. Markov-Eigenschaft bezeichnet. Ein einfaches Beispiel einer Markov-Kette mit den Zuständen „a, b, c“ ist in Abbildung 4.2 dargestellt. Die Matrixnotation der Wahrscheinlichkeitsverteilung der in Grafik 4.2 abgebildeten Markov-Kette ist der Gleichung 4.18 zu entnehmen.

$$\begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1/3 & 0 & 2/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \end{array} = P \tag{4.18}$$

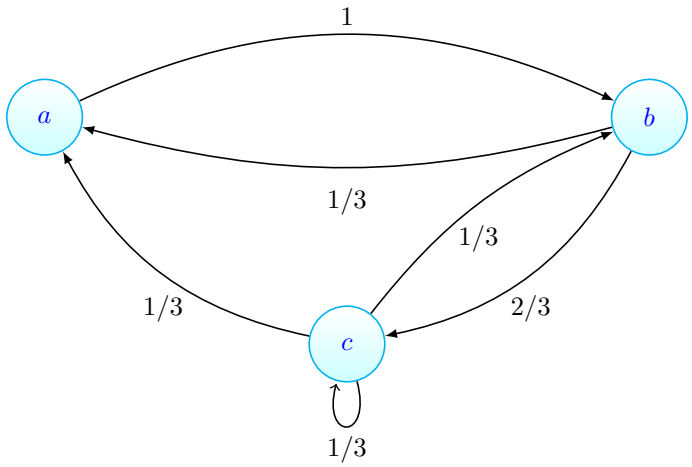


Abbildung 4.2: Beispiel Markov-Kette mit diskreter Zeit

Der Begriff der Markov-Kette wird dabei gleichbedeutend mit zeitdiskreten Markov-Ketten (DTMC) verwendet. Dabei ist zu beachten, dass die Wahrscheinlichkeit exakt für einen Zeitschritt angegeben ist. Für die DTMC sind darüber hinaus Erweiterungen, die Zeitschritte variabler Weite berücksichtigen,

verfügbar. Es gibt aber auch die Möglichkeit, die Wahrscheinlichkeit mehrerer Zeitschritte (n-steps) [64] zu berechnen. Dies erfordert eine entsprechende Anpassung der Wahrscheinlichkeitsmatrix (P). Des Weiteren sind Ansätze, die stetige Zeiteigenschaften abdecken, wie die Markov-Kette mit kontinuierlicher Zeit (CTMC), etabliert. Markov-Ketten mit kontinuierlicher Zeit unterscheiden sich nur unwesentlich von DTMC, die mehr als einen Zeitschritt [64] darstellen. Eine DTMC verharrt mindestens eine oder eine endliche Anzahl an Zeiteinheiten in jedem Zustand, eine CTMC dagegen verbringt eine zufällige Menge an Zeit in jedem Zustand, wobei angenommen wird, dass die Zeitdauer einer Exponentialverteilung genügt. Solange die Exponentialverteilung gedächnislos ist, spielt es keine Rolle, wie lange das System bereits in diesem Zustand verharrt. Die verbleibende Dauer ist immer noch exponentiell verteilt. Dies impliziert, dass ein Zustandswechsel in einer CTMC zu jedem beliebigen Zeitpunkt auftreten kann im Gegensatz zur DTMC, wo dies nur zu festen Zeitpunkten $\{n = 0, 1, 2, \ldots, n\}$ möglich ist. Die konzeptionelle Ähnlichkeit zwischen DTMC und CTMC ermöglicht auch eine Überführung (Konvertierung) der CTMC in eine DTMC. Dies kann mit Hilfe der Uniformisation [63] [6] erfolgen.
Die formale Definition der CTMC (M) wird [6] als Quadrupel mit den Bezeichnungen $(S; \bar{s}; R; L)$ wie folgt dargestellt:

- S ist eine endliche Menge von Zuständen,
- $\bar{s}$ ist der initiale Zustand, wobei $\bar{s} \subset S$ ist.
- $R : S \times S \to R_{\geq 0}$ ist die Ratenmatrix. Dabei ist $R(s, s')$ die Rate für den Wechsel vom Zustand s nach s_0, d.h. die Wahrscheinlichkeit des Übergangs von s nach s_0 in t ($\in R^{\geq 0}$) Zeiteinheiten beträgt $P(s, s', t) = 1 - e^{-R(s,s')t}$.
- $L : S \to 2^{AP}$ ist die Kennzeichnungsfunktion, welche jedem Zustand $s \in S$ die Menge $L(s)$ von atomaren Aussagen $a \in AP$ zuweist, die in s gültig sind.

M ist endlich, wenn S und AP endlich sind. Für ein endliches M wird die Größe von M mit $size(M)$ angegeben. Darunter versteht man die Anzahl der Zustände plus die Anzahl der Paare $(s, s') \in S \times S$ mit $R(s, s') > 0$.
Die Wahrscheinlichkeit $\mathcal{P}(s, s', t)$ für den Übergang von s nach s' in t Zeiteinheiten wird durch die Wahrscheinlichkeitsübergangsfunktion $\mathcal{P}(s, s', t) = \frac{R(s,s')}{E(s)}(1 - e^{-R(s,s')t})$ angegeben, wobei $E(s) = \sum_{s' \in S} R(s, s')$ die Gesamtrate angibt, bei welcher ein Übergang, ausgehend von dem Zustand s angenommen wird. Somit wird die Wahrscheinlichkeit für einen Wechsel vom Zustand s nach s' durch einen einzigen Übergang als $P(s, s') = R(s, s')/E(s)$ angegeben. Für Zustände, bei denen $R(s, s') = 0$ ist, gilt $E(s) = 0$ und somit ist $P(s, s') = 0$. Die Matrixrepräsentation von P wird als die Transitionsmatrix der eingebetteten DTMC von M angesehen.
Die Pfade in der Markov-Kette sind maximale (z.B. unendliche) Pfade des unterliegenden Graphen. Sie werden als unendliche Zustandssequenzen $\pi = s_0, s_1, s_2 \cdots \in S^\omega$ definiert, so dass $P(s_i, s_{i+1}) > 0$ für alle $i \geq 0$ gilt.

Die initiale Wahrscheinlichkeitsverteilung der Zustände wird mit $\iota_{init}(s)$ angegeben. Die Zustände mit $\iota_{init}(s) > 0$ heißen auch initiale Zustände. Sofern nur ein initialer Zustand existiert, ist die Verteilung $\iota_{init}(s) = 1$ und $\iota_{init}(s') = 0$ für alle $s' \neq s$.
Markov-Ketten werden häufig als zeitdiskrete Markov-Ketten bezeichnet. Dabei ist die DTMC zeitlich abstrahiert, wogegen die CTMC die Zeiteigenschaften wahrt, da sie einen expliziten Bezug zur Zeit hat. Die stochastische Entwicklung des durch die DTMC bzw. CTMC beschriebenen Systems über die Zeit wird durch die Transitionsswahrscheinlichkeiten bzw. Raten festgelegt. Bei einer CTMC M wird das zeitlich abstrahierte Verhalten durch ihre eingebettete DTMC ausgedrückt.
Der zugrundeliegende Graph bildet die Basis für Schlussfolgerungen über Eigenschaften und lässt Analysen hinsichtlich quantitativer und qualitativer Merkmale zu. Hierbei können die qualitativen Eigenschaften als spezieller Fall der quantitativen Eigenschaften angesehen werden. Details zu qualitativen und quantitativen Eigenschaften können Kapitel 4.5.1 entnommen werden.

4.4.2 Darstellung als Raten-Matrix

Die Markov-Kette mit kontinuierlicher Zeit wird mittels Transitionsraten (Intensitäten) dargestellt. Für eine mathematische Repräsentation wird dazu häufig die Raten-Matrix [74] genutzt. Eine gebräuchliche Bezeichnung für die Ratenmatrix ist der Begriff „infinitesimale Generatormatrix“ [77] bzw. Intensitätsmatrix. Die Raten werden als die Anzahl der Transitionen pro Zeiteinheit definiert. Für das Gesamtsystem kann mit Hilfe einer Amalgamation der Intensitätsmatrizen eine vollständige Raten-Matrix gebildet werden. Die Matrix gibt mittels der Elemente q_{ij} die Rate für das Verlassen vom Zustand i und das Eintreten in den Zustands j an. Die Diagonalelemente q_{ii} haben jedoch eine Sonderfunktion und werden immer negativ dargestellt. Sie sind nach Gleichung 4.19 definiert. Daraus resultiert eine wesentliche Eigenschaft der Matrix: Entsprechend Gleichung 4.20 ergibt die Summe jeder Reihe 0.

$$q_{ii} = -\sum_{i \neq j} q_{ij} \tag{4.19}$$

$$\sum q_{ij} = 0 \tag{4.20}$$

Wie bereits oben erwähnt enthält jede CTMC [63] eine eingebettete DTMC, die die entsprechenden Transitionswahrscheinlichkeiten (siehe Gleichung 4.22), abgeleitet aus den Raten (Gl. 4.21), angibt. Die Summe dieser Wahrscheinlichkeiten muss der Gleichung 4.23 entsprechen. Es ist zu beachten, dass die eingebettete Markov-Kette per definitionem keine Schleifen (Eigenübergänge)

unterstützt, $P_{ii} = 0$.

$$v_i = \sum_{i \neq j} q_{ij} = -q_{ii} \tag{4.21}$$

$$P_{ij} = q_{ij}/v_i \tag{4.22}$$

$$\sum_{i \neq j} P_{ij} = 1 \tag{4.23}$$

Entsprechend der Gleichungen 4.21 und 4.22 kann Q als die Ableitung von P nach der Zeit interpretiert werden. Bei dieser Transformation fällt auch die Einschränkung einer Zulassung von Eigenübergängen weg. Es resultiert eine vollwertige DTMC.

4.5 Probabilistisches Model Checking von Markov-Ketten mit kontinuierlicher Zeit

4.5.1 Grundlagen

In diesem Unterkapitel [1] wird das Model-Checking, speziell das probabilistische Model-Checking [7], näher erläutert. Diese Methode gehört zu den formalen Verifikationstechniken für die automatisierte Analyse von Systemen, die stochastische Komponenten enthalten. In der Praxis werden solche probabilistischen Systeme häufig mittels Markov-Ketten modelliert. Im weiteren Verlauf soll speziell auf das Model-Checking von Markov-Ketten mit kontinuierlichen Zeiteigenschaften [6] näher eingegangen werden.
Durch Model-Checking wird überprüft, ob ein System, das durch ein Modell, z.B. eine Markov-Kette, repräsentiert wird, eine formale Spezifikation erfüllt. Dies geschieht durch Untersuchen der möglichen Zustände, indem alle oder zumindest viele der möglichen Kombinationen (Brute-Force) ausprobiert werden. Üblicherweise wird dies nicht manuell, sondern mit Hilfe von Software-Werkzeugen durchgeführt.
Die Analyse kann entweder zustandsbasiert oder pfadbasiert erfolgen. Die zustandsbasierte Überprüfung stellt u.a. fest, ob „entlang“ (im Verlauf) eines Pfades zuvor spezifizierte Zustände erreicht werden können. Des Weiteren besteht die Möglichkeit zu überprüfen, ob vorgegebene Zustände eventuell erreichbar sind, wenn zuvor ausschließlich definierte Zustände durchlaufen wurden. Damit sind Analysen bezüglich des stationären Zustandes (Langzeitverhalten) oder die Bestimmung von transienten Zustandswahrscheinlichkeiten möglich. Bei der pfadbasierten Überprüfung lassen sich hingegen Sequenzen (Zustandsfolgen) durch das Transitionssystem, ausgehend von einem initialen Zustand bis hin zu einem definierten Endzustand, ermitteln. Es ist auch eine Spezifikation von Wahrscheinlichkeiten, die für einen Pfad gelten und bei der Überprüfung berücksichtigt werden, möglich. Ebenso kann ein bestimmter Startzustand festgelegt werden, von dem aus innerhalb einer gewissen Anzahl von Zeiteinheiten

[1] Teile dieses Unterkapitels wurden aus [7] entnommen.

definierte Zielzustände erreicht werden sollen.
Die Verifikation bzw. Analyse der modellierten Systeme kann sowohl quantitative als auch qualitative Eigenschaften umfassen. Qualitative Eigenschaften treten nach [7] als ein spezieller Fall von quantitativen Eigenschaften auf. Eine typische qualitative Eigenschaft fordert, dass die Wahrscheinlichkeit des Erreichens eines schlechten Zustandes Null ist bzw. dass ein bestimmtes gewünschtes Systemverhalten mit der Wahrscheinlichkeit Eins auftritt. Quantitative Eigenschaften nehmen üblicherweise Einschränkungen vor entweder hinsichtlich der Wahrscheinlichkeit oder hinsichtlich der Erwartung (Zeit, Anzahl, usw.) bestimmter Ereignisse.
Beispiele für quantitative Eigenschaften sind zum einen die Anforderung, dass das Auftreten eines Zustandes innerhalb der nächsten t Zeiteinheiten mindestens eine definierte Wahrscheinlichkeit beträgt oder zum anderen, dass die zu erwartende Anzahl der Fehlversuche höchstens eine festgelegte Anzahl ausmacht. Durch die Überprüfung qualitativer Eigenschaften von Markov-Modellen lässt sich die Erreichbarkeit ihrer Zustände ermitteln.
Häufige Analyseszenarien sind zum einen die Persistenz, d.h. ob ein Event jemals auftritt, und zum anderen die wiederholte Erreichbarkeit, d.h. ob bestimmte Zustände wiederholt erreicht werden können. Das Ziel ist es, qualitative und quantitative Eigenschaften zu spezifizieren und zu überprüfen.
Beim probabilistischen Model-Checking werden die gewünschten Eigenschaften, gegen die geprüft wird, durch eine formale, aus temporaler Logik bestehende Spezifikation repräsentiert. Gebräuchliche, von etablierten Model-Checker Software-Anwendungen unterstützte Logiken für die Analyse von CTMC sind Continuous Stochastic Logic (CSL) bzw. Probabilistic Computation Tree Logic (PCTL).

4.5.2 Software-Werkzeuge

Die eigentliche Modellprüfung wird, wie in Kapitel 4.5.1 bereits erwähnt, mit Hilfe von Software-Werkzeugen durchgeführt. Diese Computerprogramme müssen speziell die probabilistischen Eigenschaften, wie beispielsweise die wahrscheinlichkeitstheoretische Erreichbarkeit, berücksichtigen. Die wohl bekannteste und am weitesten verbreitete Software für diese Aufgabe ist PRISM [78] [76]. Im Rahmen der Modellprüfung kann dieses Werkzeug auch die Markov-Kette symmetrisch reduzieren [79] und somit besonders bei großen Markov-Ketten eine Beschleunigung der Berechnung erreichen. Darüber hinaus kann speziell bei Markov-Ketten mit kontinuierlichen Zeiteigenschaften (CTMC) eine Uniformisation [51] zur Laufzeit durchgeführt werden. Dies konvertiert die CTMC in eine Markov-Kette mit diskreten Zeiteigenschaften (DTMC) und ermöglicht damit [7] eine Transienten-Analyse. In diesem Rahmen wird die Wahrscheinlichkeit berechnet, die angibt, ob sich das System in einem bestimmten (definierten) Zustand zu einem spezifischen Zeitpunkt (transient) befindet.
Leider ist PRISM relativ langsam [55] und daher für größere Probleme schlecht geeignet. Dies wurde auch bei der Durchführung der vorliegenden Arbeit bei initialen Experimenten festgestellt. Für die weitere Auswertung wurde deshalb der

Markov Rewarded Model Checker (MRMC) [65] genutzt. Allerdings war zur Zeit der Durchführung dieser Arbeit die frei verfügbare MRMC-Version derart eingeschränkt, so dass nur Modelle mittlerer Größe gehandhabt werden konnten. Es zeigte sich aber, dass diese Restriktion ausschließlich auf die Repräsentation der in MRMC intern für Speicherung, Indizierung und Zugriff verwendeten Datentypen zurückzuführen war und nicht auf den implementierten Algorithmus. Die MRMC-Software wurde daher zur Verarbeitung von großen Zustandsräumen, bei denen viele Transitionen existieren, angepasst.
Eine solche erweiterte Version von MRMC machte eine Analyse der quantitativen und qualitativen Eigenschaften von Markov-Ketten mit kontinuierlichen Zeiteigenschaften durchführbar. Hierbei ist zu beachten, dass die gewünschten Eigenschaften nur in der Spezifikationssprache, der Continuous Stochastic Logic [5] (CSL), formalisiert werden können. Dies deckt jedoch sowohl Zustands- als auch Pfad-Formeln ab und gestattet daher sowohl die Auswertung von zeitlosen Eigenschaften als auch Langzeit-Analysen (Steady-State) und Eigenschaften, die nur zu einem bestimmten Zeitpunk gelten (transient/zeitbegrenzt).
Für alle CTMC-Modellanalysen im Rahmen dieser Arbeit wurde die angepasste Version des MRMC verwendet.

4.6 Nichtlineares Kalman Filter

Das Kalman-Filter [152] ist ein statistischer Zustandsschätzer für dynamische Systeme mit zeitlich veränderlichem Zustand, denen ein dynamisches Modell zugrunde liegt. Auf der Basis dieses mathematischen Modells ermöglicht es das Kalman-Filter, nicht direkt messbare bzw. beobachtbare Zustandsgrößen aus gemessenen Ein- und Ausgangsgrößen zu bestimmen. Das damit beschriebene Systemverhalten wird als Zufallsvariable x aufgefasst. Das Standard Kalman-Filter umfasst ausschließlich zeitdiskrete Modelle, d.h. Anwendungen, bei denen die Zustände zu diskreten, jedoch nicht notwendigerweise äquidistanten Zeitpunkten $t_0, t_1, t_2, \ldots$ betrachtet werden. Die Zustandswerte, die den einzelnen Zuständen zugeordnet werden, sind mit $x_k = X_t$, die Beobachtungswerte mit y_k bezeichnet. Die Unsicherheiten bzw. Ungenauigkeiten und Störungen im Prozess lassen sich mittels Messrauschen (v_k) und Prozessrauschen (w_k) abbilden. Dabei wird angenommen, dass Prozessrauschen v_k und Messrauschen w_k jeweils voneinander unabhängig sind. Ein einfaches Beispiel einer Kalman-Filter-Zustandssequenz ist in Abbildung 4.3 dargestellt. Auf Grund der Einwirkung von (v_k) und (w_k) kann die zeitliche Entwicklung des Systemzustandes, selbst bei gegebenem Anfangszustand, als nicht gesichert angenommen werden.
Für die Schätzung des Zustandes werden Zustandsraummodelle der nachfolgenden Form verwendet, wobei f die Transfer- bzw. Übergangsfunktion für den Zustandswechsel ist und h die Messfunktion (Überwachungsfunktion), die aus dem errechneten Zustand den Messwert angibt, bezeichnet.

$$x_{k+1} = f(x_k, v_k) \tag{4.24}$$

$$y_k = h(x_k, w_k) \tag{4.25}$$

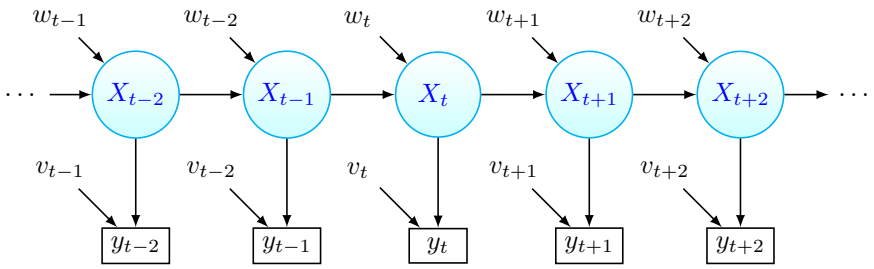


Abbildung 4.3: Kalman-Zustandssequenz

Hierbei wird das Rauschen (v_k, w_k) als additive Komponente angesehen.

$$x_{k+1} = f(x_k) + v_k \tag{4.26}$$

$$y_k = h(x_k) + w_k \tag{4.27}$$

Die Zustände x_k können mehrdimensional sein, sie werden dann in Form eines Vektors dargestellt. Die Menge aller über Gleichung 4.26 erreichbaren Zustände x_k bildet einen speziellen stochastischen Prozess ab, nämlich eine Markov-Kette bzw. ein Markov-Modell. Der Systemzustand, und somit der geschätzte Zustand, kann üblicherweise nicht direkt überwacht werden und hängt ausschließlich vom Vorgängerzustand und ggf. von Kontrolleingaben ab. Die vorgenannten Besonderheiten sind wesentliche Merkmale des Hidden-Markov-Modells [110], wie es auch hier vorliegt. Modelle, wie sie in Gleichung 4.24 dargestellt werden, bringen die Einschränkung mit, ausschließlich lineare Zusammenhänge, z.B. der Daten oder des Systems, zu repräsentieren. Diese Einschränkung wird durch die Markov-Parameter, die den Zustandsraum beschreiben bzw. den Prozess repräsentieren, vorgegeben.
Ein anderer weit verbreiteter Ansatz für die Schätzung von Nichtlinearitäten, der nachfolgend näher beschrieben wird, ist das Erweiterte-Kalman-Filter (EKF) [56] [57]. Wie auch beim normalen Kalmanfilter wird hier die Systemdynamik durch zeitlich veränderbare Matrizen (A,B,C,D) in Form der Zustandsraumdarstellung dargestellt, s. Gleichung 4.39. Diese Darstellung legt nahe, dass die reale Systemdynamik (f und h) für eine weitere Anwendung linearisiert werden muss und näherungsweise in linearisierter Form abgebildet wird. Die Schätzung des nichtlinearen Systemverhaltens wird durch eine Taylor-Reihen-Entwicklung, die üblicherweise nach dem ersten Glied abgebrochen wird, nur approximiert. Ein solches Vorgehen lässt den Schluss zu, dass korrekte Ergebnisse nur im linearisierten Bereich erzielt werden können.
Eine Verbesserung, die Einschränkungen durch Linearisierung und Approximation vermeidet, ist durch das Unscented Kalman Filter (UKF) [150] [60] [58], häufig auch Sigma-Point Kalman Filter [145] genannt, gegeben.
Im Gegensatz zum EKF nutzt das UKF eine nicht-lineare Zustandsraum-Darstellung und verwendet eine spezielle Transformation, die „unscented Transformation“ [59], für die Berechnung der Abschätzung von nicht-linearen Filterproblemen. Diese Transformation [59] berechnet nicht wie das EKF direkt aus

dem aktuellen Zustand den Folgezustand, sondern benötigt einen deterministischen Zwischenschritt. Dieser leitet aus dem Mittelwert und der Kovarianz des vorherigen Zustandes Sigma-Punkte her. Die Wahrscheinlichkeitsverteilung einer Zufallsvariablen wird auf der Basis der Sigmapunkte mit Hilfe einer nichtlinearen Funktion approximiert. Die statistischen Eigenschaften von Zustands- und Messwertschätzung werden auf der Basis der transformierten Sigma-Punkte berechnet. Daraus ergibt sich gegenüber dem EKF eine exaktere stochastische Näherung zur Verarbeitung nicht-linearer Funktionen.
Ein Überblick über den UKF-Algorithmus wird im Folgenden gegeben, wobei die Besonderheiten im Vergleich zum linearen Kalman Filter detailliert erläutert werden. Der Pseudo-Code des UKF-Algorithmus ist in Algorithmus 1 dargestellt.
Der Algorithmus [151] benötigt für die Verarbeitung einer Zufallsvariablen x (der Dimension n) nach Gleichung 4.26 den Mittelwert μ und die Kovarianz Σ dieser Zufallsvariablen sowie die Steuereingabe u als Eingabe; diese Größen müssen zum Zeitpunkt $k-1$ korrespondieren. Des Weiteren werden die Messwertbeobachtung z, das Messrauschen R_k und das Prozessrauschen Q_k, jeweils zum Zeitpunkt k benötigt.
Den Kern des UKF bildet die sog. „unscented-Transformation“. Dabei dienen die Sigma-Punkte $\mathcal{X}$ (eq. 4.28), die sich aus dem Systemzustand ergeben und auf dem Mittelwert μ und der Kovarianz Σ basieren, als Eingabewert der Transferfunktion f. Im allgemeinen Fall sind diese Sigma-Punkte beim Mittelwert μ und symmetrisch entlang der Hauptachsen der Kovarianz Σ platziert. Für die Wahl der 2n + 1 Sigma-Punkte kommt die Regel aus Gleichung 4.28 zum Einsatz:

$$\begin{aligned} \mathcal{X}^{[0]} &= \mu \\ \mathcal{X}^{[i]} &= \mu + (\sqrt{(n+\lambda)\Sigma})_i \qquad \text{for } i = 1, \ldots, n \\ \mathcal{X}^{[i]} &= \mu - (\sqrt{(n+\lambda)\Sigma})_{i-n} \quad \text{for } i = n+1, \ldots, 2n \end{aligned} \tag{4.28}$$

Dabei ist $(\sqrt{(n+\lambda)\Sigma})_i$ die i-te Spalte der Matrix-Quadratwurzel. λ ist ein Skalierungsparameter, der die Distanz der Sigmapunkte um den Mittelwert festlegt. Entsprechend der Anzahl der Sigma-Punkte wird eine Menge von passend gewählten Gewichten $w_m^{[i]}$ und $w_c^{[i]}$ mit Hilfe der folgenden Gleichung (eq. 4.29) [59] berechnet. Hierzu müssen die Parameter α und β für die Berechnung der Gewichte $w_m^{[i]}$ und $w_c^{[i]}$ gewählt werden. α kontrolliert dabei die „Größe der Sigma-Punkt-Verteilung und β ist ein Gewichtungsfaktor, der den Einfluss des nullten Sigma-Punktes steuert. Weitere Details zur Auswahl der Parameter λ , α und β können [145] entnommen werden.

$$\begin{aligned} w_m^{[0]} &= \frac{\lambda}{n+\lambda} \\ w_c^{[0]} &= \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta), \quad 0 < \alpha < 1, \quad \beta > 0 \\ w_m^{[i]} &= w_c^{[i]} = \frac{1}{2(n+\lambda)}, \quad \text{i} = 1, \ldots, 2n \end{aligned} \tag{4.29}$$

Die Schätzungen des Mittelwertes $\hat{\mu_k}$ (Gl. 4.31) und der Kovarianz $\hat{\Sigma_k}$ (Gl. 4.32) basieren auf den transformierten Sigma-Punkten $\bar{\mathcal{Y}}_k^{[\mathbf{i}]}$ (siehe Gl. 4.30). Die Transformation der Sigma-Punkte wird in Abbildung 4.4 grafisch veranschaulicht. Eine Aktualisierung der Sigma-Punkte erfolgt, wie in Gleichung 4.33 dargestellt, auf der Basis der neuberechneten Mittelwerte und der aktualisierten Kovarianzen. $\hat{\mathcal{X}}_k^{[i]}$ weist die identische Struktur wie $\mathcal{X}^{[i]}$ aus Gleichung 4.28 auf.

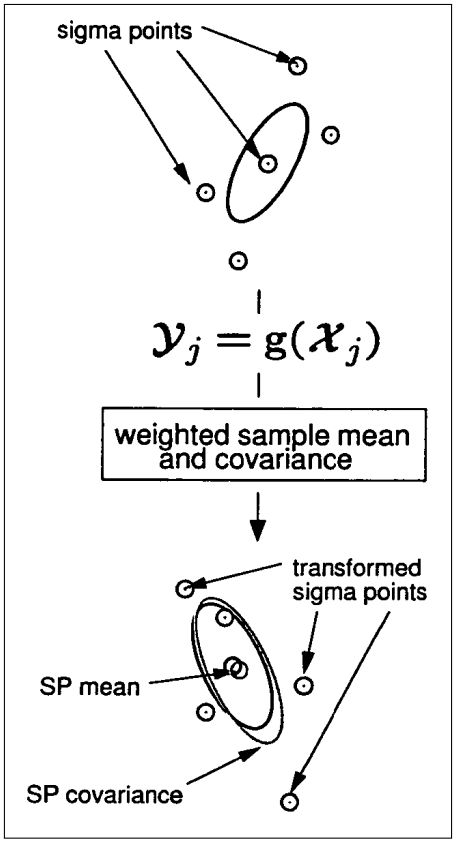


Abbildung 4.4: Sigma-Punkt Transformation[2]

$$\bar{\mathcal{Y}}_k^{[\mathbf{i}]} = g_\mu(\mathbf{u}_{k-1},\ \mathcal{X}_{k-1}^{[i]}) \tag{4.30}$$

$$\hat{\mu}_k = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Y}}_k^{[\mathbf{i}]} \tag{4.31}$$

[2]Die Grafik in Abbildung 4.4 wurde aus [150] entnommen.

$$\hat{\Sigma}_k = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Y}}_k^{[\mathbf{i}]} - \hat{\mu}_k)(\bar{\mathcal{Y}}_k^{[\mathbf{i}]} - \hat{\mu}_k)^T + Q_k \tag{4.32}$$

$$\hat{\mathcal{X}}_k^{[i]} = (\hat{\mu}_k \ , \ \hat{\mu}_k + \sqrt{(n+\lambda)\hat{\Sigma}_k} \ , \ldots, \ \hat{\mu}_k - \sqrt{(n+\lambda)\hat{\Sigma}_k} \ , \ldots) \tag{4.33}$$

Das Kalman-Update erfolgt auf der Basis der aktualisierten transformierten Sigma-Punkte (siehe Gleichung 4.34), die als Eingabe der Überwachungsfunktion h dienen. Die so entstandenen Abschätzungen der Messwerte-Näherungen werden, wie in Gleichung 4.35 dargestellt, gewichtet fusioniert, so dass ein Mittelwert der Messgrößenabschätzung resultiert.

$$\hat{\mathcal{Z}}_k^{[\mathbf{i}]} = h_\mu(\hat{\mathcal{X}}_k^{[\mathbf{i}]}) \tag{4.34}$$

$$\hat{z}_k = \sum_{i=0}^{2n} w_m^{[i]} \hat{\mathcal{Z}}_k^{[\mathbf{i}]} \tag{4.35}$$

Auf der Basis dieses Mittelwertes können die Residuen, Abweichungen vom gemessenen zum geschätzten Messwert, berechnet werden. Diese fließen direkt in die Berechnung der Kovarianz der Überwachung (Gl. 4.36) ein.

$$S_k = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Z}}_k^{[\mathbf{i}]} - \hat{z}_k)(\bar{\mathcal{Z}}_k^{[\mathbf{i}]} - \hat{z}_k)^T + R_k \tag{4.36}$$

Im Anschluß wird die Cross-Kovarianz nach Gleichung 4.37 berechnet. Diese wird für die Berechnung des Kalman-Ergebnisses (Kalman-Gain) benötigt, das die Bedeutung des Fehlers in Bezug auf die vorherige Schätzung angibt:

$$\hat{\Sigma}_k^{\mu,z} = \sum_{i=0}^{2n} w_c^{[i]} (\hat{\mathcal{X}}_k^{[\mathbf{i}]} - \hat{\mu}_k)(\bar{\mathcal{Z}}_k^{[\mathbf{i}]} - \hat{z}_k)^T \tag{4.37}$$

Für den Betrieb des UKF-Algorithmus wird u.a. die Kenntnis des Prozessrauschens Q vorausgesetzt. Bei einigen Anwendungen ist dies jedoch unbekannt bzw. kann sich zur Laufzeit ändern. Eine einfache Möglichkeit, damit umzugehen besteht darin, das Prozessrauschen Q mit Hilfe der Prädiktionsabweichung vom Messwert und der Kovarianz zu ermitteln. Dieser Ansatz wurde in [88] [129] erwähnt und ist in Gleichung 4.38 dargestellt:

$$Q_{k+1} = (1 - d_k)\, Q_k + d_k \left[K_k \left(z_k - \hat{z}_k \right) \cdot \left(z_k - \hat{z}_k \right)^T K_k^T + \Sigma_k - \hat{\Sigma}_{k-1} \right] \tag{4.38}$$

Durch Einfügen der Gleichung 4.38 als letzte Operation im UKF-Algorithmus ist eine Erweiterung zum adaptiven UKF (AUKF) vollzogen. Die Gleichungen (4.28 ... 4.37) werden nachfolgend zusammengefasst in „Algorithmus 1“ als Pseudo-Code des UKF-Algorithmus dargestellt. Die Transferfunktion f, Überwachungsfunktion h und die Dimension n der Zufallsvariablen werden als bekannt vorausgesetzt.

Algorithmus 1 Unscented Kalman Filter

Eingabe: $\mu_{k-1}; \Sigma_{k-1}; u_{k-1}; z_k; Q_k; R_k$:

$\mathcal{X}^n_{k-1} = (\mu_{k-1} \ , \ \mu_{k-1} + \sqrt{(n+\lambda)\Sigma_{k-1}} \ , \ldots, \ \mu_{k-1} - \sqrt{(n+\lambda)\Sigma_{k-1}} \ , \ldots)$

$w^0_m = \frac{\lambda}{n+\lambda}$

$w^0_c = \frac{\lambda}{n+\lambda} + (1 - a^2 + \beta)$

$w^n_m = w^n_c = \frac{1}{2(n+\lambda)}$

for $i = 0 \ldots 2n$: **do**

  $\bar{\mathcal{Y}}^{[\mathbf{i}]}_k = g_\mu(\mathbf{u}_{k-1}, \ \mathcal{X}^{[i]}_{k-1})$

  $\hat{\mu}_k = \sum_{i=0}^{2n} w^{[i]}_m \bar{\mathcal{Y}}^{[\mathbf{i}]}_k$

  $\hat{\Sigma}_k = \sum_{i=0}^{2n} w^{[i]}_c (\bar{\mathcal{Y}}^{[\mathbf{i}]}_k - \hat{\mu}_k)(\bar{\mathcal{Y}}^{[\mathbf{i}]}_k - \hat{\mu}_k)^T + Q_k$

  $\hat{\mathcal{X}}^{[i]}_k = (\hat{\mu}_k \ , \ \hat{\mu}_k + \sqrt{(n+\lambda)\hat{\Sigma}_k} \ , \ldots, \ \hat{\mu}_k - \sqrt{(n+\lambda)\hat{\Sigma}_k} \ , \ldots)$

end for

for $i = 0 \ldots 2n$: **do**

  $\hat{\mathcal{Z}}^{[\mathbf{i}]}_k = h_\mu(\hat{\mathcal{X}}^{[\mathbf{i}]}_k)$

  $\hat{z}_k = \sum_{i=0}^{2n} w^{[i]}_m \hat{\mathcal{Z}}^{[\mathbf{i}]}_k$

  $S_k = \sum_{i=0}^{2n} w^{[i]}_c (\bar{\mathcal{Z}}^{[\mathbf{i}]}_k - \hat{z}_k)(\bar{\mathcal{Z}}^{[\mathbf{i}]}_k - \hat{z}_k)^T + R_k$

  $\hat{\Sigma}^{\mu,z}_k = \sum_{i=0}^{2n} w^{[i]}_c (\hat{\mathcal{X}}^{[\mathbf{i}]}_k - \hat{\mu}_k)(\bar{\mathcal{Z}}^{[\mathbf{i}]}_k - \hat{z}_k)^T$

end for

$K_k = \hat{\Sigma}^{\mu,z}_k S^{-1}_k$

$\mu_k = \hat{\mu}_k + K_k(z_k - \hat{z}_k)$

$\Sigma_k = \hat{\Sigma}_k - K_k S_k K^T_k$

Ausgabe: $\mu_k, \Sigma_k, \hat{z}_k$

4.7 System-Identifikation mittels Unterraumidentifikation

In vielen praktischen Anwendungen ist das Systemverhalten nicht ausreichend bekannt, um ein mathematisches Modell, z.B. Differentialgleichung, aufzustellen. Deshalb kann in der Praxis keine Spezifikation bzw. Definition der Transfer- und der Überwachungsfunktion des Kalmanfilters erfolgen. Alternativ können diese Funktionen aber aus aufgezeichneten Messwerten abgeleitet werden, indem durch die Messwerte auf die Zustandssequenz des Systems geschlossen wird. Dies wurde für verschiedene lineare Systeme bereits umgesetzt. Da ein zu identifizierendes System über beobachtbare Eingangswerte (u_k) und messbare Ausgaben (y_k) verfügt, ist noch eine andere gebräuchliche Methode möglich, die durch Sub-Space-Algorithmen (engl. für Unterraum-basierte Algorithmen) abgedeckt

wird. Diese verwenden eine Zustandsraumdarstellung (Gl. 4.39) als Grundlage. Bei Zustandsraum-Modellen wird die System-Dynamik mit Hilfe der Markov-Parameter üblicherweise als Quadrupel von zeitlich veränderbaren Matrizen (A,B,C,D) dargestellt. In Gl. 4.39 wird diese durch ein Sextupel (A,B,C,D,E,F), das die Systemmatrizen repräsentiert, angegeben. Eine solche Darstellung wird in der System-Theorie auch als Realisierung bezeichnet.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + Ev_k \\ y_k &= Cx_k + Du_k + Fw_k \end{aligned} \tag{4.39}$$

In Gleichung 4.39 gibt $k \geq 0$ die diskrete Zeit, $x_k \in R^n$ den Zustandsvektor, $u_k \in R^r$ den Eingabevektor, $y_k \in R^m$ den Ausgabevektor und $v_k \in R^l$ sowie $w_k \in R^l$, externe Eingabevektoren für z.B. Prozessrauschen oder Messwertstörungen an.
Ein Algorithmus, mit welchem aus Messwerten Zustandsraum-Modelle berechnet werden können, wurde in [113] vorgestellt. Dabei handelt es sich um einen kombinierten „deterministischen und stochastischen System- und Realisierungs“-Algorithmus (DSR) [3] für lineare zeitinvariante diskrete Systeme.
Eine wesentliche Rolle innerhalb des DSR-Algorithmus spielt die Hankel-Matrix (Gleichung 4.40), welche häufig bei der Realisierung in der Systemtheorie und Unterraumidentifikation (Sub-Space-Identifikation) verwendet wird. Die spezielle Struktur der Hankel-Matrix, die symmetrisch ist und auf der Anti-Diagonalen nur konstante Werte [114] aufweist, wie auch wesentliche Notationen der Systemidentifikation werden im Folgenden erläutert: Die Hankel-Matrix wird gewöhnlich aus Zeitreihendaten erzeugt. Dabei wird die Anzahl der Zeilen mit „nr“ und die Anzahl der Spalten mit „nc“ angegeben. Auf Basis dieser Daten werden sog. erweiterte Daten-Matrizen generiert. Exemplarisch ist dies für die Ausgabewerte, wie in 4.40 dargestellt.

$$\begin{aligned} Y_{k|L} \equiv \begin{bmatrix} y_k & y_{k+1} & y_{k+2} & \cdots & y_{k+K-1} \\ y_{k+1} & y_{k+2} & y_{k+3} & \cdots & y_{k+K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{k+L-1} & y_{k+L} & y_{k+L+1} & \cdots & y_{k+L+K-2} \end{bmatrix} \\ \equiv \left[\frac{Y_p}{Y_f}\right] \in R^{Lm \times K} \end{aligned} \tag{4.40}$$

In dieser erweiterten Ausgabedaten-Matrix (4.40) gibt $k \hat{=} t_0$, den Startindex bzw. die initiale Zeit in der Sequenz an, die aus aufgezeichneten Messwerten y_k besteht. Der Startindex ist im oberen linken Block der Hankel-Matrix platziert. Des Weiteren gibt L die Anzahl von nr-Blockreihen in $Y_{k|L}$ und K die Anzahl von nc-Blockspalten in $Y_{k|L}$ an, d.h. die Anzahl der Messwerte in der erweiterten Zeitserie [113].
Diese Art der Darstellung ermöglicht eine Auswertung der Blockreihen und der

[3] Teile dieses Unterkapitels wurden aus [113] entnommen.

Blockspalten als Vergangenheits- versus Zukunfts-Schema. Die benötigten, jedoch nicht vorhandenen Zukunftsmesswerte werden erzeugt, indem die real existierenden Messwerte in zwei Gruppen unterteilt werden. Diese Gruppen werden als Y_p- und Y_f-Messungen angesehen. Dies impliziert, dass Y_f als Pseudo-Zukunftsmessung generiert wird. Basierend auf dieser Darstellung (Y_p/Y_f) kann die generelle Entwicklungsrichtung (Trend), z.B. des Systemverhaltens, auf der Basis der Daten mittels QR-SVD-Zerlegung berechnet werden. Dabei berechnet der QR-Dekompositionsteil [113] die Projektion der Vergangenheitswerte auf die Zukunftswerte, d.h. von Y_p nach Y_f. Die QR-Dekomposition kann als eine Datenkompression angesehen werden. Dabei wird gewöhnlich die Datenmatrix $Y_{k|L}$, die üblicherweise eine große Anzahl an Spalten hat, in eine viel kleinere untere Dreiecksmatrix, die alle relevanten Informationen beinhaltet, überführt. Auf das so erzielte Ergebnis wird eine Singulärwertzerlegung (SVD) angewandt, durch die neben der Systemordnung auch die Eigenvektoren / Eigenwerte errechnet werden können. Geometrisch können diese als Richtungskosinus (Winkel), der die Orthogonal-„Drehung“ angibt, aufgefasst werden. Dies entspricht den Hauptwinkeln (principal angles) zwischen Matrizen, die Unterräume aufspannen.
Aus dem Zustandsraum-Modell 4.39 lässt sich unter Zuhilfenahme der erweiterten Daten-Matrizen $(Y_{k|L}, U_{k|L+1}, E_{k|L+1})$ nun das erweitere Zustandsraum-Modell (ESSM) 4.41 herleiten. Dabei werden die Zustandsraum-Modell-Matrizen durch $\tilde{A}$, $\tilde{B}$, $\tilde{E}$ angegeben. Hierbei ist zu beachten, dass die ersten $(L-1)m$ Reihen von $Y_{k+1|L}$ identisch mit den letzten $(L-1)m$ Reihen von $Y_{k|L}$ sind. Dies bedeutet, dass $Y_{k+1|L}$ für den Fall $Y_{k+L|1}$ direkt durch $Y_{k|L}$ ersetzt werden kann.

$$Y_{k+1|L} = \tilde{A}Y_{k|L} + \tilde{B}U_{k|L+1} + \tilde{E}E_{k|L+1} \tag{4.41}$$

Auf der Basis der erweiterten (Projektions-) Matrizen $U_{k+1|L}, Y_{k|L}, Y_{k+1|L}$ wird der Spaltenraum mit der QR- (LQ-) Zerlegung errechnet. Dies führt zur Gleichung 4.42. Dabei ist W_i [113] eine künstliche Variable (Matrix), die das Rauschen reduzieren soll. Im vorliegenden Fall soll die Auswertung ausschließlich auf Systemausgaben beschränkt sein. Es genügt daher, die Matrix $Y_{k|L}$ und deren Extrakt $Y_{k+1|L}$ der QR- (LQ-) Zerlegung zu unterziehen.

$$\frac{1}{\sqrt{K}}\tilde{Y} = \frac{1}{\sqrt{K}} \begin{bmatrix} U_{k+1|L} \\ W_i \\ Y_{k|L} \\ Y_{k+1|L} \end{bmatrix} = RQ = \tag{4.42}$$
$$\begin{bmatrix} R_{11} & 0 & 0 & 0 \\ R_{21} & R_{22} & 0 & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ R_{41} & R_{42} & R_{43} & R_{44} \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}$$

Durch Einsetzen der entsprechenden R_{ij}-Untermatrizen aus 4.42 in Gleichung 4.41 und unter Berücksichtigung der Grenzwertbetrachtung aus [113] folgt 4.43.

$$[R_{41} \quad R_{42}] = \tilde{A}[R_{31} \quad R_{32}] + \tilde{B}[R_{11} \quad 0] \tag{4.43}$$

Aus 4.43 ergibt sich 4.44, die den weiteren Bearbeitungen 4.45 und 4.46 zugrunde liegt.

$$[R_{42}] = \tilde{A}[R_{32}] \tag{4.44}$$

$$Z_{K+1|L} := R_{42} \tag{4.45}$$

$$Z_{K|L} := R_{32} = USV^T \tag{4.46}$$

Nach [113] kann nun die erweiterte Beobachtbarkeitsmatrix O_L aus $Z_{K|L}$ und $Z_{K+1|L}$ bestimmt werden. Dies erfolgt im konkreten Fall auf der Basis des Spaltenraums, der durch die Matrizen R_{32} und R_{42} gebildet wird. Dabei wird O_L 4.48 mittels Singulärwertzerlegung (SVD) 4.47 berechnet.

$$\begin{aligned} \begin{bmatrix} R_{32} \\ R_{42} \end{bmatrix} &= \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} S_n & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \\ &= \begin{bmatrix} U_{11}S_nV_1^T \\ U_{21}S_nV_1^T \end{bmatrix} \end{aligned} \tag{4.47}$$

Wie in 4.48 dargestellt kann die erweiterte Beobachtbarkeitsmatrix O_L nun direkt aus den SVD Ergebnissen extrahiert werden. Dabei ist zu beachten, dass Lm, vorgegeben durch $Y_{k|L}$ und y_k, die maximal mögliche Ordnung des zu identifizierenden Systems angibt.

$$O_L = U(1 : Lm, 1 : n) = U_{11} \tag{4.48}$$

Die vereinfachte Zustandsraummodell-Gleichung für autonome Systeme kann daher wie in 4.49 geschrieben werden. Dies ermöglicht eine direkte Berechnung der Zustands-Sequenz X_k.

$$Y_{k|L} = O_L X_k \tag{4.49}$$

Es ist zu beachten, dass die Zustands-Sequenz X_k aus einer Menge von Zustandsvektoren besteht und daher als Matrix dargestellt wird.

$$X_k \equiv \begin{bmatrix} x_k & x_{k+1} & x_{k+2} & \cdots & x_{k+K-1} \end{bmatrix} \in R^{n \times K} \tag{4.50}$$

4.8 Kernel-Methoden

4.8.1 Einführung

Bei Kernel-Methoden [4] [125] werden die vorhandenen Daten (x) in einen höher dimensionalen Merkmalsraum, der Dimension F, abgebildet. Die Strategie, die dabei Anwendung findet, ist, die Daten in einen Raum zu transformieren, in

[4] Dieses Unterkapitels basiert auf Auszügen von [125].

dem die Struktur als lineare Beziehung dargestellt werden kann. Dies soll mit einer Transformation erreicht werden, die die Nichtlinearitäten mit Hilfe einer Merkmalsabbildung $\phi : x \in R^n \rightarrow \phi(x) \in F \subseteq R^N$ in einen sog. Merkmalsraum überführt, so dass die Daten eine lineare Charakteristik aufweisen. Zu beachten ist dabei die Wahl der Abbildung ϕ, s. hierzu die Transformation in Grafik 4.5, in der ϕ die Daten in einen Merkmalsraum einbettet. Die so linearisierten Daten ermöglichen die Anwendung von klassischen linearen Methoden der Algebra, wie beispielsweise Singulär-Wertzerlegung (SVD), Hauptkomponentenanalyse (PCA), QR-Zerlegung oder lineare Regression.

Die Wahl von ϕ ist nicht unbedingt trivial, da die Abbildung sehr hoch dimen-

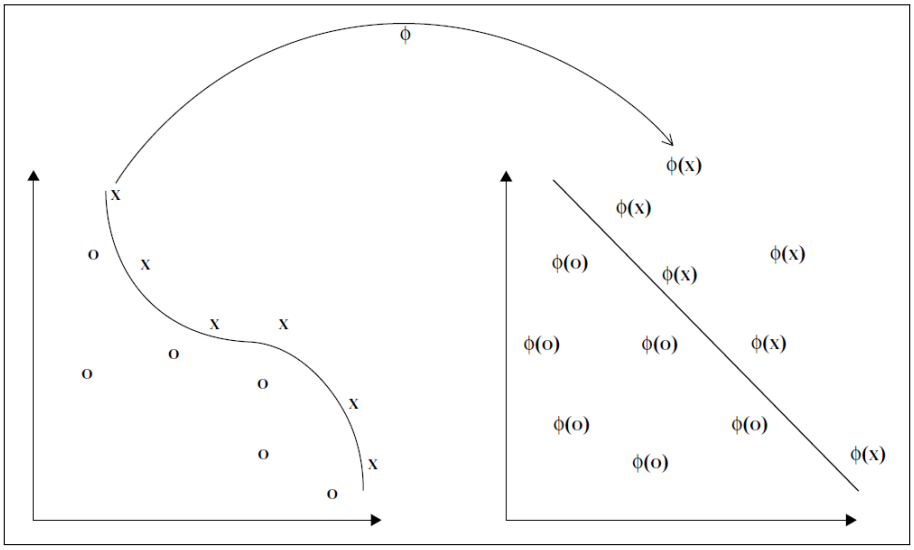


Abbildung 4.5: Kernel-Funktion

sional sein kann. Dies kann eine mitunter sehr aufwändige Bestimmung und Berechnung erforderlich machen. Einen rechnerischen Ausweg bilden Kernel-Funktionen. Kernel definieren die Abbildung ϕ implizit, nämlich in Form des inneren Produkts (siehe Gleichung 4.51), das sich aus den Argumenten errechnet. Dies ist eine einfache Art der Generierung von Merkmalsabbildungen von Daten und somit eine indirekte Linearisierung. Das wird als Kernel-Trick bezeichnet.

$$K(x_s, x_t) = \phi(x_s)^T \phi(x_t) \quad x_s, x_t \in R^n \tag{4.51}$$

Kernelmethoden ermöglichen auch die Konstruktion von Merkmalsräumen auf der Basis mehrdimensionaler Eingabewerte (Eingaberäume), wie z.B. Matrizen. Dies soll beispielhaft in Gleichung 4.52 dargestellt werden, in der die Abbildung ϕ auf eine Hankel-Matrix (Y) aus Datenpunkten (y_i), die eine nicht-lineare

Relation (φ) zueinander aufweisen, Anwendung findet.

$$\Phi_y \equiv \phi(Y) = \phi\left(\begin{bmatrix} \varphi(y_0) & \varphi(y_1) & \cdots & \varphi(y_{j-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(y_{i-1}) & \varphi(y_i) & \cdots & \varphi(y_{i+j-2}) \end{bmatrix}\right) \tag{4.52}$$

Bei mehrdimensionalen Werten, wie im vorliegenden Fall (Gl.4.52), wird die Abbildung ϕ nicht auf jedes Element (y_i) der Matrix angewandt, sondern auf das Gesamtobjekt. Bei der Berechnung des Kernels (K) muss daher nur das innere Produkt zwischen Paaren von Datenpunkten, d.h. den Matrizen, berechnet werden. Es ist nicht erforderlich, ϕ für jedes Element y_i auszuwerten. Diese Vereinfachung resultiert in $K^y \equiv K(Y,Y) = \Phi_y^T \Phi_y$.

4.8.2 Skaleninvarianter Kernel

Ein weitverbreiteter und in vielen Anwendungen erfolgreich eingesetzter Kernel ist der Gauß'sche Kernel (RBF-Kernel):

$$K(y, y_i) = e^{\frac{\|y-y_j\|^2}{2\sigma^2}} \tag{4.53}$$

Hierbei ist σ die Weite des Kernels und $\|x - x_i\|^2$ die euklidische Distanz. Die euklidische Distanz, wie u.a. in [67] beschrieben, wird nicht skaleninvariant (skalenunabhängig), selbst wenn die Messwerte (Samples) zur selben Klasse gehören. Skaleninvarianz beschreibt z.B. die Eigenschaft eines Zustands einer Variablen (x), bei der trotz Veränderung der Betrachtungsgrößen (Skalierung) die Charakteristik der Daten inklusive ihrer Eckwerte weitestgehend exakt gleich bleibt. In der Praxis bedeutet dies, dass sich bei einer Funktion ($f(x)$), die von einer Variablen abhängt, trotz Reskalierung der Variablen ($x \to ax$), z.B. durch einen Faktor (a), die wesentlichen Eigenschaften nicht ändern. Solche Fälle können mit skaleninvarianten Metriken, z.B. dem Spectral Angle Mapper (SAM), verarbeitet werden. SAM fokussiert dabei die Winkel zwischen zwei Vektoren (siehe Gleichung 4.54).

$$\alpha(y_i, y_j) = arccos(\frac{y_i \cdot y_j}{\|y_i\| \cdot \|y_j\|}) \tag{4.54}$$

Die Integration der SAM-Methodik in den RBF-Kernel führt zu folgender Gleichung (4.55):

$$K_{SAM}(y_i, y_j) = e^{\frac{\alpha(y_i,y_j)^2}{2\sigma^2}} \tag{4.55}$$

Diese Metriken wurden häufig bei Fernerkundungsproblemen genutzt, da sie robust gegen Variationen in der spektralen Energie sind [67].

4.9 Support-Vektor-Maschinen

4.9.1 Einführung

In praktischen Anwendungen kommt es häufiger vor, dass sich Daten nichtlinear verhalten. Für die Auswertung gebräuchlicher Ansätze werden üblicherweise lineare Methoden genutzt. Diese Diskrepanz wird durch die Support-Vektor-Maschinen-Regression (SVR) [29] [119] beseitigt.
Die Grundidee hinter der Support-Vektor-Maschinen-Regression [29] [119] ist, die Daten in einen höher dimensionalen Merkmalsraum H (Hilbert space) zu transformieren. Dies geschieht durch eine nichtlineare Abbildung φ, die die Daten linear in H darstellt. Anschließend wird im Merkmalsraum eine lineare Regression durchgeführt. Dazu wird ein Trainingsset mit l Trainings-Samples (Ein- und Ausgabe) benötigt:

$$(x_1, y_1), \ldots, (x_l, y_l) \in R^n \times R \tag{4.56}$$

Die nicht-lineare Abbildung $\varphi(\cdot) : R^n \rightarrow H$, die die Sampels zu einem neuen (linearen) Datensatz in einem höher-dimensionalen Raum abbildet, wird in Gleichung 4.57 dargestellt. Dabei ist r die Größe des Merkmalsvektors (z.B. die Dimension des Merkmalsraums), und mit n bezeichnet man die Anzahl der Samples. Die Anwendung der nicht-linearen Abbildung wird in Grafik 4.6 illustriert.

$$(x_1, x_2, \ldots, x_N) \mapsto_m (\varphi_1, \varphi_2, \ldots, \varphi_r) \circ (x_1, x_2, \ldots, x_N) \tag{4.57}$$

Ein Ziel der SVR [132] ist es auch, eine Funktion $f(x)$ zu finden, die eine

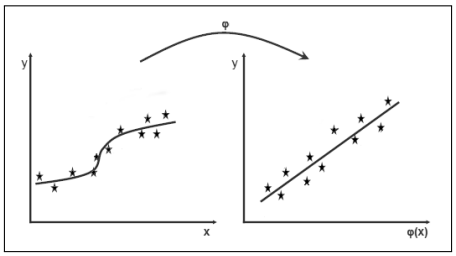


Abbildung 4.6: Nichtlineare Transformation

möglichst geringe Abweichung, einen Fehler, von der realen Zielvariablen (y) im Trainingsdatensatz hat. Häufig wird die Abweichung mit ϵ angegeben und daher die SVR als ϵ-SVR bezeichnet. Für die Bestimmung von $f(x)$ wird das Regressionsmodell $y_t = f(x_t) + e_t$, in dem die Funktion $f : R^d \rightarrow R$ ermittelt werden soll, verwendet. Für den linearen Fall hat f die nachfolgend angegebene Form: $f(x) = \langle w, x \rangle + b$ mit $\omega \in X$, $b \in R$, wobei $\langle \cdot, \cdot \rangle$ das Skalarprodukt in X angibt. Nach Substitution von f im Regressionsmodell kann für die Ermittlung von $f(x)$ das Modell in Gleichung 4.58 angenommen werden:

$$f(x) = w^T \varphi(x) + b + e_k \tag{4.58}$$

Die Bestimmung des Vektors w und $b \in R$ als Regression kann als Optimierungsproblem, siehe Gleichung 4.59, formuliert werden. Die Vorgehensweise zur Lösung dieses Problems ist detailliert in [119] beschrieben.

$$\begin{aligned} \min_{w,b,e} \mathcal{J}(w,e) &= \frac{1}{2} w^T w + \frac{\gamma}{2} \sum_{k=1}^{n} e_k^2 \\ \text{Unter der Voraussetzung: } y_t &= w^T \varphi(x_t) + b + e_t, \\ t &= 1, \ldots, l, \quad \gamma > 0, \quad e_k \geq 0 \end{aligned} \tag{4.59}$$

Es resultiert ein lineares System, nachdem die Variablen w und e eliminiert wurden. Das Optimierungsproblem, s. Gleichung 4.60, kann als SVM-Modell ausgedrückt werden, wobei a_k ein Koeffizient ist und $K(x, x_i)$ einen positiv definierten Kernel repräsentiert:

$$\hat{f}(x) = \sum_{k=1}^{n} \hat{a}_k K(x, x_i) + \hat{b} \tag{4.60}$$

Die Nutzung des SVM-Modells bietet den Vorteil, dass im Rahmen der Trainingsphase die nichtlinearen Daten durch Regression [132] als Support-Vektoren im höher dimensionalen Raum gelernt werden. Hierdurch werden nicht-lineare Zusammenhänge direkt abgebildet. Anschließend ist eine unkomplizierte Anwendung bei der Prädiktion basierend auf den Support-Vektoren möglich. In der Praxis könnte dies auch für die Prädiktion von nicht-linearem Systemverhalten Anwendung finden.

4.9.2 Least-Squares-Support-Vektor-Maschinen

Eine spezielle Form der Support-Vektor-Maschinen ist die Least-Squares-Support-Vektor-Maschine [141] (LS-SVM). Diese unterscheidet sich von der normalen SVM dadurch, dass anstelle eines rechenintensiven Optimierungsproblems (Gl. 4.59) eine Menge linearer Gleichungen zu lösen sind. Dies führt nach [142] in Bezug auf die „normale“ SVM zu einer Reformulierung der Optimierungsgleichung (siehe Formel 4.61.

$$\begin{aligned} \min_{w,b,e} \mathcal{J}(w,e) &= \frac{1}{2} w^T w + \frac{\gamma}{2} \sum_{i=1}^{n} e_i^2 \\ \text{Unter der Voraussetzung: } y_i \left[w^T \varphi(x_i) + b \right] &= 1 - e_i, \forall i \\ \gamma &> 0 \end{aligned} \tag{4.61}$$

Daraus resultieren nachfolgende Bedingungen, dargestellt als lineare Gleichungen, für die Optimalität:

$$w = \sum_{i=1}^{N} \alpha_i y_i \varphi(x_i) \tag{4.62}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{4.63}$$

$$\alpha_i = \gamma e_i \quad i = 1, \ldots, n \tag{4.64}$$

$$y_i \left[w^T \phi(x_i) + b \right] - 1 + e_i = 0 \quad i = 1, \ldots, n \tag{4.65}$$

Hierbei müssen w, e eliminiert werden, um die Lösung für α, b zu finden. Dabei gibt α die Support-Vektoren an.

Kapitel 5

Realisierungsansätze

Im nachfolgenden Kapitel werden die im Rahmen der vorliegenden Arbeit entwickelten Realisierungsansätze vorgestellt. Diese umfassen eine Fehlerinstrumentierung mit Hilfe von Kontrollgruppen (CGroups), eine nichtlineare Systemidentifikation, eine statistische Anomalie-Erkennung auf der Basis eines Kalman-Filters, eine Systemmodellierung mittels Bayes'scher Netze und die Auswertung der erstellten Modelle unter Zuhilfenahme von Model-Checking.

5.1 Fehler-Instrumentierung

5.1.1 Kontrollgruppen

Häufig ist es notwendig, das Testen bzw. die Evaluierung von Software-Anwendungen in Grenzbereichen zu betreiben. Dazu müssen gelegentlich Zustände bei der Applikation wie auch im Betriebssystem erzeugt werden, die üblicherweise im Normalbetrieb nicht auftreten würden. Solche Zustände erreicht man mittels Stimulation bzw. gezieltem Einbringen von Anomalien oder Fehlern. Für komplexere Objekte werden hierzu Vorrichtungen, z.B. Testsysteme, genutzt, die für einen Anwendungsfall speziell entwickelt oder an die jeweiligen Gegebenheiten angepasst werden müssen.
Teilweise können Testsysteme mit Mitteln bzw. Mechanismen (Bordmitteln), die bereits in das System integriert sind, aufgebaut werden. Bei Verwendung dieser Funktionalitäten ist der Aufwand der Anpassung an das jeweilige Problem deutlich geringer, da definierte Schnittstellen im System bereits vorhanden sind.
Neuere LINUX-Betriebssysteme bieten entsprechende nutzbare Eigenschaften, z.B. eine integrierte Schnittstelle zum Ressourcen-Management. Bei den erwähnten Mechanismen handelt es sich z.B. um Kontrollgruppen, sogenannte CGroups [131], die direkt vom LINUX-Kernel bereitgestellt werden, siehe Abbildung 5.1.
CGroups unterstützen direkt die Limitierung, Priorisierung, Isolation und Zugriffssteuerung von Systemressourcen. Um das zu gewährleisten werden innerhalb der CGroups Gruppen angelegt mit Berechtigungen bzw. Rechten für di-

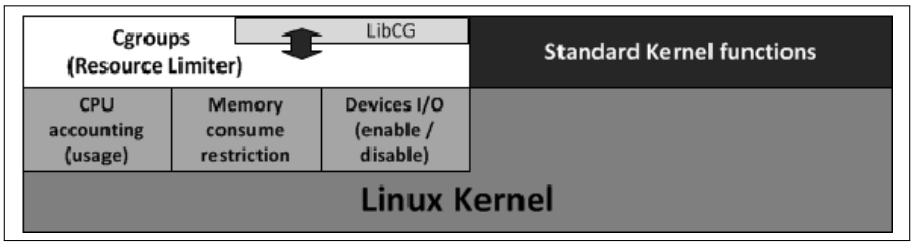


Abbildung 5.1: Überblick CGroups im Linux Kernel

verse Betriebsmittel (z.B. CPU, Speicher, Geräte). Die Ressourcen-Limitierung auf Gruppenebene wird durch eine vordefinierte Menge an bereitgestellten Systemressourcen, die nicht überschritten wird bzw. überschritten werden kann, erreicht. Eine solche Einschränkung kann auch anderweitig genutzt werden, u.a. um Anwendungen in einem isolierten Bereich (Sandbox) zu betreiben und darin die Anwendungsressourcen zu beschränken. Die Isolation der Kontrollgruppen bietet eine effektive und restriktive Möglichkeit, die Gruppen gegeneinander abzuriegeln. Dies betrifft speziell die in einer CGroup bereitgestellten Systemressourcen, die sowohl gegen die im System verfügbaren als auch gegen die übrigen vom Nutzer angelegten Gruppen abgeschottet werden. Außerdem kann über das Accounting, das den Verbrauch von bestimmten Betriebsmitteln bzw. Komponenten misst, verwaltet und diese den CGroups zuweist, eine individuelle Kontrolle der Ressourcen vorgenommen werden. Der Vorteil einer solchen systematischen Erfassung des Ressourcenkonsums - verglichen mit System-Messwerkzeugen (wie top, htop) - bietet durch eine zeitlich genauere Auflösung für LINUX eine interne schnellere Reaktionsmöglichkeit auf Veränderungen im Ressourcenkonsum, da die Daten den Überwachungswerkzeugen im Sekundentakt zur Verfügung gestellt werden.
CGroups sind in viele LINUX-Distributionen integriert, so dass eine entsprechende Nutzung ohne Installation spezieller Treiber und ohne Modifikation des Kernels möglich ist. Seit dem Jahr 2010 wird Hardware- und Ressourcen-Unterstützung, u.a. für eine Verwaltung der CPU- bzw Arbeitsspeichernutzung und die individuelle Steuerung von I/O Geräten, bereitgestellt. Die nachfolgende Aufzählung gibt eine Übersicht über verfügbare CGroups, die auch als CGroup-Teilsysteme bezeichnet werden:

- blkio - Hiermit können Limits für die Ein-/Ausgabe zu und von blockorientierten Geräten, z.B. physikalischen Laufwerken (Festplatte, SSD, USB, etc.) gesetzt werden.
- cpu - Dies benutzt den Scheduler, um Anwendungen innerhalb der CGroup Zugriff auf die CPU zur Verfügung zu stellen.
- cpuacct - Dieses Teilsystem generiert automatisch Berichte über verwendete CPU-Ressourcen, die von Anwendungen innerhalb der CGroup genutzt

werden.

- cpuset - Mit dieser Komponente werden einzelne CPUs (auf einem Multicore-System) und Speicherknoten den Tasks (Programme) in einer CGroup zugeordnet.
- devices - Dieses Subsystem erlaubt oder verweigert den Zugriff auf Geräte zur Nutzung in einer Kontrollgruppe.
- memory - Hiermit können Grenzwerte für die Speichernutzung der in einer Kontrollgruppe befindlichen Programme gesetzt werden. Zusätzlich werden automatisch Berichte über die Nutzung der von den CGroups verwendeten Speicherressourcen generiert.

Betriebsszenarien von Software-Anwendungen sind mit CGroups bedingt nachstellbar, jedoch ermöglicht eine Kombination von Ressourcenlimitierung und Last-Generatoren eine Emulation. Ansätze können sein:

- eine dynamische differenzierte Reduktion bzw. eine Erhöhung von Ressourcen zur Laufzeit
- das Bereitstellen eines definierten Ressourcen-Volumens bei Ressourcen, deren Bereitstellung nur in der Quantität beschränkt werden kann z.B. CPUs

Letzteres kann durch Last-Generatoren erzielt werden, die die in der Quantität beschränkte Resource ebenfalls nutzen und parallel zur Anwendung betrieben werden. Dies setzt voraus, dass Software-Anwendungen und der Last-Generator in der identischen CGroup ausgeführt werden. Ein solcher Aufbau macht auch die Simulation von speziellen Fehlerszenarien durchführbar.
Die in LINUX integrierten CGroups sehen standardmäßig eine dynamische Anpassung der Parameter nicht vor, jedoch können durch Anwendung der API-Bibliothek (Libcg) während der Systemlaufzeit Größenänderungen der Parameter erfolgen. Dies ermöglicht die Generierung von Szenarien während der Laufzeit, wie z.B. die Reduktion von Parameterwerten unter das verwendete Quantum der Applikation. Hierdurch ist es möglich, Software unter unterschiedlichen Umgebungsbedingungen, die auch Grenzfälle einschließen, zu testen, ohne das Betriebssystem oder die Zielapplikation zu modifizieren. Das wird erreicht, indem eine CGroup, bei der die Menge von verfügbaren Ressourcen eingestellt ist, instanziiert wird und darin sowohl Lastgeneratoren wie auch die zu testende Software betrieben wird. Das wird in Abbildung 5.2 dargestellt und gleicht einem Ressourcen-Sandboxing. Auf diese Weise ist die Simulation von Grenzwertsituationen, beispielsweise einer sehr hohen CPU-Last, durchführbar, ohne das Computersystem übermäßig zu belasten bzw. es einem Stress auszusetzen und so in einen anormalen Zustand zu bringen.
Eine Verringerung von Ressourcen, wie z.B. eine Reduktion des verfügbaren Arbeitsspeichers während der Laufzeit, kann nicht nur zu realistischen Betriebsszenarien führen, sondern auch eine Anwendung in Grenzfällen ermöglichen.

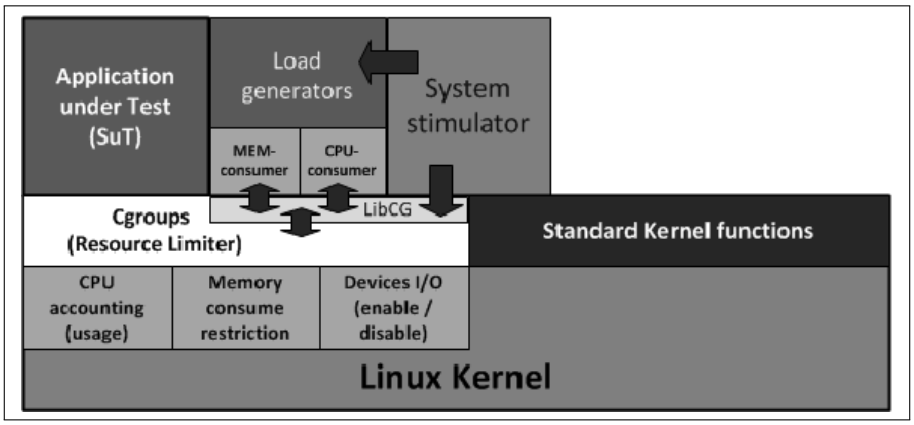


Abbildung 5.2: Einstellbare (CGroups) Sandbox mit Lastgenerator

Ein Beispiel für solch ein Verhalten ist ein zunehmender Speicherbedarf wie er durch Speicherlecks hervorgerufen wird, der üblicherweise das Computersystem in einen Swap-Zustand oder zum Zusammenbruch (Crash) bringt. Wie bereits erwähnt, ermöglicht die Anwendung des oben dargestellten Ressourcen-Sandboxings jedoch eine Simulation, ohne das Verhalten des Betriebssystems zu beeinflussen. Hierdurch sind Grenzfälle im Applikationstest wie Out-Of-Memory (OOM) umsetzbar. Unter Bezugnahme auf die erwähnten Szenarien wurde für die dynamische Gestaltung von Grenzfällen der von den CGroups abgeleitete Name CGFail [44] gewählt.

5.1.2 Fehler-Instrumentierungsansatz CGFail

In dem nachfolgenden Kapitel wird ein Überblick über die aktuell unterstützten Fähigkeiten von CGFail gegeben.
In Abhängigkeit von den oben bereits erwähnten kontrollierten Ressourcen (blkio, cpu, cpuacct, cpuset, devices, memory) kann durch Analyse des zu testenden Systems eine Ableitung mehrerer Fehlerszenarien erfolgen, welche mittels CGFail emulierbar sind. Des Weiteren können die Eigenschaften von SWIFT (Software Implemented Fault Tolerance), einer Gruppe von Methoden, die Hardware-Fehler detektiert und eine Tolerierung dieser Fehler durch die Software ermöglicht, mit CGFail in Bezug gesetzt werden. Da beim Erstellen der vorliegenden Arbeit das Ressourcenmanagement für Prozessoren (CPU), Speicher und Geräte durch die CGroups unterstützt wird, sind nur die Fehlerzustände bzw. -szenarien, die damit im Zusammenhang stehen, generierbar. Die im Folgenden vorgestellten Fehlerklassen lassen sich durch Beschränken dieser Größen, speziell unter dem Gesichtspunkt von Grenzfällen für die Software, emulieren.

- Niedrige CPU-Verfügbarkeit

- Geringe Speicher-Verfügbarkeit
- Nicht genügend Speicher (Out of memory)
- Dauerhaft und dynamisch nicht verfügbare Geräte

Durch eine Kombination des CGroup-Ansatzes mit in den CGroup-Sandboxen laufenden Lastgeneratoren, wie in Abbildung 5.2 dargestellt, kann der Funktionsumfang erweitert werden. Sofern die Parameter der Lastgeneratoren und jede individuelle Ressource über dynamische Einstellmöglichkeiten verfügen, werden die weiteren nachstehend aufgeführten Fehlerklassen eingeführt:

- CPU-Überlast
- Speicherlecks

Der Vergleich ähnlicher Arbeiten [61] [143] [14] lässt die Vermutung zu, dass bei den meisten SWIFT-Ansätzen der Fokus auf Prozessorfehlern liegt, denen unterschiedliche Ursachen zu Grunde liegen können, u.a. Ausfall des Prozessors oder fehlerhafte Ausführung von Instruktionen. Einige Lösungen unterstützen zusätzlich Arbeitsspeicher- oder I/O-Treiber-Fehlerinjektionen. Die meisten dieser Ansätze benötigen die Installation spezieller Treiber, die vom Betriebssystem (OS) und teilweise auch von dessen Version abhängig sind. Andere Techniken erfordern systemnahe Änderungen am OS oder die Modifikationen des zu testenden Objekts (System under Test = SuT) während der Laufzeit. Dies kann das Verhalten des Betriebssystems beeinflussen. Wenige Lösungen basieren auf maschinennahen Betriebssystemfunktionen, die es erfordern, dass die zu testende Anwendung in einem speziellen Betriebsmodus, dem Trace-Modus, läuft. Der vorgestellte Ansatz CGFail ist auf allen LINUX-Betriebssystemen anwendbar, da die meisten LINUX-Distributionen die benötigten Komponenten direkt beinhalten. Die Hardware-Architektur des zu Grunde liegenden, genutzten Computersystems beeinflusst nicht die Anwendbarkeit dieses Ansatzes.

5.2 Nichtlineare System-Identifikation

Bei Informationssystemen (IS), deren Charakteristik unbekannt ist oder deren Verhalten als nicht-linear angenommen wird, kann die Anwendung von linearen Methoden unzureichende Ergebnisse erzielen. Nichtlinearität wird angenommen, wenn sich beobachtbare Messgrößen nicht proportional zueinander verhalten. Für die Analyse kann die Anwendung von linearen Auswertemethoden, z.B. der Statistik oder Algebra, unzureichende Ergebnisse erzielen, da die Eingangsgrößen, wie beispielsweise Nutzereingaben oder autark arbeitende Prozesse, nicht immer für den Nutzer beobachtbar sind. Die Spezifikation des Systemverhaltens erfolgt in der Praxis üblicherweise mittels Differentialgleichungen bzw. Zustandsraummethoden. Bei Informationssystemen erfolgt, z.B. im Rahmen des Entwicklungsprozesses, normalerweise keine mathematische Verhaltensspezifikation. Dies wird nicht nur durch die variierende Betriebskonfigurationen solcher Systeme beeinflusst, sondern auch durch die Tatsache, dass

für viele solcher praktischen Anwendungen auf Grund variierender bzw. zur Entwicklungszeit unbekannter Parameter entweder keine mathematischen Beschreibungen existieren oder diese ungenau sind. Eine Möglichkeit, das Systemverhalten auf der Basis von Messdaten mathematisch zu spezifizieren, ergibt sich durch die Systemidentifikation, bei der mittels mathematischer Transformationen das Zustandsraummodell ermittelt wird. Die Systemidentifikation wurde bisher überwiegend für lineare Modelle entwickelt. Repräsentative Algorithmen dieser Kategorie, die eine unterraum-basierte Systemidentifikation ermöglichen, sind DSR [113] und N4SID [146]. Über die Ableitung von nichtlinearen Zustandsraummodellen wurde bisher nur wenig [40] publiziert. Der Ansatz [40] setzt voraus, dass das zu behandelnde System über beobachtbare Eingänge und Ausgänge verfügt. Da das in praktischen Anwendungen nicht immer gegeben ist [66] [41] [9] [90], wurde eine Methode entwickelt, die ausschließlich auf der Grundlage von Systemausgaben eine System-Identifikation durchführt und dabei nicht-lineares Systemverhalten berücksichtigt. Es handelt sich um eine Modifikation des in Kapitel 4.7 beschriebenen kombinierten deterministischen und stochastischen System- und Realisierungs- Algorithmus (DSR). Dieser wurde zur Verarbeitung von Nichtlinearitäten mit Kerneltransformation (siehe Kapitel 4.8.1) erweitert.
Eine Hankel-Matrix mit nicht-linearen Relationen zwischen den Elementen wird in Gleichung 4.52 dargestellt. Basierend auf einer solchen Matrix wird der Kernel $K_y \equiv \Phi_y^T \Phi_y$ unter Zuhilfenahme des Kernel-Tricks berechnet. Dabei repräsentiert die Zeilenanzahl der Hankel-Matrix die Dimensionen des Kernelraums. Zur Beibehaltung der Datencharakteristik, z.B. durch Reskalierung, wurde der üblicherweise verwendete Gauß'sche Kernel gegen einen SAM-Kernel (siehe Kapitel 4.8.2) ausgetauscht. Der SAM-Kernel wird als Standard für sämtliche weitere Kernel-Transformationen in diesem Kapitel angesehen.
Auf der Basis des bei der Kerneltransformation eingeführten Merkmalsraums wird die Projektion der Daten durch Anwendung der Kernel-QR-Zerlegung (K-QR) [125] berechnet. Es resultiert eine orthonormale Matrix Q und eine reduzierte (komprimierte) Darstellung der Daten R. Die reduzierte Form (R) beinhaltet alle relevanten Informationen des Systems. Somit bietet es sich an, die bei der Kernel-Transformation entstandenen Daten aus R in Gleichung 4.43 einzusetzen und auf dieser Basis weiter zu arbeiten. Auf Grund der Tatsache, dass sich die Daten im Kernelraum befinden, muss nach [125] anstelle der gewöhnlichen SVD eine Kernel Singulär-Wertzerlegung (K-SVD) in Erwägung gezogen werden. Bei der Berechnung der K-QR-Zerlegung ($K = \Phi'_y \Phi_y = R'Q'QR = R'R$) muss im Allgemeinen die Matrix Q im hoch-dimensionalen Merkmalsraum berechnet werden, das jedoch hier für das weitere Vorgehen nicht notwendig ist. Es wird eine weniger rechenintensive Version [154] verwendet, die ohne die aufwändige Berechnung von Q auskommt und die lediglich R als Ergebnis liefert. Danach kann auf der Basis von R und wie in den Gleichungen 4.46, 4.47 und 4.48 angeführt die erweiterte Überwachungsmatrix O_L aus R durch Anwendung der K-SVD extrahiert werden. Es ist zu beachten, dass im Gegensatz zu den in Gleichung 4.44 aufgeführten Matrizen R32 und R42 für die Abschätzung des Spaltenraumes die im Kernel-Bereich korrespondierenden Submatrizen KR32

und KR42 genutzt werden.
Die Singulär-Wertzerlegung des Kernels ($K \in R^{N \times N}$) bis Rang (r) ist in Gleichung 5.1 wie folgt definiert: $Q^r = Q(1:N, 1:r)$ und $\Delta^r = \Delta(1:r, 1:r)$, wobei es sich bei Δ um eine Diagonal-Matrix handelt, deren Elemente aus $K = Q\Delta Q'$ resultieren und bei denen es sich um die korrespondierenden Eigenwerte handelt.

$$K^r = \left[\Phi_y Q^r \left(\Delta^r\right)^{-\frac{1}{2}}\right] \left[\left(\Delta^r\right)^{\frac{1}{2}}\right] \left[\left(Q^r\right)^T\right] \equiv U^r \Sigma^r \left(V^r\right)^T \tag{5.1}$$

Auf Grund der Tatsache, dass im Kernelraum die Matrix U^r die wichtigsten Hauptkomponenten von Φ_y enthält und ausschließlich implizit als die lineare Kombination der abgebildeten Daten $U^r = \Phi_y Q^r (\Delta^r)^{-\frac{1}{2}} := \Phi_y \alpha$ [24] definiert ist, muß für die Berechnung von U^r nach [24], [11], [134] ein weiterer Datensatz Φ_{y_1} genutzt werden. Abhilfe schafft die K-QR-SVD, die direkt auf das Ergebnis R der K-QR-Berechnung [154] eine normale (lineare) SVD [23] anwendet und die Eigenwerte berechnet.
Die Ergebnisse dieser K-QR-SVD entsprechen der rechten Seite von Gleichung 4.47. Obwohl die Eigenwerte im Kernel-Bereich unter Anwendung der K-QR-SVD resultieren, ist eine Rücktransformationen in den Ausgangsraum (Y), wie sie üblicherweise durch Pre-Image-Abschätzungsmethoden [8] [80] durchgeführt wird, im konkreten Fall nicht notwendig, denn die Eigenwerte können auch als die Hauptwinkel zwischen den Matrizen R32 und R42 aufgefasst werden. Dazu sind die Hauptwinkel zwischen zwei Unterräumen (Matrizen) [153] eindeutig ($0 \leq \Theta_1 \leq \ldots \leq \Theta_k \leq \pi/2$) als $cos(\Theta_k) = \max \max u'v \quad u \in R32, v \in R42, k = 1...K \subset$ „nc-Blockspalten“ definiert. Die Werte von $cos(\Theta_k)$ werden auch als die kanonische Korrelation des Matrixpaares (KR32, KR42) bezeichnet.
Abschließend kann auf der Basis der Winkel die Beobachtbarkeitsmatrix O_L wie in Gl. 4.48 bestimmt werden und damit die Abschätzung der Zustandssequenz (Gl. 4.49) erfolgen.
In der Praxis werden häufig auf der Basis der zuvor angeführten Berechnungen die Markov-Parameter (s. Gleichung 4.39) für die Zustandsraumdarstellung ermittelt. Diese Darstellung schränkt aber insofern ein, als sie nur lineares Verhalten abdecken kann. Dies würde im Falle von nichtlinearen Transfer- und Überwachungsfunktionen entweder eine Linearisierung des Verhaltens erfordern oder in eine unpräzise Modellierung münden. Eine präzisere Darstellung von nichtlinearen Funktionen bieten neben Gauß'schen-Prozessen (GP) u.a. die Support-Vektor-Maschinen (SVM). Bei beiden Ansätzen lassen sich die Modelle aus Messdaten ableiten. Konzeptionell kann es bei GP, bedingt durch die Darstellung als überlagerte Gaußverteilungen, bei Regressionsanwendungen am Rande des Wertebereichs zu erhöhten Ungenauigkeiten kommen. Bei Support-Vektor-Maschinen verhält sich das Modell im Randbereich entsprechend der beim Lernvorgang zu Grunde liegenden Daten. Allerdings bieten Gauß'sche Prozesse die Möglichkeit, die Kovarianz, die die Präzision des Modells für jeden Datenpunkt individuell angibt, direkt aus dem Modell zu bestimmen. Jedoch kann die Kovarianz nicht direkt aus dem SVM-Modell, das die Transfer- bzw. Überwachungsfunktionen repräsentiert, ermittelt werden; hierfür ist eine zweite

SVM oder eine aufwändigere Berechnung erforderlich.
In der vorliegenden Arbeit wird die nichtlineare Transfer- und Überwachungsfunktion durch eine spezielle SVM-Variante, die Least-Squares-Support-Vector-Machine (LS-SVM) [141], repräsentiert. Das duale LS-SVM-Problem [141] für Transfer- und Überwachungsfunktionen wird in Gleichung 5.2 dargestellt.

$$\min_{f(x),h(x)} \left\| \begin{array}{ccc} \hat{X}_{k+1} & - & f(\hat{X}_k) \\ Y_k & - & h(\hat{X}_k) \end{array} \right\|_F^2 \tag{5.2}$$

Die Lösung des Optimierungs-Problems in Gleichung 5.2 macht die Berechnung von nichtlinearen Transfer- $f(x)$ und Überwachungsmodellen $h(x)$ durch die Anwendung von LS-SVM möglich. Der Pseudo-Code des vorab theoretisch erklärten Algorithmus' wird im nachstehenden Algorithmus 2 (in Pseudo-Code) präsentiert.

Algorithmus 2 Nichtlineare Kernel-Systemidentifikation inkl. SVM-Modellierung

Eingabe: Prozessausgabe (Messwerte)
1: Konstruktion der Hankel-Matrix auf der Basis der Eingabe-Messwerte
2: Berechnung des Kernel (K) auf der Basis der Hankel-Matrix (Systemausgaben)
3: Berechnung der Projektion im Merkmalsraum durch die K-QR(K)
4: Anwendung der SVD auf die Kernel-Transformation von R32 und R42
5: Extraktion der erweiterten Überwachungsmatrix
6: Ableitung der Zustandssequenz
7: Berechnung des LS-SVM-Modells der Transferfunktion $f(x)$ aus der Zustandssequenz
8: Berechnung des LS-SVM-Modells der Überwachungsfunktion $h(x)$ aus der Zustandssequenz und Messwerten
9: **Ausgabe:** $f(x), h(x)$

5.3 Anomalie-Erkennung in dynamischen Informationssystemen

5.3.1 Statistische Anomalie-Erkennung

Statistische Tests sind objektive Methoden zur Klassifikation von Daten, die in der Praxis bei der Bewertung von Meßergebnissen im Hinblick auf die Überprüfung der Gültigkeit einer Annahme zum Einsatz kommen. Ein solch neutrales Testverfahren ist gut geeignet für die Bewertung und Eingruppierung von Daten, beispielsweise für die Erkennung anormaler Messwerte und somit die Klassifikation von Fehlern. In Abhängigkeit von der individuellen Problemstellung und den jeweiligen Daten (z.B. Skalierung, Verteilung, Merkmalsart) muss der anzuwendende Test aus einer Vielzahl statistischer Tests [148] ausgewählt werden,

da ein allgemein gültiger Test zur generellen Lösung der o. a. Thematik aktuell nicht vorhanden ist.
Für die Bewertung von Daten kommen gelegentlich Distanzmaße [111], die den Abstand zwischen Objekten im multidimensionalen Raum bestimmen, zur Anwendung. Diese geben jedoch nur einen Verhältniswert an, der anwendungsspezifisch zu bewerten ist und liefern keine eindeutige Aussage über Anomalien in den Daten. Eine weitere Möglichkeit besteht darin, mit Hilfe der zuvor gemessenen Daten (Historie) ein sogenanntes Toleranzband für die Bewertung neuer Daten zu generieren. Auf dieser Basis kann eine Aussage getroffen werden, ob eine Messung innerhalb oder außerhalb der Toleranz liegt. Sollte ein Messwert von den Erwartungen abweichen, also außerhalb des Toleranzbereichs liegen, handelt es sich um einen Ausreißer und impliziert eine Anomalie. Für die Unterscheidung von Daten in normal bzw. anormal können verschiedene Methoden wie beispielsweise Cluster-Analyse, Diskriminanten-Analyse angewandt werden. Diese werden jedoch hier nicht berücksichtigt.
In der vorliegenden Arbeit werden statistische Signifikanztests angewandt, z.B. der χ^2 Test, der allgemeine Log-Likelihood Verhältnistest oder die Distanz zur Standard-Abweichung während des Betriebes, um entscheiden zu können, ob ein aufgezeichneter Messwert als normal oder anormal angesehen werden kann. Üblicherweise werden diese Tests auf den Mittelwert eines Signals angewandt, jedoch ist bei Prädiktionsanwendungen die Auswertung des Residuums (die Differenz zwischen Prädiktion und Messung) aussagekräftiger. Dies bedingt die enge Verwandtschaft des Residuums zum Prädiktionsfehler.
Ein etabliertes Maß in der Statistik für die Varianz oder Streuung eines Signals ist die Standard-Abweichung. Eine Klassifikation im Bezug auf normales, ungewöhnliches oder abnormales (fehlerhaftes) Verhalten wird unter Bewertung der Abweichung vom Mittelwert durchgeführt.
Folgende Messverfahren wurden angewandt:
3-Sigma-Regel: In praktischen Anwendungen [52] wird ungewöhnliches Verhalten vermutet, wenn es um das zwei- oder dreifache vom Mittelwert abweicht. Hier werden Messungen, welche mehr als drei Sigma vom Mittelwert abweichen, als anormal kategorisiert. Bei der Anwendung der 3-Sigma-Regel wird ein Messwert, der eine Distanz zwischen 4-5-Sigma vom Mittelwert aufweist, als Außreiser bzw. Fehler klassifiziert. Messwerte, die mehr als das fünffache vom Mittelwert abweichen, können als heftige Außreiser angesehen werden. Entsprechend der Normalverteilungskurve liegen nur 0.00006% der Messwerte außerhalb des 5-Sigma Bereichs.
Log-Likelihood-Test: Bei dynamischen Signalen bzw Datenströmen, die kein stationäres Verhalten aufweisen, kann es auf Grund von Signalschwankungen bzw. möglichen Signaldrifts zu Fehlzuordnungen führen. Ein erheblich robusterer Anomalie-Indikator wird [68] durch ein Moving-Average-Filter (bzw. Tiefpass), welches auf die Log-Likelihood-Werte der vergangenen Messungen angewandt wird, erreicht. Dabei wird durch das Moving-Average-Filter ein gleitender Mittelwert der Auftretenswahrscheinlichkeiten der vorangegangenen Messungen gebildet. Dies gleicht einer Glättung. Dieser geglättete Mittelwert wird gegen die Wahrscheinlichkeit des aktuellen Messwertes verglichen.

Goodness-of-Fit-Test: Der Goodness-of-Fit-Test (GOF) beruht auf einer mathematischen Funktion, mit der herausgefunden werden kann wie stark ein beobachteter Wert eines bestimmten gegebenen Signals signifikant von dem erwarteten Wert der Verteilung abweicht und sagt aus, ob und in welchem Umfang Anzeichen für ein unerwünschtes Systemverhalten vorliegen. Eine andere Variante ist die Nutzung einer Messung, die als Maßstab die quantitative Anpassungsgüte beurteilt, die sich aus der Differenz der gemessenen Daten zum Modell ergibt. Typischerweise fassen GOF-Messungen die Diskrepanz zwischen beobachteten Werten und Erwartungswerten des bekannten Modells zusammen. Ein in der Praxis sehr gebräuchlicher Test hinsichtlich der Übereinstimmung zweier Verteilungen ist der χ^2 (GOF) Test.

5.3.2 Adaptives Unscented Kalman Filter

Die Identifikation bzw. Extraktion von Fehlverhalten eines Systems, das sich in den Messungen von Systemparametern niederschlägt, stellt speziell bei dynamischen Signalen ein in der Praxis häufig vorkommendes Problem dar. Üblicherweise werden fest eingestellte Schwellwerte, gegen die die Daten verglichen werden, für die Zuordnung von Fehlverhalten verwendet. Da dies jedoch bei dynamischen Größen, die Schwankungen und ggf. einem Drift unterliegen, häufig ein unzureichender Klassifikator ist, wurde ein algorithmischer Ansatz, basierend auf einer Prädiktion der Prozessmesswerte und einer anschließenden statistischen Auswertung entwickelt. Ein Kalman-Filter prognostiziert dabei sowohl das Verhalten eines Informations-Systems wie auch auf Basis der Prädiktion die erwarteten Werte der Messungen. Dies erfolgt durch mathematische Modelle. Die Entwicklung des internen Systemverhaltens durch eine Transferfunktion und die Generierung der erwarteten Messwerte aus einem Systemzustand werden auf diese Weise durch eine Überwachungsfunktion realisiert. Die Parameter der Modelle werden anhand von aufgezeichneten Messdaten, wie in Kapitel 5.2 beschrieben, bestimmt. Diese Vorgehensweise bezeichnet man auch als datengesteuert (data-driven). Bei normalen Kalman-Filtern werden die Modelle durch die Markov-Parameter gebildet, die sich jedoch ausschließlich dazu eignen, lineare Dynamiken auszudrücken. Das bringt eine generelle Einschränkung auf lineare Modelle mit sich.
Die Behandlung nicht-linearer Systeme ist durch die Verwendung einer speziellen Kalman-Filter-Variante, dem Unscented Kalman Filter (UKF), möglich. Die mathematische Repräsentation der nicht-linearen Modelle stellt dabei eine Hürde dar, da sie, wie bereits in Kap. 5.2 erwähnt, nicht durch Markov-Parameter abgebildet werden kann. Es existieren aktuell nur wenige statistische Methoden, die aus Messdaten abgeleitet werden können und zugleich Nicht-Linearitäten bzw. nicht-lineares Verhalten dieser Daten korrekt abbilden. Solche Methoden sind beispielsweise Gauß'sche Prozesse (GP) und die Support Vektor Maschinen (SVM).
Für die Darstellung des dynamischen Verhaltens wurde in der vorliegenden Arbeit eine spezielle SVM-Methode, die Least-Squares-Support-Vektor-Maschine, gewählt (s. Kap. 4.9.2). Dieses Konzept erforderte eine Anpassung des Unscen-

ted Kalman-Filters dahingehend, dass die verwendete Transfer- und Überwachungsfunktion mittels Least-Squares-Support-Vektor-Maschinen (LS-SVM) repräsentiert wird. Neben der Abdeckung eines nicht-linearen Verhaltens gestatten solche Modelle mit Hilfe der LS-SVM-Kovarianz die direkte Ermittlung der Rauschparameter, wie Meßrauschen und Prozessrauschen, die im Verlauf des UKF-Algorithmus benötigt werden. Somit ist die Hürde für die mathematische Darstellung überwunden. Allerdings haben speziell bei Datenströmen, die sich über die Zeit verändern, statische, im Voraus berechnete Modelle den Nachteil, dass sie Änderungen der Dynamik, z.B. durch einen Drift, nicht berücksichtigen können.
Eine Option, dies zu kompensieren, besteht im Allgemeinen in der Rückkopplung des Prozesses zur adaptiven Anpassung interner Filterparameter. Im Falle des UKF ist der einzige problemlos anzupassende Parameter die Prozess-Kovarianz Q, die auf adaptive Weise berechnet wird. Dies ist eine Möglichkeit, mit Prozessen, die sich über die Zeit verändern, umzugehen. Im Gegensatz zur sonst üblichen Adaption des Kalman-Filters, die das Modell der Transferfunktion anpasst, wird bei der Kovarianz-Adaption, s. Gleichung 4.38, das Modell nicht modifiziert. Es erfolgt stattdessen eine Anpassung der Kovarianz. Diese beeinflusst indirekt als additive Komponente das Endergebnis der UKF-Berechnung. Auf Grund der indirekten Wirkungsweise besteht die Einschränkung, dass das resultierende adaptive UKF (AUKF) nur innerhalb eines Bereichs auf Prozessentwicklungen reagieren kann. Dieser Bereich ist abhängig vom Wert der Kovarianz. Eine Zusammenfassung der zuvor erklärten Erweiterung wird als LSSVM-AUKF bezeichnet und in Algorithmus 3 vorgestellt.
Eine detaillierte Erläuterung über die UKF-Erweiterungen wird nun basierend auf Algorithmus 3 gegeben: Beim UKF werden, inspiriert durch [69][70], die Transfer- und Überwachungsfunktionen als nichtlineare lernbare Funktionen dargestellt, jedoch wird anstelle von Gauß'schen Prozessen der Least-Squares-Support-Vektor-Maschinen (LS-SVM)-Ansatz verwendet. Der aktuelle Zustand des UKF wird durch ein LS-SVM-Modell, welches als Transferfunktion fungiert, transformiert. Dies ist in Algorithmus 3 Zeile 4 und Gleichung 5.3 dargestellt, wobei u_{k-1} den Eingabewert zum Zeitpunkt $k-1$ und $X_{k-1}^{[i]}$, die $i-te$ Komponente, den internen Systemzustand zum Zeitpunkt $k-1$ definiert. Des Weiteren gibt $\bar{\mathcal{Y}}_k^{[\mathbf{i}]}$ den $i-ten$ Sigmapunkt im Vektor Y an und $LSSVM_{f_\mu}$ bezeichnet die Transferfunktion als LS-SVM-Modell.

$$\bar{\mathcal{Y}}_k^{[\mathbf{i}]} = LSSVM_{f_\mu}(\mathbf{u}_{k-1},\ \mathcal{X}_{k-1}^{[i]}) \tag{5.3}$$

Wie bereits zuvor erwähnt, wird für den Betrieb des UKF sowohl das Prozessrauschen als auch das Messrauschen benötigt. Es empfiehlt sich nicht, das benötigte Prozessrauschen Q (Zeile 6) als Kovarianz der LS-SVM zu berechnen, da sich bei dynamischen Systemen über die Zeit das Rauschen verändern kann. Stattdessen ist es sinnvoll, eine Abschätzung in einer adaptiven Art durchzuführen. Die Idee der Adaption von Q [88] [129] basiert auf einer modifizierten Sage-Husa [116] Schätzfunktion für die Abschätzung des Rauschens. Eine Alternative zu diesem Ansatz wurde in [135] vorgestellt, jedoch muss dabei zur Laufzeit die

Algorithmus 3 LSSVM-AUKF

Eingabe: $\mu_{k-1}; \Sigma_{k-1}; u_{k-1}; z_k; Q_k$:

1: $\mathcal{X}_{k-1}^{n} = (\mu_{k-1} \ , \ \mu_{k-1} + \sqrt{(n+\lambda)\Sigma_{k-1}} \ , \ldots, \ \mu_{k-1} - \sqrt{(n+\lambda)\Sigma_{k-1}} \ , \ldots)$

2: $w_m^0 = \frac{\lambda}{n+\lambda}$
$w_c^0 = \frac{\lambda}{n+\lambda} + (1 - a^2 + \beta)$
$w_m^n = w_c^n = \frac{1}{2(n+\lambda)}$

3: **for** $i = 0 \ldots 2n$: **do**

4: $\bar{\mathcal{Y}}_k^{[\mathbf{i}]} = LSSVM_{f_\mu}(\mathbf{u}_{k-1}, \ \mathcal{X}_{k-1}^{[i]})$

5: $\hat{\mu}_k = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Y}}_k^{[\mathbf{i}]}$

6: $\hat{\Sigma}_k = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Y}}_k^{[\mathbf{i}]} - \hat{\mu}_k)(\bar{\mathcal{Y}}_k^{[\mathbf{i}]} - \hat{\mu}_k)^T + Q_k$

7: $\hat{\mathcal{X}}_k^{[i]} = (\hat{\mu}_k \ , \ \hat{\mu}_k + \sqrt{(n+\lambda)\hat{\Sigma}_k} \ , \ldots, \ \hat{\mu}_k - \sqrt{(n+\lambda)\hat{\Sigma}_k} \ , \ldots)$

8: **end for**

9: **for** $i = 0 \ldots 2n$: **do**

10: $\hat{\mathcal{Z}}_k^{[\mathbf{i}]} = LSSVM_{h_\mu}(\hat{\mathcal{X}}_k^{[\mathbf{i}]})$

11: $R_k = LSSVM_{h_\Sigma}(\hat{\mu}_k)$

12: $\hat{z}_k = \sum_{i=0}^{2n} w_m^{[i]} \hat{\mathcal{Z}}_k^{[\mathbf{i}]}$

13: $S_k = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Z}}_k^{[\mathbf{i}]} - \hat{z}_k)(\bar{\mathcal{Z}}_k^{[\mathbf{i}]} - \hat{z}_k)^T + R_k$

14: $\hat{\Sigma}_k^{\mu,z} = \sum_{i=0}^{2n} w_c^{[i]} (\hat{\mathcal{X}}_k^{[\mathbf{i}]} - \hat{\mu}_k)(\bar{\mathcal{Z}}_k^{[\mathbf{i}]} - \hat{z}_k)^T$

15: **end for**

16: $K_k = \hat{\Sigma}_k^{\mu,z} S_k^{-1}$

17: $\mu_k = \hat{\mu}_k + K_k(z_k - \hat{z}_k)$

18: $\Sigma_k = \hat{\Sigma}_k - K_k S_k K_k^T$

19: $Q_{k+1} = (1 - d_k)\, Q_k + d_k \left[K_k \left(z_k - \hat{z}_k \right) \cdot \left(z_k - \hat{z}_k \right)^T \ K_k^T + \Sigma_k - \hat{\Sigma}_{k-1} \right]$

20: **Ausgabe:** $\mu_k, \Sigma_k, \hat{z_k}$

Ableitung des UKF-Algorithmus gebildet werden, die die Initialisierung, die Erzeugung der Sigma-Punkte, die Propagierung und den Update-Schritt umfasst. Dies stellt einen Mehraufwand bei den UKF-Berechnungen dar im Gegensatz zu dem ausgewählten alternativen Adaptionsschema (Gleichung 5.4), das ausschließlich auf der Filter-Innovation basiert, d.h. der Differenz zwischen dem beobachteten Wert und der optimalen Vorhersage dieses Wertes, der Abweichung zwischen gemessener Ausgangsgröße und deren Schätzung, dem Residuum sowie der neu berechneten Kovarianz und der Kovarianz-Eingabe des vorangegangenen Berechnungsdurchlaufs. Des Weiteren ist ein Vergessensfaktor (d_k) enthalten, der historische Ergebnisse schwächer und aktuelle stärker berücksichtigt. Dies wird in Form einer Gewichtungsfunktion zwischen dem alten Rauschwert

und den neuen Schätzwerten des Prozessrauschen und der Kalman-Verstärkung realisiert.

$$\begin{aligned} Q_{k+1} = (1 - d_k)\, Q_k + \\ d_k \left[K_k \left(z_k - \hat{z}_k \right) \cdot \left(z_k - \hat{z}_k \right)^T \right. \\ \left. K_k^T + \Sigma_k - \hat{\Sigma}_{k-1} \right] \end{aligned} \tag{5.4}$$

Die Überwachungsfunktion h, von der aus dem aktuellen Zustand die Schätzung des Messwertes gebildet wird, ist ebenfalls als LS-SVM (Zeile 12) ausgeführt. $LSSVM_{h_\mu}$ kann als LS-SVM-Modell der Überwachungsfunktion angesehen werden. Hierbei gibt der Index beim LS-SVM-Modell ($LSSVM_{h_\mu}$) an, dass sich daraus der Mittelwert ergibt.

$$\hat{\mathcal{Z}}_k^{[\mathbf{i}]} = LSSVM_{h_\mu}(\mathcal{X}_k^{[i]}) \tag{5.5}$$

Das vom UKF benötigte Messrauschen R (Zeile 13) wird mit Hilfe der LS-SVM bestimmt. Anstelle der Verwendung eines weiteren LS-SVM-Modells wird die Kovarianz (Σ) des bereits bestehenden Modells $LSSVM_h$ genutzt (siehe Zeile 13 aus Algorithmus 3). Im Gegensatz zu GP bieten SVMs (und auch die LS-SVMs) nicht die Option, die Kovarianz direkt aus dem Modell abzuschätzen. Bei der LS-SVM besteht jedoch die Möglichkeit, diese online (zur Laufzeit) zu berechnen. Dies kann auf der Basis des Überwachungsmodells $LSSVM_{h_\mu}$, geschehen, dafür sind jedoch zeitintensive Berechnungen notwendig. Alternativ und deutlich schneller ist dies mittels eines vorberechneten LS-SVM-Modells ($LSSVM_{h_\Sigma}$), wie in Gleichung 5.6 dargestellt, möglich.

$$R_k = LSSVM_{h_\Sigma}(\hat{\mu}_k) \tag{5.6}$$

Das LSSVM-AUKF aus Algorithmus 3 bildet die Grundlage für einen statistischen Fehlerprädiktor [45]. Der Fehlerprädiktor führt die statistische Auswertung basierend auf den Residuen des Kalman-Filters durch und nicht, wie man vermuten würde, auf der Basis der Prädiktionsergebnisse und Beobachtungswerte (Messwerte). Dies ermöglicht eine einfache Gruppenbildung für die Unterscheidung von normal und anormal.
Während des Betriebes kann die Wahrscheinlichkeit eines beobachteten Datenpunktes sehr einfach auf der Grundlage der vorhergesagten Wahrscheinlichkeitsverteilung, welche durch den Kalman-Filter (LSSVM-AUKF-Algorithmus) bereitgestellt wird, berechnet werden. Dies ist in Gleichung 5.7 dargestellt.

$$p\left(y_k | y_{k-1}, \ldots, y_0\right) = \frac{1}{\sqrt{2\pi\sigma_{k|k-1}^2}} \exp\left(\frac{\hat{e}_{k|k-1}^2}{\sigma_{k|k-1}^2}\right) \tag{5.7}$$

Eine robustere und zuverlässigere Berechnung für die Wahrscheinlichkeit der aktuellen Schätzung wird durch einen gewichteten Durchschnitt der zuvor ermittelten Wahrscheinlichkeiten (siehe Gleichung 5.8) erreicht. Hierbei ist w ein

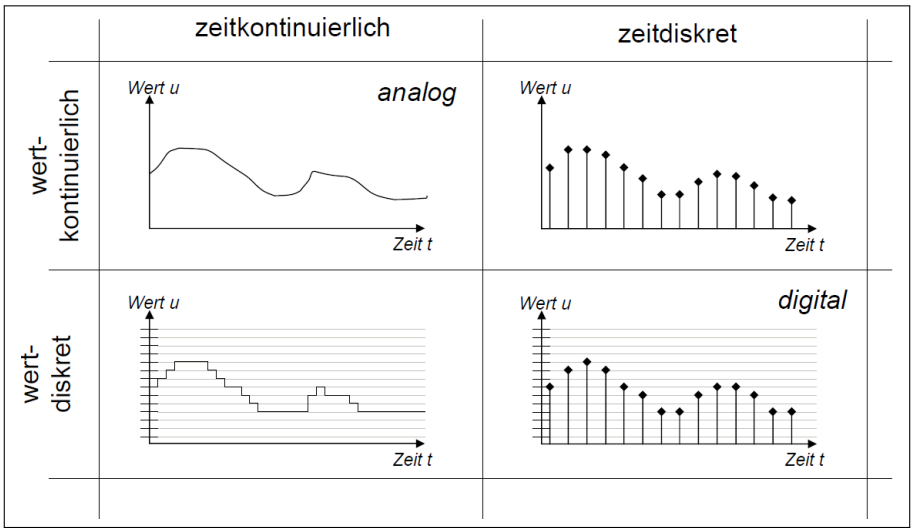


Abbildung 5.3: Signalformen (Entnommen aus[124])

Glättungsfaktor und z_{k-1} ist die Wahrscheinlichkeit aller vergangener Messungen.

$$z_k = w_k z_{k-1} + (1 - w_k) \ln p\,(y_k) \tag{5.8}$$

5.4 Modellierung der kontinuierlichen Variablendomäne für Bayes'sche Netze

Die in den vorangegangenen Unterkapiteln vorgestellten Ansätze deckten ausschließlich diskrete Variablen ab. In vielen Anwendungen sind jedoch auch kontinuierliche Signale vorhanden. Häufig kommen zu deren Verarbeitung Methoden der Signalanalyse bzw. Signalverarbeitung zum Einsatz, die meist eine Digitalisierung bzw. Diskretisierung des Signals fordern. Methoden zur Diskretisierung eines kontinuierlichen Signals können in die folgenden vier Kategorien, siehe auch Grafik 5.3, unterteilt werden:

- zeitkontinuierliche und wertkontinuierliche
- zeitdiskrete und wertkontinuierliche
- zeitkontinuierliche und wertdiskrete
- zeitdiskrete und wertdiskrete

Die zuvor angesprochenen Diskretisierungsmethoden sind für den Einsatz bei Bayes'schen Netzen nur bedingt verwendbar, da nicht ausgeschlossen werden kann, dass einzelne Werte fehlen. Dies impliziert, dass nicht alle vom BN benötigten

Wahrscheinlichkeiten vorhanden sind. Nachfolgend wird deshalb eine alternative Diskretisierungsmethode für den Einsatz bei BN vorgestellt.

5.4.1 Diskretisierung mit Hilfe von trunkierten (beschränkten) Basisfunktionen

Die Diskretisierung von kontinuierlichen Variablen ist mit zusammengesetzten, beschränkten Exponentialfunktionen (MTE) möglich. Der MTE-Ansatz [95] stellt die Wahrscheinlichkeitsdichte-Verteilung der Variablen-Domäne dar und kann als eine generalisierte Form der Diskretisierung angesehen werden, bei der die Dichte der Zielvariablen durch zusammengesetzte uniforme Verteilungen abgeschätzt wird.
Zur Anwendung des MTE-Ansatzes muss die Domäne der kontinuierlichen Variablen, die mit Ω_x bezeichnet wird, in eine Menge von Partitionen $\Omega_1, \ldots, \Omega_n$ aufgeteilt werden. Die Partitionen der Domäne sind zusammenhängend angeordnet und können daher als Verteilungen [81] auf Hyperwürfeln [36] dargestellt werden. Die Darstellung eines solchen Hyperwürfels kann mittels einer MTE, bestehend aus n-Komponenten $f : \Omega_x \mapsto \Re^+$ erfolgen. Das MTE-Potential (f) hat folgende Komponenten:

$$f(x) = a_0 + \sum_{i=1}^{m} a_i \exp\left\{ \sum_{j=1}^{n} b_i^{(j)} x_j \right\} \tag{5.9}$$

Durch die Partitionierung $\Omega_1, \ldots, \Omega_n$ wird sichergestellt, dass die Domäne der kontinuierlichen Variablen X in Hyperwürfel aufgeteilt ist, so dass f als $f(x) = f_i(x)$ definiert ist, wenn $x \in \Omega_i$ ist, wobei jedes f_i, $i = 1, 2, \ldots, n$, der Gleichung 5.9 entspricht. Jeder Hyperwürfel muss einigen Rahmenbedingungen gerecht werden, die die Aufteilung der Domäne in Intervalle Ω_x, einschränken. Dies betrifft die Auswahl der Partitionen Ω_x in der Form, dass die Wahrscheinlichkeitsdichtefunktion, die abgeschätzt werden soll, keine Veränderungen von Konkavität zu Konvexität oder Zunahme zu Abnahme aufweist. Bedingt wird dies durch die zu Grunde liegenden exponentiellen Funktionen exp(x), die konkav sind und innerhalb ihrer Domäne ansteigen. Eine detaillierte Beschreibung des Domänen-Partitionierungsprozesses wird in [112] gegeben. Diese zuvor benannten Anforderungen schränken die Anwendung von MTE in der Praxis dahingehend ein, dass Dichtefunktionen nicht oder nur unter großem Aufwand abbildbar sind.
Eine Generalisierung des MTE-Ansatzes ist eine Mischung aus beschränkten Basis-Funktionen (MoTBF) [82], bei denen die abstrakte Notation von Basisfunktionen $(\psi(\cdot))$ genutzt wird. Bei MoTBF finden u.a. exponentielle und polynomiale Funktionen Anwendung. Ein Vorteil bei der Anwendung von MoTBF gegenüber MTE ist, dass speziell bei der Abschätzung von univariaten Verteilungen die Domäne nicht in Intervalle aufgeteilt werden muss, da die komplette Domäne $\Omega_x \subseteq \mathrm{R}$ von einer Menge bestehend aus Basisfunktionen abgedeckt wird. Entsprechend wird die Funktion $g_k : \Omega_X \mapsto \mathbb{R}_0^+$, welche die Domäne als

eine Menge von beschränkten Basis-Funktionen (MoTBF) darstellt, s. Gleichung 5.10 definiert. In dieser Gleichung stellt a_i eine Konstante (reelle Zahl) dar, wobei der Index $i = 0, ...m$ das Intervall kennzeichnet.

$$g_k(x) = \sum_{i=0}^{k} a_i \psi_i(x) \tag{5.10}$$

Eine Darstellung von kontinuierlichen Variablen als MoTBF ermöglicht beispielsweise die Verarbeitung solcher Variablen durch das in Kap. 4.2 vorgestellte Lernverfahren. In der Praxis kann die Bestimmung von MoTBF aus aufgezeichneten Daten erfolgen. Eine detaillierte Beschreibung wird in [83] gegeben. Des Weiteren wird in [81] gezeigt, dass eine Wahrscheinlichkeitsdichtefunktion (PDF) einer kontinuierlichen Variablen, die in der o.a. Weise abgeschätzt wird, deutlich genauere Ergebnisse erzielt als eine PDF, die mit herkömmlichen Methoden bestimmt wird.

5.4.2 Domänen-Partitionierung

In praktischen Anwendungen kommen vermehrt kontinuierliche Variablen vor. Eine Diskretisierung der Variablen findet in vielen praktischen Anwendungen Einsatz. Bei Bayes'schen Netzen ist die Besonderheit zu beachten, dass nur bei der Diskretisierung vorhandene Instanzen während des Betriebes behandelt werden können. Des Weiteren muss je nach Granularität des Datensatzes eine größere Menge an Daten, z.B. die Übergangswahrscheinlichkeit zwischen den Instanzen, bekannt sein und vorgehalten werden. Im Folgenden wird eine Methode vorgestellt, die sowohl mit unbekannten Instanzen umgehen wie auch den Datenbestand der Wahrscheinlichkeiten reduzieren kann und somit handhabbar macht. Diese Lösung ist auch bei Bayes'schen Netzen, die Zeitaspekte berücksichtigen, anwendbar.
Im Rahmen der Verarbeitung werden die kontinuierlichen Variablen als Dichtekurve über den Wertebereich der Variablen abgebildet, so dass sich eine empirische Schätzung einer Wahrscheinlichkeitsverteilung, ein Histogramm, ergibt. Auf der Basis der Hüllkurve, die sich über die Histogrammbalken erstreckt, erfolgt die weitere Verarbeitung. Diese wird in charakteristische Regionen partitioniert, wie beispielsweise in Zonen mit hoher respektive niedriger Dichte und ist vergleichbar mit einer Diskretisierung. Da jedoch die Aufteilung der Hüllkurve und nicht die der eigentlichen Daten erfolgt, ist die Granularität üblicherweise geringer, d.h. es existieren somit weniger Instanzen.
Für die Diskretisierung von kontinuierlichen Variablen haben sich bereits einige Algorithmen [35] [71] etabliert. Die wenigsten davon berücksichtigen jedoch die Beziehung zwischen benachbarten Elementen, den Instanzen einer Variablen. Diese Beziehung kann u.a. durch die Ähnlichkeit in der Dichteverteilung, d.h. die Abweichung der Werte, ausgedrückt werden und gestattet eine Weiterverarbeitung, z.B. durch Methoden des Data-Mining.
Ein etabliertes Verfahren des Data-Mining ist die Gruppierung ähnlicher Elemente. Neben der häufig verwendeten Clusteranalyse kommt auch das Binning,

eine weitere Darstellungsart der Quantisierung, zum Einsatz. Dabei werden die Originalwerte, die in ein vorgegebenes Intervall, ein Bin (Gebinde), fallen, durch den repräsentativen Wert des Intervalls, häufig den zentralen Wert (z.B. Median bzw. Mittelwert) ersetzt. Üblicherweise findet man beim Binning identische und feste Intervallbreiten (Equal Width Binning) vor, d.h. die Daten werden in Intervalle gleicher Breite aufgeteilt. Des Weiteren sind auch gleiche Datenmengen innerhalb des Intervalls (Equal Frequency Binning) üblich, d.h. jedes Intervall beinhaltet die gleiche Anzahl an Elementen. In beiden Fällen ist es schwierig, charakteristische Eigenschaften der Daten, wie z.B. Ähnlichkeiten in der Verteilung, abzudecken. Diesem Manko kann durch ein Binning mit variabler Breite entgegengewirkt werden.
Für die Repräsentation kontinuierlicher Variablen wurde, wie bereits oben erwähnt, die Darstellung des Histogramms gewählt. Mittels der Hüllkurve und einem Binning mit variabler Weite werden die charakteristischen Merkmale der Variablen abgedeckt. Dieser Ansatz liefert die repräsentativen Intervalle einer Dichtefunktion sowie deren Grenzen über Regionen ähnlicher Dichte. Die Charakteristik der Kurve wird somit durch Bins variabler Breite, bestehend aus benachbarten Elementen des Wertebereichs repräsentiert. Die Gruppierung ähnlicher Elemente erfolgt meist mittels Distanz- oder Ähnlichkeitsmaßen. Für die Berechnung der Ähnlichkeit existieren mehrere Möglichkeiten, z.B. Distanz-Messungen, statistische Tests oder prozentuale Abweichungen.
Im vorliegenden Fall fand eine Distanz-Messung, die sogenannte Minkowski-Distanz (Gleichung 5.11), als Metrik für die Gruppierung von benachbarten Elementen Anwendung.

$$dist = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} \tag{5.11}$$

(p gibt die Ordnung des zu Grunde liegenden Vektorraumes an.)

Mit der Gleichung 5.11 wird die Distanz von benachbarten Elementen der Histogrammkurve berechnet. Diese stehen bekanntlich in direkter Relation zu den Werten der entsprechenden Variablen. Die Minkowski-Distanz wird mit einem Schwellwert verglichen, um festzustellen, ob diese zu dem aktuellen Bin gehört oder eine neue erstellt werden muss. Die Auswahl und Anpassung des Minkowski-Distanz-Schwellwertes hängt im Allgemeinen sowohl von den Daten ab als auch von dem Ziel, das erreicht werden soll, wie beispielsweise einer Reduktion von Daten. Die so erstellten Bins decken jeweils einen individuellen Bereich mit klar definierten Grenzen ab.
Eine Identifizierung der Bins ist durch eine Ordnungsnummer gewährleistet, die eine eindeutige Zuordnung ohne Detailkenntnis der Intervallgrenzen bzw. des spezifischen Wertes gestattet und eine maschinelle Weiterverarbeitung ermöglicht.
Der bisher angeführte Ansatz eliminiert keine Werte, wie dies beispielsweise bei anderen Ansätzen vorkommt, sondern fasst diese nur zusammen. Dadurch können auch Regionen der Kurve, die eine niedrigere Dichte darstellen, abgedeckt werden. Diese Zonen werden häufig nicht berücksichtigt, rücken aber speziell bei der Systemmodellierung, bei der explizit Fehler enthalten sein sollen, in den Fokus. Beachtenswert sind besonders Regionen hoher Dichte, da hier meist

eine feinere Granularität gefordert wird. Für die Systemmodellierung wird bei der Diskretisierung der Daten die Abdeckung von Regionen mit hoher wie auch niedriger Dichte benötigt.

5.5 Initiale Wahrscheinlichkeit für Bayes'sches Netze mit kontinuierlicher Zeit

Durch die in Kapitel 5.4.2 beschriebene Art der Diskretisierung lässt sich prinzipiell der Ansatz zur Vorhersage von Systemausfällen anwenden. Sollen Zeitcharakteristika bei der Anwendung abgedeckt werden, kann dies u.a. mit einer Erweiterung der Bayes'schen Netze, den Continuous Time Bayesian Networks (CTBN), erfolgen. Dieser Formalismus benötigt jedoch eine initiale Wahrscheinlichkeitsverteilung für jeden Wert einer Variablen.
Eine wie in Kapitel 5.4.2 angesprochene Möglichkeit der Wahrscheinlichkeitsbestimmung ist durch Histogramme gegeben, die von Messdaten abgeleitet werden. Allerdings sind empirische Histogramme ein schlechtes Modell [112] für die Dichterepräsentation von Daten, denn wenn Daten spärlich bzw. dünn vorhanden sind, wirkt sich das in einer groben Verteilung bzw. unebenen Kurvenform aus. In solchen Situationen wird eine bessere und glattere Annäherung durch die Anwendung von Kernel-Dichte-Schätzern [27] erreicht. Bei CTBN wird eine vollständige Dichteschätzung für die Bestimmung der initialen Wahrscheinlichkeiten benötigt, um die initiale Wahrscheinlichkeit für die Ausprägungen (Instanzen) berechnen zu können. Zur Erzielung einer hohen Abdeckung in der kontinuierlichen Domäne wird daher der MoTBF-Ansatz für die Bestimmung der Wahrscheinlichkeits-Dichte-Funktion (PDF) benutzt.
Eine weitere Anforderung an die mathematische Darstellung ist die Normierung der Fläche unter der Wahrscheinlichkeitsdichtekurve, um Korrekturberechnungen zu vermeiden. Der MoTBF-Ansatz erfüllt die notwendige Voraussetzung, so dass das Integral der kompletten Domäne (Ω) über eine MoTBF-Funktion $g_k(x)$ 1 wird (Gleichung 5.12).

$$\int_{\Omega} g_k(x)\,dx = 1 \tag{5.12}$$

Diese Normierung ermöglicht partielle Integrationen über eine PDF und resultiert in Wahrscheinlichkeiten für den entsprechenden Bereich. In der Folge kann die Berechnung der Wahrscheinlichkeit eines jeden Bins vorgenommen werden. Sie wird erreicht, indem die Grenzen der beim Binning berechneten Intervalle als Intervallgrenzen $(\mu, \mu + \Delta)$ verwendet werden. Dies wird ergänzend in der Gleichung 5.13 veranschaulicht.

$$P_x^0 = \int_{\mu}^{\mu+\Delta} g_k(x)\,dx \tag{5.13}$$

Das Binning und die Bestimmung der initialen Wahrscheinlichkeiten ermöglichen die Verarbeitung von kontinuierlichen Variablen mit Continuous Time Bayesian Networks, da alle erforderlichen Spezifikationen für CTBN erfüllt werden. Die Verarbeitung der kontinuierlichen Variablen in einer diskreten Art ist nun bei CTBN mit etablierten Methoden sowohl für den Lernvorgang als auch zur Laufzeit durchführbar.

5.6 Fehlermodellierung

Die Abbildung bzw. mathematische Modellierung von Fehlern stellt bei der Vorhersage von Systemausfällen eine wesentliche Grundlage dar. In der vorliegenden Arbeit werden für die Systemmodellierung CTBN verwendet, die für die maschinelle Weiterverarbeitung eine numerische Darstellung erfordern.
Fehler können in unterschiedlichen Charakteristika und Ausprägungen auftreten und kommen häufig bei kontinuierlichen und diskreten Daten vor. Sie werden aber auch in Textform im Rahmen von (System-) Events signalisiert. Des Weiteren können Fehler auch in Datenströmen auftreten. Hier sind sie als Anomalien enthalten und werden meist in Form von Mustern identifiziert. Die o.a. Vielfalt der Signalformen lässt sich jedoch selten komplett mit herkömmlichen Ansätzen modellieren.
Eine Möglichkeit, Fehler bei CTBN darzustellen, ist die Einführung von Hilfsvariablen. Diese können eine Vielzahl von Zuständen darstellen, unabhängig von der Art oder Signalform. Des Weiteren können auch Vorkenntnisse von bestimmten bekannten Fehlern, Event-Signalen, anomalen Zuständen oder Anomalien in den Daten abgebildet werden. Im Allgemeinen haben diese Variablen einen binären Charakter, jedoch kann dies im Falle einer Abbildung von höherwertigen Zuständen abweichen. Bei einer binären Darstellung wird das Fehlverhalten durch Signalisierung der Hilfsvariablen (Flagging) abgebildet (siehe Bild 5.4). Dieses Flagging wird nur zu eindeutigen Zeitpunkten durchgeführt, genau dann, wenn der entsprechende Vorfall auftritt. Dieser Mechanismus signalisiert ein Fehlverhalten, das durch den angewandten Ansatz in den Datenbestand eingegliedert wird. Besonders bei CTBN umfasst die Integration die Berücksichtigung der korrespondierenden Zeitinformationen. In der Praxis resultiert dies in einer Erweiterung des Datenbestandes, d.h. der Logdatei der aufgezeichneten Messwerte.

5.7 Modellierungsansatz für Bayes'sche Netze mit kontinuierlicher Zeit

Für die Verlässlichkeitsmodellierung von Informations-Systemen auf der Grundlage von Messdaten wurde ein algorithmisches Konzept entwickelt. Das Verfahren lässt sich generell in drei Phasen gliedern:

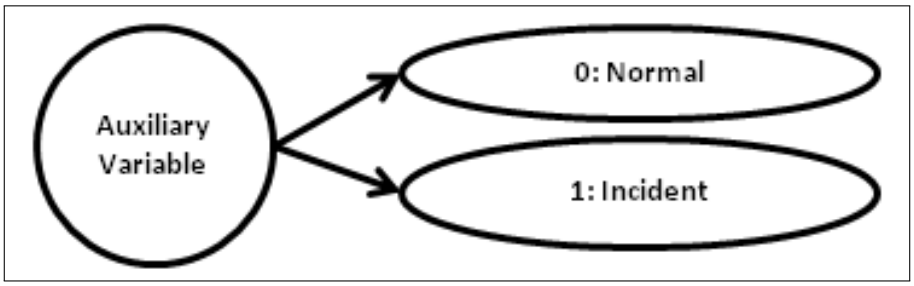


Abbildung 5.4: Hilfsvariable für die Signalisierung von Ereignissen

- Im ersten Schritt wird eine Diskretisierung der kontinuierlichen Variablen durchgeführt. Des Weiteren werden Hilfsvariablen in den Datenbestand eingefügt, die bekanntes Fehlverhalten bzw. Fehler, Events oder kritische Zustände repräsentieren (s. Kapitel 5.6).
- Der zweite Schritt dient zur Ermittlung der Struktur eines Bayes'schen Netzwerks. Dabei ist besonders darauf zu achten, dass im Lern-Algorithmus eine Option vorhanden ist, mit der Bedingungen (Constraints) integriert werden können. Solche Bedingungen sind beispielsweise potentiell aufgetretenes Fehlverhalten bzw. Fehler, welche in den zu lernenden Daten signalisiert werden und in direkter Beziehung zu einer in den Daten (Messwerten) enthaltenen Variablen stehen.
- Im dritten Schritt wird die für den CTBN-Lernvorgang benötigte initiale Wahrscheinlichkeit bestimmt, mit der als Basis anschließend die CTBN-Parameter in Form von Intensitäten ermittelt werden, z.B. für das Verlassen eines Zustands oder den Zustandswechsel. Mit Hilfe des so ermittelten CTBNs können Schlußfolgerungen über das Verhalten des zu Grunde liegenden Systems getätigt werden.

Nachfolgend wird eine detaillierte Beschreibung der Methode vorgestellt:

* In der ersten Phase wird eine Daten-Konditionierung der kontinuierlichen Variablen durchgeführt, s. Kapitel 5.4.2. Die dabei durchgeführte Diskretisierung wird, s. Kapitel 5.4, mit einem kleinen Minkowski-Distanz-Schwellwert, und somit mit geringer Datenreduktion auf Basis der Messdaten vorgenommen, um einen möglichst hochwertigen Datenbestand und keine Ungenauigkeiten in Form von diffusen Daten zu erreichen. In diesem Schritt werden die Zeitinformationen des Messdatensatzes ignoriert und die Hilfsvariablen in den Datenbestand eingefügt.

* Im zweiten Schritt wird basierend auf dem erweiterten Datensatz die Struktur eines Bayes'schen-Netzwerks (BN) explorierend ermittelt, indem Relationen zwischen Variablen statistisch ausgewertet werden. Zum Lernen der Struktur des BN wird ein etablierter Lernalgorithmus, der in Kapitel 4.2 vorgestellte MMHC-Algorithmus [144], angewandt. Dieses Verfah-

ren bietet folgende Vorteile: Hybride Algorithmen liefern im Allgemeinen bessere Ergebnisse als State-of-the-art Algorithmen-Vertreter. Darüber hinaus bietet MMHC die Option, vorhandenes Domänenwissen in Form von Bedingungen (Constraints) zu berücksichtigen. Das ist vor allem dann nützlich, wenn Quelle bzw. Ursache eines Fehlers bekannt sind. Des Weiteren lassen sich Beziehungen zwischen Hilfsvariablen und beobachteten, gemessenen Systemvariablen (Zufallsvariablen) zur Modellierung expliziter Abhängigkeiten darstellen. Solche bekannten Relationen werden über Constraints abgebildet.

* Ein BN-Struktur-Modell, das auf diese Art und Weise generiert wurde, bildet die Grundlage der dritten und letzten Phase, in der die CTBN-Zeitcharakteristika gelernt werden. In diesem Schritt müssen Zeitinformationen (wie beispielsweise Zeitstempel) im Datensatz ausgewertet werden. In vielen praktischen Anwendungen stehen für die Durchführung weiterer Berechnungen nur moderate Hardwareressourcen zur Verfügung. Je nach Messdatenstruktur werden diese voll ausgeschöpft bzw. reichen diese, z.B. für die Ermittlung der CTBN Intensitäten, nicht aus. Daher ist es erforderlich, die aufgezeichneten Messdaten mit einem höheren Grad, d.h. mit einem größeren Minkowski-Distanz-Schwellwert als im ersten Schritt, zu diskretisieren. Dies impliziert eine höhere Datenreduktion, die nicht die Anzahl der Messwerte im Datensatz, sondern lediglich die Menge der Instanzen pro Variable verringert. Die im Originaldatensatz vorhandenen und zum Messwert korrespondierenden Zeitinformationen bleiben unverändert. Hier muss angemerkt werden, dass der Grad der Reduktion Auswirkungen auf die Präzision des zu lernenden Modells hat, d.h. je höher der Grad der Reduktion desto geringer die Präzision.
  Der durch die Datenreduktion verringerte Speicherbedarf lässt sich wie folgt darlegen: Die Intensitäten (CIM) werden als Tensor vom Grad drei dargestellt, wobei der Tensor als eine Menge von Intensitätsmatrizen aufgefasst werden kann. Die erste und die zweite Dimension des Tensors werden von der Anzahl der Elemente der entsprechenden Variablen ($Card(X_i)$) vorgegeben. Die Ausdehnung in die dritte Dimension wird von der Anzahl der Instanzen der Variablen, die die jeweilige Variable beeinflussen (konditionieren), durch $\prod_j Card(PA_j(X_i))$ vorgegeben. Besonders bei der Anwendung von hoch granularen Daten führt dies zu riesigen Zustandsräumen und resultiert in einem enormen Speicherbedarf durch die Intensitätsmatrizen-Darstellung. Ein Ausweg zur Vermeidung eines solch hohen Speicherbedarfs ist die erneute Ausführung des in Kapitel 5.4 beschriebenen Verfahrens zur Reduktion der Original-Daten in einem deutlich höheren Niveau.

Auf der erzeugten diskretisierten Datenbasis werden nun etablierte Lern-Algorithmen für die Bestimmung der Continuous Time Bayesian Network (CTBN)-Parameter angewandt. Diese Intensitätsmatrizen decken den dynamischen Teil des CTBN ab. Nach deren Ermittlung ist es möglich, Annahmen über das Zeitverhalten des Systems zu treffen. Ein CTBN-Modell, welches aus einem redu-

zierten Datensatz abgeleitet wurde, ermöglicht die Ableitung von Prognosen des Systemverhaltens. Dies kann auf unterschiedliche Weise erfolgen: Entweder Offline durch Anwendung analytischer Methoden auf das Modell oder Online, wobei Betriebsdaten (Messwerte) als Eingabewerte des Modells verwendet werden.

5.8 Probabilistisches Model-Checking von Informationssystemen

Die Basis des Ansatzes bildet ein CTBN-Modell, welches wie in Kapitel 5.7 beschrieben generiert wurde. Vor der Modell-Generierung ist zu prüfen, ob alle im Datensatz vorhandenen Variablen benötigt werden und ob ggf. Redundanzen vorhanden sind. Eine Eliminierung von entbehrlichen Variablen führt zu einer Verringerung des Gesamt-Zustandsbereichs. Dies reduziert nicht nur den Speicherbedarf, sondern auch die Rechenzeit. Anschließend findet die Amalgamation auf das CTBN-Modell Anwendung.
Hierbei werden die faktorisierten Markov-Prozesse, von denen üblicherweise einer für jede Variable existiert, vereinigt. In diesem Schritt hängt die Größe jedes Markov-Subprozesses von der Anzahl der ihn beeinflussenden Variablen und deren jeweiliger Menge an Instanzen ab.
Der erste Schritt ist die Konstruktion eines vollständigen Markov-Prozesses aus einem CTBN-Modell durch eine verbesserte Version der Amalgamation, s. Kapitel 4.3.3. Damit wird eine Markov-Kette, die den kompletten Zustandsraum des Systems abdeckt, berechnet. Bei praktischen Anwendungen ist zu beachten, dass dieser Zustandsraum sehr groß werden kann und unweigerlich zu einem sehr hohen Speicherbedarf führt. Zur Repräsentation im Speicher empfiehlt sich die Darstellung als Sparse-Matrix, die nur Elemente ungleich Null abspeichert und somit die Anforderungen an den Speicherbedarf geringer ausfallen lässt.
Die Auswertung des Gesamtmodells kann unter verschiedenen Aspekten erfolgen. Neben der Verhaltensvorhersage von Systemen im laufenden Betrieb (Online) wird das Modell losgelöst von Betriebsdaten (Offline) ausgewertet. Für solche Analysen existieren neben klassischen Analysetechniken auch automatisierte Verfahren für die Modellauswertung. Hierzu zählt unter anderem das Model-Checking.
Im Fall des vereinigten CTBN-Modells wird der vollständige Markov-Prozess durch eine Continuous Time Markov Chain (CTMC) repräsentiert. Aus diesem Grund kommt eine spezielle Erweiterung des Model-Checking, das probabilistische Model-Checking, zum Einsatz, welches umfangreichere Analysefunktionen für Markovketten mit Zeiteigenschaften ermöglicht. Im Gegensatz zum normalen Model-Checking wertet das probabilistische Model-Checking nicht nur das Vorhandensein von Übergängen zwischen den Zuständen aus, sondern berücksichtigt die Wahrscheinlichkeit bzw. Rate der Transitionen. Diese Erweiterung gestattet es, einen Pfad im Modell über die Zustände unter Berücksichtigung der Wahrscheinlichkeiten zu ermitteln. Häufig wird hier der Pfad mit der höchsten Wahrscheinlichkeit gesucht. Des Weiteren sind CTMC-Analysen

hinsichtlich des Zeitverhaltens durchführbar. Beispielsweise ist dabei die Zeit ermittelbar, in der das System in einem stabilen Zustand verharrt bzw. bis es in einen Fehlerzustand übergeht.
Eine solche Funktion bietet auch unter dem Gesichtspunkt der Verlässlichkeit von Systemen Potential, Annahmen über den wahrscheinlichsten Pfad zu einem Fehler und die Zeit bis zum Auftreten eines Fehlers (Ausfall) zu bestimmen. Außerdem können Abschätzungen über bevorstehende Fehler in zuvor definierten Zeitintervallen berechnet werden.
Für die bisher vorgestellten Analysen der CTMC sind etablierte Software-Werkzeuge im Bereich des probabilistischen Model-Checkings allgemein verfügbar. Im weiteren Verlauf der Arbeit wird die Model-Checking-Software MRMC [65] eingesetzt. Bei diesem Model-Checker müssen die zu überprüfenden Eigenschaften mit Logik-Formeln ausgedrückt werden; dabei kommen PCTL- und CSL-Formeln zum Einsatz. Wie bereits in Kapitel 4.5.1 erwähnt, lassen sich damit zustandsbasierte oder pfadbasierte Eigenschaften spezifizieren. Speziell bei der Fehlerprädiktion können entsprechend aufgestellte Formeln für die Analyse von Fehlverhalten genutzt werden, wie beispielsweise Pfad bzw. Zeit zu einem bekannten Fehler.
Auf der Basis der exportierten Modell-Daten lassen sich Analysen hinsichtlich bekannter Fehler bzw. Vorfälle in Form von Fehlverhalten durchführen. Besonders bei Fehlerprädiktionsumgebungen lassen sich mit individuellen, modellspezifischen Formeln sowohl die Pfade mit ihrer entsprechenden Wahrscheinlichkeit zu einem Fehler erkunden und ausgeben, d.h. wie sich das System in den Fehlerzustand entwickelt, als auch Zeitanalysen durchzuführen, z.B. die Zeit bis zu einem Wechsel in einen Fehlerzustand.

Kapitel 6

Evaluierung

In diesem Kapitel werden die in Kapitel 5 vorgestellten Methoden sowohl zur Fehlerinstrumentierung mit Hilfe von Kontrollgruppen (CGroups) als auch zur statistischen Anomalie-Erkennung auf der Basis eines Kalman-Filters und Systemmodellierung mittels Bayes'scher Netze angewandt. Des Weiteren werden die erstellten Bayes'schen Netzmodelle unter Zuhilfenahme von Model-Checking evaluiert.

6.1 Fehler-Instrumentierung mit Kontrollgruppen (CGroups)

6.1.1 Implementierung

Zum Zweck der Evaluierung der in Kapitel 5.1 vorgestellten CGFail-Methode [44] war es erforderlich, die Methode in Software umzusetzen. Dazu wurde CGFail in eine LINUX-Umgebung implementiert.
Bei der Softwareentwicklung werden üblicherweise Bibliotheken verwendet, die wie im vorliegenden Fall vom Betriebssystem abhängen. Es handelt sich beispielsweise um Bibliotheken, die Zeit-Funktionalitäten beinhalten, wie z.B. Timer, Ereignis-Behandlung /-Benachrichtigung (Events) oder Funktionen zum Einlesen und Analysieren von Daten (Parser). Da im vorliegenden Fall diverse betriebssystemkern-nahe Funktionen (LINUX-Kernel) genutzt werden mussten, war es erforderlich, die Software in der Programmiersprache C++ zu entwickeln. Hinzu kam, dass für die Nutzung der Kontrollgruppen (CGroups) [131], die von LINUX bereitgestellte API-Bibliothek (libcg) (s. Kap. 5.1.1) erforderlich war. Diese Kontrollgruppen (CGroups) sind in den Betriebssystemkern integrierte Eigenschaften zur Verwaltung, Beschränkung und Isolation von Ressourcen. Die C++ Implementierung begünstigt die Umsetzung und Integration von dynamischen Lastgeneratoren, welche CPU- und Arbeitsspeicher-Last (I/O) erzeugen, da auch hierbei direkt betriebssystemnahe Funktionen genutzt werden können. Die Nutzung betriebssystemkernnaher (LINUX-Kernel) Funktionen, z.B. Er-

eignisschleifen / Timer, die eine präzise Zeitsteuerung von Programmabläufen ermöglichen, sind für eine zeitlich deterministische und reproduzierbare Programmausführung essentiell. Solche Funktionen, die mit z.B. als Datei abgelegten vordefinierten Zeitprofilen gekoppelt sind, gestatten eine Wiederholbarkeit der Ausführung mit identischer zeitlicher Steuerung. Durch die Daten der Zeitprofile und auch durch die Definition von Ressourcenobjekten, die jeweils eine individuelle Ressource inkl. deren Auslastung / Nutzung, wie CPU-Last und Arbeitsspeicher-Nutzung beschreiben, kann sowohl eine Ansteuerung von Ereignissen innerhalb des Programms wie auch die Beeinflussung externer Komponenten erfolgen. Dies impliziert, dass zu diesem Zweck die Ressourcenobjekte mit zugehörigen Parametern als Profil, ähnlich der Zeitprofile, gespeichert werden. Die Kombination der o.a. Profile zu einem Anwendungsprofil, das im Wesentlichen die CGroup-Parameter, aber auch die Daten der Lastgeneratoren und ggf. die Parameter der zu testenden Applikation enthält, gestattet sowohl die Steuerung von Programmabläufen mittels Ressourcenobjekten wie auch das gezielte Betätigen von Lastgeneratoren. In einem späteren Abschnitt werden sowohl die Parameter (Daten) als auch die Struktur des Profils detailliert erläutert.
Im Rahmen der Entwicklung der Software CGFail wurde im Hinblick auf eine persistente Speicherung der Profildaten ein für den Anwender gut lesbares und verständliches Format verwendet. Die Wahl fiel dabei auf das XML-Format. Dieses Format ermöglicht neben einer einfachen manuellen Erstellung der Profile eine unkomplizierte maschinelle Verarbeitung. Zum Einlesen und Analysieren von XML-Dateien kommt üblicherweise eine spezielle Software, der Parser, zum Einsatz. Dieser wird mit Hilfe eines sogenannten Parser-Generators zur Laufzeit erzeugt. Im vorliegenden Fall fand der BISON-Parser-Generator [10] Anwendung, da er auch einige syntaktische Überprüfungen durchführt.
Das Parsing der XML-Dateien extrahiert die abgespeicherten Parameter und stellt diese strukturiert der Software zur entsprechenden Weiterverarbeitung zur Verfügung. Auf diese Weise werden auch Zeitinformationen, die in Form von Zeitprofilen im XML abgelegt sind, extrahiert und in der Software in die Form einer Aktionsliste (Aktions-Trigger) überführt. Dabei wird durch die Zeitinformation zum korrespondierenden Zeitpunkt während des Programmlaufs jeweils ein Aktions-Trigger ausgelöst, der in Relation zu einem Ressourcenobjekt steht. Dies impliziert, dass der Aktions-Trigger direkt auf die Auslastung / Nutzung einer Ressource, wie z.B. dem Arbeitsspeicher oder der CPU, wirkt. Die gezielte Beeinflussung der Ressourcen ist allerdings nur innerhalb einer CGroup sinnvoll und daher sind in der XML-Datei weiterhin alle für den Betrieb einer CGroup benötigten Informationen in Form von Parametern enthalten.
Die abgespeicherten CGroup-Parameter ermöglichen das automatisierte Erzeugen einer bestimmten CGroup während der Systemlaufzeit und die Veränderung bzw. Adaption ihrer Werte, wie beispielsweise das Volumen des zur Verfügung stehenden Arbeitsspeichers oder die Anzahl der möglichen CPU-Kerne. Dieser Steuerungsmechanismus und die daraus resultierende CGroup bilden die Basis von CGFail. Die so angelegte CGroup kann zur Isolation von Ressourcen, was auch als Sandbox [149] bezeichnet wird, genutzt werden. Eine Nutzung als Sandbox impliziert, dass alle im XML-Profil spezifizierten Aktionen und alle im

Kontext von CGFail ausgeführten Programme innerhalb der im XML-Profil definierten CGroup ausgeführt werden müssen. Dies gilt auch für die zu testende Software (SuT). Speziell die SuT wie auch die Lastgeneratoren, die Bestandteil von CGFail sind, müssen daher in die angelegte CGroup integriert werden. Die Integration geschieht durch Portierungsmechanismen, die in der CGroup API-Bibliothek vorhanden sind. Eine solche Portierung beinhaltet auch, dass die zuvor genannten Programme innerhalb der CGroup ausgeführt werden.

6.1.2 Interna von CGFail

Wie bereits in Kapitel 6.1.1 erwähnt, ist ein essentieller Bestandteil von CGFail die Zeitsteuerung, welche sämtliche von der Software auszuführenden Aktionen beeinflusst. Der in CGFail implementierte zeitgesteuerte Mechanismus basiert im Wesentlichen auf den Linux-Bibliotheken libev [86] und libevent [87] und beeinflusst sämtliche Aktionen, die sowohl das Verändern und Adaptieren von CGroup-Parametern (siehe Aufzählung Kapitel 5.1.1) als auch die Aktivierung, Konfiguration und Anpassung der Last-Generatoren umfassen, welche CPU- und Arbeitsspeicher-Last (I/O) erzeugen. Dies setzt voraus, dass CGroup-Parameter in der XML-Datei angegeben werden müssen. Dabei ist zu beachten, dass die Hardware-Ressourcen entsprechend ihrer tatsächlichen Verfügbarkeit bzw. ihres Volumens vordefiniert werden. Es obliegt dem Anwender, dies sowohl zum Erstellungszeitpunkt des XML-Profils als auch zur Programmausführung zu prüfen bzw. sicherzustellen, dass die Ressourcen in ausreichendem Maße zur Verfügung stehen. Während des Programmlaufs von CGFail ist zu berücksichtigen, dass die durch die Lastgeneratoren tatsächlich generierte Last sowohl durch die Parameter der CGroup wie auch durch Umgebungsbedingungen des Systems, d.h. durch parallel laufende Anwendungen, beschränkt werden kann. Das kann sich unter Umständen auf die real generierte Last in der Form auswirken, dass sie den spezifizierten Wert nicht erreicht. Ein Grund hierfür kann auch sein, dass die innerhalb von CGFail generierte Last die in Form der CGroup-Parameter spezifizierten Ressourcen nicht überschreiten darf, auch wenn das unterliegende System, also das System, auf dem CGFail betrieben wird, mehr Kapazität bereitstellt. Dieser Fall ist im Rahmen einer Anwendung von CGFail durchaus realistisch, da sich sowohl die zu testende Anwendung wie auch die Lastgeneratoren in der gleichen CGroup befinden und sich somit die Ressourcen teilen. Man könnte dies auch als eine Art konkurrierenden Betrieb ansehen, der Nebeneffekte mit sich bringen kann. Um dies zu vermeiden, erfolgt die Lastbeschränkung sowohl in Form einer Begrenzung durch die CGroup als auch durch eine Überwachung und Regulierung der Lastgeneratoren.
Bei der Initialisierung und Anpassung der CGroup-Parameter zur Systemlaufzeit erfolgt hingegen keine Überprüfung auf zulässige Werte. Diese Vorgänge werden während der CGFail-Laufzeit weder beschränkt noch überwacht und sollten auch sonst in keiner Weise beeinflusst werden, da sie die Erzeugung von Grenzfällen für die zu testende Software, die in der CGroup ausgeführt wird, ermöglichen.

6.1.3 Initiale Experimente

Zum Zweck der Evaluierung wurden initiale Experimente durchgeführt, die dazu dienen sollen, die Leistungsfähigkeit und die Möglichkeiten des CGFail-Ansatzes nachzuweisen. Die Portabilität der Software wurde auf unterschiedlichen Hardwarekonfigurationen, z.B. einem Notebook und zwei Servern, getestet. Dadurch war ein breites Spektrum an System-Konfigurationen abgedeckt, von einem Dual-Core Prozessor bis hin zum Server mit vier HEXA-Core-Prozessoren. Der Arbeitsspeicher umfasste zwischen 3GB und 16GB RAM. Bei der Durchführung der Experimente sind unterschiedliche LINUX-Distributionen, wie Ubuntu und Debian, verwendet worden. Im Rahmen der Evaluierung wurden ausschließlich Szenarien durchgeführt, bei denen die Ressourcen „Speicher und CPU“ betroffen waren bzw. verändert wurden. Neben realitätsnahen Zuständen, wie Speicherlecks und nicht genügend vorhandenem Arbeitsspeicher (Out-Of-Memory / OOM), wurden auch Grenzfälle, u.a. der Ausfall eines CPU-Kerns, nachgestellt.

6.2 Nichtlineare System-Identifikation des Lastverhaltens

Die nichtlineare System-Identifikation wurde wie in Kapitel 5.2 beschrieben durchgeführt. Die Darstellung der Least-Squares-Support-Vektor-Maschinen- (LS-SVM) Modelle, die aus dem theoretischen Ansatz resultieren, sind für sich alleine genommen wenig aussagekräftig, da die LS-SVM Modelle essentielle Teile des Zustandsraumsmodells (siehe Kapitel 4.6) darstellen, und die Auswertung des Zustandsraummodells nur im Gesamten sinnvolle Ergebnisse liefert. Dennoch wurden in den Grafiken 6.1 und 6.2 die ermittelten Abschätzungen der LS-SVM für Transfer- und Überwachungsfunktion abgebildet. Hierbei ist zu beachten, dass die Systemidentifikation einen dreidimensionalen internen Systemzustand ermittelt hat und somit die Transferfunktion einen dreidimensionalen Ausgabewert liefert. Dieser wurde, für jede Dimension getrennt, als separate Kurve in Abbildung 6.1 dargestellt. Die Evaluierung dieser Modelle erfolgte im Rahmen der in Kapitel 6.3 ausgeführten Auswertung. Dort wurden die SVM-Modelle aus den Grafiken 6.1 und 6.2 als Transfer- und Überwachungsfunktion im Kalman-Filter (UKF) verwendet. Zur Demonstration der Fähigkeiten der nichtlinearen System-Identifikation wurden einige initiale Experimente auf der Datenbasis einer Systemmonitoring-Lösung durchgeführt. Die Systemüberwachungsdaten eines Informationssystems, welches aus einem Server mit Linux-Betriebssystem, einer MySQL-Datenbank, einer Last-Test-Software für Datenbanken und den Lastgeneratoren von CGFail bestand, wurden aufgezeichnet. Die Last-Test-Software und der Lastgenerator bildeten dabei einen „Normal-Betrieb“ nach, bei dem aus einer Vielzahl aufgezeichneter Messwerte ausschließlich die Messwerte der Systemlast ausgewählt wurden, um die Verhaltenscharakteristik eines realen Systems abzubilden. Diese Maßnahme reduzierte die Komplexität und war somit einfach nachvollziehbar. Auf Basis dieser Daten konnten,

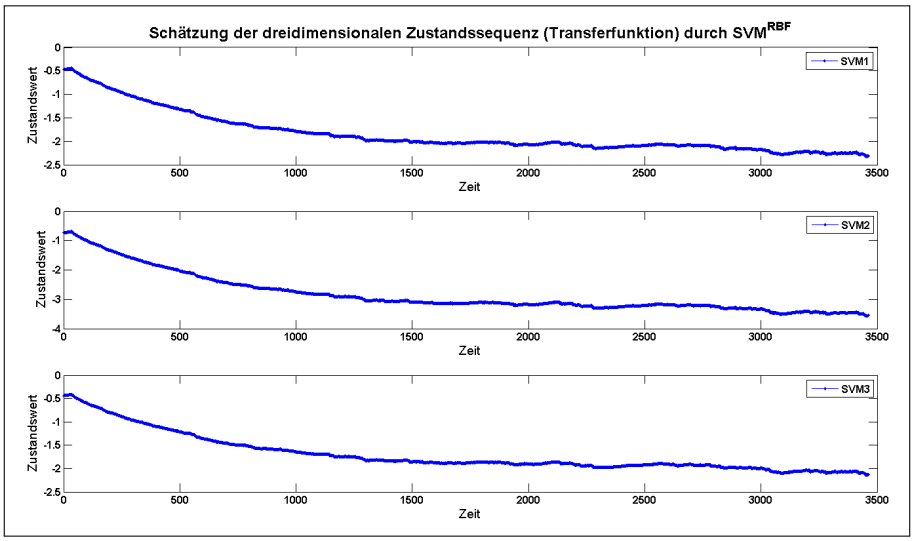


Abbildung 6.1: Ermittelte Abschätzung der dreidimensionalen Transferfunktion

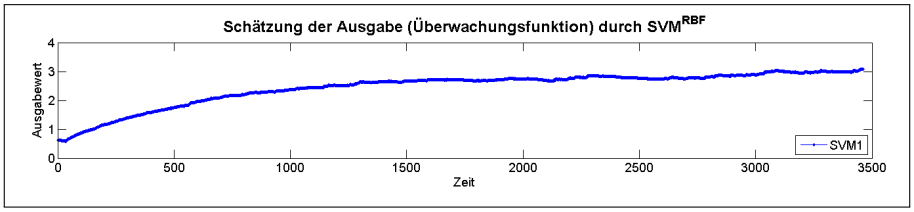


Abbildung 6.2: Abschätzung der Überwachungsfunktionen

wie in Kapitel 5.2 beschrieben, Modelle abgeleitet werden. Diese Modelle wurden auf zwei Beispiele angewandt, die in Kapitel 6.3 näher erläutert werden.

6.3 Anomalie-Erkennung im Lastverhalten

Die Anomalie-Erkennung in Informations-Systemen wurde auf der Basis des in Kapitel 5.3.2 vorgestellten Ansatzes durchgeführt. Dazu wurde das LSSVM-AUKF Kalman-Filter [45] inklusive der statistischen Methoden zur Prädiktionsergebnis-Bewertung in MATLAB [92] implementiert. Für die Auswertung wurde, wie bereits in Kapitel 6.2 erwähnt, eine systemcharakteristische Größe, die Last, gewählt. Dazu war es erforderlich, Messdaten des Lastverhaltens eines realen Informationssystems zu akquirieren. Diese Messdaten, welche essentiell für die Auswertung notwendig waren, wurden während des laufenden Betriebs eines auf einem Acht-Core-Server betriebenen Informationssystems aufgezeichnet. Bei der vorliegenden Arbeit umfasste das Informationssystem u.a. ein LINUX-

Betriebssystem und eine MySQL Datenbank (DB). Die Simulation eines realen Betriebsverhaltens wurde mittels einer Datenbank-Leistungstest-Software [46] nachgestellt, welche die Last in Form von Zugriffen bzw. Anfragen, in Anlehnung an das TPC-H Benchmark, auf die DB generierte. Die Datenbank-Leistungstest-Software [46] beinhaltet sowohl Lastprofile, Anfragen bzw. Daten und Schnittstellen zu deren Ausführung in einer Datenbank. Darüber hinaus wurde die Software CGFAIL [44] zum Erzeugen einer erhöhten Systembelastung und somit zur Einleitung anomaler Zustände eingesetzt. Dadurch sollte eine gezielte zeitgesteuerte Belastung des Systems erreicht werden, wie es z.B. auch auf Grund abgestürzter Programme / Dienste der Fall sein könnte.
Die Daten des Lastverhaltens wurden, s. Kapitel 6.2, durch eine Systemmonitoring-Lösung aufgezeichnet und enthielten auch Anomalien im Lastverhalten in Form von Schwankungen oder Änderungen bei den Lastwerten. Im Gegensatz zu den weitverbreiteten System-Messwerkzeugen, wie top, htop, die nur den aktuellen Wert ausgeben, wurden die Daten über einen längeren Zeitraum hinweg inklusive der korrespondierenden Zeitinformation erfasst. Dadurch war zu erkennen, dass die Anomalien auch die Kurvenform der aufgezeichneten Messwerte beeinflussen. Im Gegensatz zu schwellwertbasierten Verfahren begünstigte dieses Verfahren weitere Auswertungsmöglichkeiten, beispielsweise die Erkennung von Fehler-Mustern bzw. deren Extraktion. Eine deutlich robustere Bewertung von Schwankungen in den Messwerten, z.B. Ausreißer, war darüber hinaus möglich. Die Auswertung erfolgt wie nachfolgend beschrieben in zwei Schritten, sofern es sich um ein Informationssystem mit unbekannter Charakteristik handelt. Bei Systemen mit bekanntem Verhalten kann der erste Schritt entfallen.
In der ersten Phase, die wie bereits erwähnt nur bei Systemen mit unbekannter Verhaltensweise erforderlich ist, wird das Normalverhalten des Systems überwacht und die entsprechenden Messgrößen erfasst. Dabei muss sichergestellt sein, dass keine bzw. möglichst wenige Störquellen im System, und somit in den resultierenden Messwerten, vorliegen, z.B. durch ausgefallene oder falsch konfigurierte Komponenten. Auf der Basis der so gewonnenen Daten wird eine Systemidentifikation für die Modelle der Transfer- und Überwachungsfunktion ermittelt, entsprechend der Beschreibung in Kapitel 5.2 und der Ausführung in Kapitel 6.2.
In der zweiten Phase, der eigentlichen Anomalie-Erkennung, wird das System unter Betriebsbedingungen beobachtet und zur Evaluierung des Ansatzes im Bezug auf Fehlererkennung eine gezielte Stimulation durchgeführt. Dies ist erforderlich, da Fehler üblicherweise seltene Ereignisse sind, und geschieht in Form einer übermäßigen Belastung deutlich über Normalbetrieb. Im Rahmen der Evaluierung erfolgte die erhöhte Belastung sowohl durch den Datenbank-Leistungstest mit einem erhöhten Nutzungsaufkommen, als auch durch CGFail. Das Verhalten bzw. die Lastdefinition war in Form von Zeitprofilen, die die korrespondierenden Lastparameter enthielten, spezifiziert und resultierte in erkennbaren Verhaltensänderungen, wie z.B. Verzögerungen in der Antwortzeit der Datenbank, aber auch in der Signal-Charakteristik der Lastkurve. Solche Abweichungen können als Fehlverhalten interpretiert werden. Die DB-Antwortzeitverzögerungen wurden ausschließlich innerhalb des Datenbank-Leistungstests gemessen und ange-

zeigt, konnten jedoch auf Grund einer Einschränkung der Software nicht abgespeichert werden. Da keine Protokollierung erfolgen konnte, sind die Messungen nicht Bestandteil der nachfolgenden Auswertung. Hingegen wurden die Daten des Lastverhaltens erfasst. Dies ermöglichte einen Vergleich der aktuellen Lastkurve mit der im ersten Schritt aufgezeichneten, gestattete die Erkennung von Abweichungen im Signalverhalten und ließ somit Schlüsse hinsichtlich eines Fehlverhaltens zu.
In der vorliegenden Arbeit wurde die Auswertung wie folgt durchgeführt: Im ersten optionalen Schritt der normalen Lastphase, wurde das zu testende Informationssystem (IS) in einen regulären Betriebszustand versetzt und mit simulierter Last betrieben. Dies erfolgte durch eine Belastung der auf dem IS installierten MySQL-Datenbank mit einem TPC-H Benchmark-Leistungstest. Während der Laufzeit wurde eine Aufzeichnung der System-Messdaten vorgenommen, um im Anschluss an die Stimulationsphase eine datengetriebene Modell-Generierung zu erreichen. Im vorliegenden Fall erfolgte diese mit Hilfe einer System-Identifikations-Methode.
Im zweiten Schritt wurde das identische Informationssystem auf die gleiche Weise wie zuvor genutzt: Während Normal-Last generiert wurde, wurden parallel dazu Systemdaten aufgezeichnet. Zur Simulation von Fehlverhalten wurde CGFail eingesetzt, das eine gezielte und zeitgesteuerte Erzeugung weiterer Systembelastungen ermöglichte, da erhöhte Belastungen als Fehler aufgefasst wurden. Ein ähnlicher Effekt wäre auch durch ein erhöhtes Aufkommen an Datenbankabfragen im Rahmen eines Datenbank-Leistungstests erzielbar. Dies ließ sich jedoch mit der eingesetzten Datenbank-Leistungstest-Software nicht zuverlässig und reproduzierbar umsetzen, da ausschließlich eine Verzögerung bei den Antwort- und Bearbeitungszeiten der DB angezeigt wurde. Daher war es sinnvoll, die gezielte Belastung ausschließlich mit Hilfe von CGFail durchzuführen. Die in diesem Rahmen aufgezeichneten Messdaten bildeten die Basis für die eigentlichen Auswertungen der Anomalie-Erkennung.
Auf der Grundlage der aus der ersten Phase resultierenden Daten und der gewonnenen Betriebsdaten wurde die eigentliche Anomalie-Erkennung mit Hilfe von MATLAB [92] durchgeführt. Dazu wurden die Algorithmen aus den Kapiteln 5.2, 5.3.1 und 5.3.2 entsprechend implementiert. Ausgangspunkt für das LSSVM-AUKF Kalman-Filter waren die Modelle der Transfer- und Überwachungsfunktion, welche in datengetriebener Art und Weise abgeleitet wurden, s. Kapitel 5.2.
Die Messwerte aus dem zweiten Schritt wurden als Eingabedaten (y) für das LSSVM-AUKF Kalman Filter benutzt. Damit konnte eine Vorhersage des Lastverhaltens berechnet werden. Des Weiteren war eine statistische Bewertung der Prädiktionsergebnisse gegen die als Kalman-Eingabewerte genutzten Messdaten durchführbar. Hierzu kamen unterschiedliche statistische Methoden zur Identifikation von aufgetretenen oder bevorstehenden Fehlern zur Anwendung, die nachfolgend näher erläutert werden:
Eine in der Statistik gebräuchliche Methode zur Identifikation von Anomalien bzw. „Ausreißern“ ist die Distanz zur Standard-Abweichung (Sigma). Diese kann wie in Kapitel 5.3.1 beschrieben auf neue, zuvor unbekannte Messwerte

im Datenstrom angewandt werden. Dies impliziert jedoch, dass die Standard-Abweichung der unmittelbar zuvor aufgezeichneten Messwerte bekannt ist und ggf. regelmäßig aktualisiert wird. Das Ergebnis der Distanzberechnung ist ein Wert, der in praktischen Anwendungen häufig durch den Einsatz der Sigma-Regel ausgewertet wird. Im Bereich der Ausreißererkennung stellt die 3-Sigma Regel eine etablierte Metrik dar, bei der ein Konfidenzintervall von 99.73% über alle Messwerte zu Grunde liegt. Dies besagt, daß ca. 0.27% der Werte außerhalb des Konfidenzintervalls (CI) liegen. Für die Klassifikation von Fehlern ist dies nicht ausreichend. Bei noch selteneren Vorkommnissen können nur solche Werte berücksichtigt werden, die außerhalb des Mittelwertes +/− 4 Sigma (99.993% CI) bzw. 5 Sigma (99.99994% CI) liegen. Bei praktischen Anwendungen kann die vorgenannte Metrik als ausreichend genau angenommen werden, da hier üblicherweise Fehler selten auftreten.
Häufig treten bei realen Messwerten Schwankungen auf, die auch eine Instabilität der Prädiktionsergebnisse des Kalman-Filters bewirken, so dass diese variieren können. Würde nun eine Auswertung des Prädiktionsergebnisses gegen den letzten Messwert erfolgen, so kann diese zu unpräzisen bzw. falschen Resultaten führen und die Aussage der nachgelagerten statistischen Auswertung unbrauchbar machen. Eine Option, diesem entgegenzuwirken, wäre die Mittelung von Prädiktionswerten. Um genauere Ergebnisse zu erreichen, ist die Anwendung eines gewichteten Durchschnitts sinnvoll, s. Kapitel 5.3.2. In der vorliegenden Arbeit wurde ein solcher Durchschnitt, der auf der Mittelung von Prädiktionswerten basiert, zusätzlich mit einem Vergessenswert (Fading-Factor) gewichtet und danach auf die vom Kalman-Filter ermittelten Prädiktionsergebnisse angewandt. Im Rahmen der Evaluierung von empirischen Experimenten stellte sich ein Gewichtungsfaktor von 30% als guter Kompromiss für diese praktische Anwendung heraus. Dadurch wurden geglättete Ergebnisse erreicht, die jedoch nicht der Signaldynamik nacheilen. Die geglätteten Prädiktionswerte dienten als Datenbasis für sämtliche qualitativen Auswertungen.
Neben der Berechnung der Standard-Abweichung wurde auch ein Goodness-of-Fit-Test (GOF) berechnet. Die vorliegende Anwendung schränkte die Auswahl der einsetzbaren GOF-Tests sehr stark ein, daher wurde als GOF-Test die mittlere quadratische Abweichung (MSE) für die Systemmesswerte gegen die stabilisierten Prädiktionswerte ermittelt.
Basierend auf den zuvor getätigten Ausführungen wurden Experimente zur Anomalie-Erkennung durchgeführt. Die Evaluierungsergebnisse des Anomalie-Erkennungs-Ansatzes werden nachfolgend beschrieben. Eine ausdrucksvolle Darstellung wurde erreicht, indem jeweils die Systemmesswerte (Last) gegen die Prädiktionsergebnisse (Kalman-Ausgabewert) in den Grafiken abgebildet wurden. Eine Visualisierung der Werte erfolgte in der oberen Grafik der Abbildung 6.3. Dort werden die Original-Messkurve mit „Plus (+)“ und die Ergebnisse der Prädiktion mit „Kreis (o)“ gekennzeichnet abgebildet. Die korrespondierenden Werte der dargestellten Kurvenverläufe bilden die Datenbasis für alle weiteren statistischen Auswertungstests. Wie bereits zuvor erwähnt, war jedoch zu beachten, dass die Prädiktionsergebnisse mit einem gewichteten Durchschnitt stabilisiert werden.

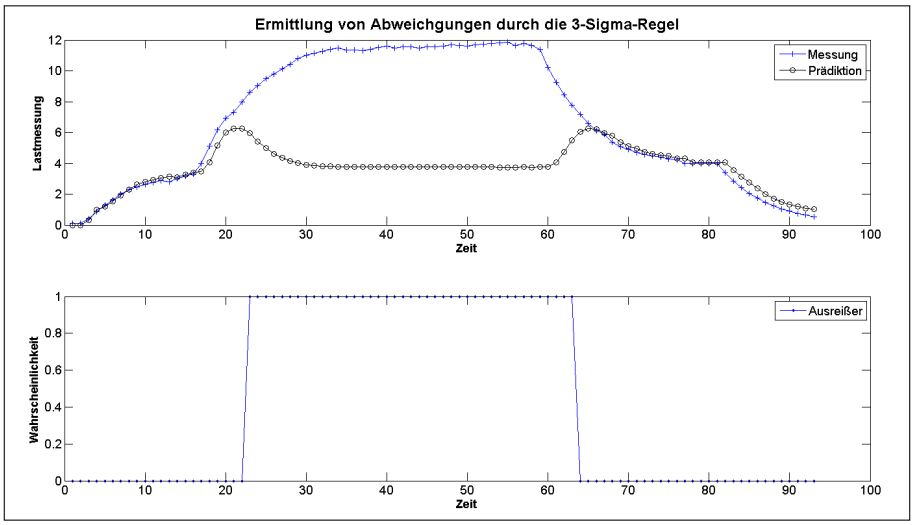


Abbildung 6.3: Auswertung mit der 3-Sigma Standardabweichung

Die Ergebnisse der Anwendung der 3-Sigma Regel auf die Standard-Abweichung der Prädiktionskurve sind in der unteren Grafik der Abbildung 6.3 dargestellt. Dabei werden Werte, die mehr als 3-Sigma abweichen, detektiert und sind in der unteren Grafik unschwer als die Werte zwischen den zwei Flanken im Kurvenverlauf zu erkennen. Sie signalisieren eine erhebliche Abweichung zwischen den Kurvenverläufen der Messung und der Prädiktion. Durch eine solche Markierung kann eine Klassifikation von fehlerhaftem Verhalten oder eine Merkmalserkennung (Muster) vom Auftreten über das Anstehen bis zum Abklingen erfolgen. Diese Technik gestattet nicht nur die Einordnung von Fehlverhalten, es ist auf dieser Basis auch eine Extraktion der Muster in Form von Kurvenverlaufsabschnitten möglich.

Des Weiteren wurden statistische Auswertungen durchgeführt, die die Modellgüte durch einen Vergleich der tatsächlichen Messwerte mit den Vorhersagen des Modells ermitteln und die Abweichungen der korrespondierenden Werte (Messwert versus Prädiktionswert) bewerten. Die Ergebnisse sind in der Abbildung 6.4 dargestellt. Die Resultate der Auswertung des Log-Likelihood-Tests sind der mittleren Grafik zu entnehmen. Hier ist die Divergenz der in Beziehung stehenden Werte durch einen deutlichen Anstieg im Kurvenverlauf zu erkennen. Der Anstieg tritt jedoch erst sehr stark verzögert ein, so dass, verglichen mit der 3-Sigma-Auswertung aus Abbildung 6.3 eine Signalisierung des Fehlverhaltens erst viel später erfolgen kann.
In der unteren Grafik der Abbildung 6.4 werden die Ergebnisse, die die Anwendung des Goodness-of-Fit- (GOF) Tests ergaben, vorgestellt. Auch hier wird der Unterschied von Messwert zu Prädiktionswert analysiert. Es ist zu erkennen, dass der Kurvenverlauf verglichen mit den Ergebnissen der Auswertung

des Log-Likelihood-Tests generell weniger steile Flanken aufweist und weiterhin deutlich früher mit dem Anstieg beginnt, allerdings wird durch den sanften Anstieg des Kurvenverlauf eine Klassifikation hinsichtlich des Fehlverhaltens erschwert. Weiterhin wird bedingt durch das flache Ansteigen der Auswertekurve das maximale Niveau erst zu einem deutlich verzögerten Zeitpunkt erreicht als dies beim Log-Likelihood Test und der Sigma-Abweichung der Fall ist.
Für die Anwendung im Bereich der Anomalie-Erkennung kann bei beiden Tests die Klassifikation mit einem ausgewählten Schwellwert durchgeführt werden. Im Falle des GOF könnte ein entsprechender Schwellwert deutlich vor Erreichen des Maximums passiert werden. Das würde das verzögerte Verhalten kompensieren und präzisere Ergebnisse in Bezug auf Abweichungen liefern. Des Weiteren wäre, ähnlich wie bei der Anwendung der 3-Sigma-Regel, eine Merkmalserkennung (Muster) bzw. eine Merkmalsextraktion vom Auftreten über das Anstehen bis zum Abklingen möglich.
Zum Abschluss wurde ein genereller Vergleich der Log-Likelihood- und der Goodness-of-Fit-Test-Resultate durchgeführt. Wie bereits erwähnt werden Abweichungen beim GOF-Test deutlich früher als beim Log-Likelihood-Test im Kurvenverlauf darstellt. Prinzipiell ist bei der Anomalie-Erkennung bzw. Fehlerdetektion eine möglichst frühe Signalisierung wünschenswert, um zeitnah Abhilfe schaffen zu können.

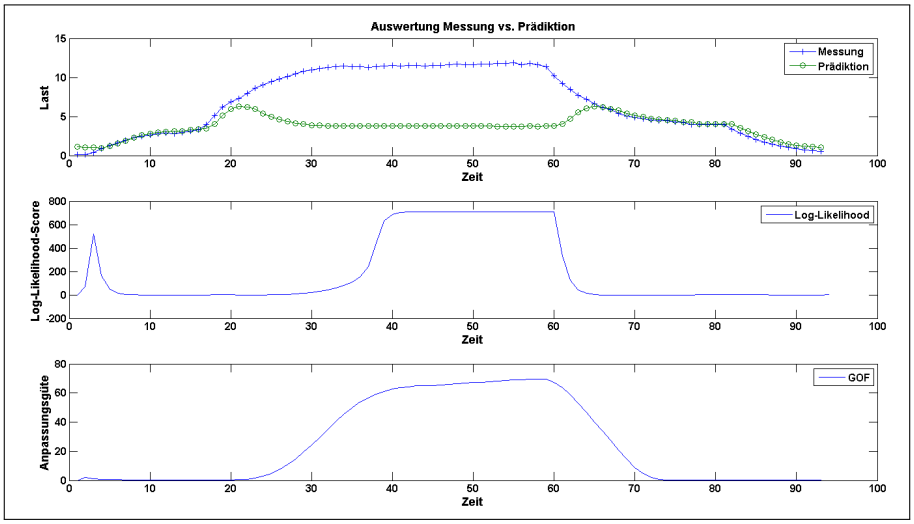


Abbildung 6.4: Auswertung mit GOF und Log-Likelihood

6.4 Initiale Wahrscheinlichkeit für Bayes'sche Netze mit kontinuierlicher Zeit

6.4.1 Diskretisierung kontinuierlicher Variablen

Die Überwachung und Betriebszustandskontrolle von Informations-Systemen erzeugt sowohl diskrete als auch kontinuierliche Signalformen und eine große Menge an Daten mit unterschiedlichen Charakteristika.
Die Anzahl unterschiedlicher Signalcharakteristika erweist sich mitunter bei der elektronischen Verarbeitung als Hindernis, da viele in der Informatik etablierte Verfahren und Algorithmen auf die Verarbeitung einer Signalform beschränkt sind, die häufig ausschließlich diskrete Daten umfasst. Dies ist auch bei der in der vorliegenden Arbeit angewandten Methodik, den Bayes'schen Netzen mit kontinuierlicher Zeit, der Fall. In diesem Kontext muss im Rahmen der Diskretisierung die korrespondierende Zeitinformation beibehalten werden, da dadurch die Auswahl hinsichtlich einer zeitkontinuierlichen und wertdiskreten Darstellung von kontinuierlichen Messgrößen eingeschränkt wird. Die Diskretisierung wird wie in Kapitel 5.4 beschrieben durchgeführt.
Eine realitätsnahe Evaluierung des in Kapitel 5.4 beschriebenen Diskretisierungsansatzes erfolgte auf der Basis von Systemüberwachungsdaten, die von dem verwendeten Informationssystem (IS) aufgezeichnet wurden. Der aufgezeichnete Datensatz enthielt kontinuierliche Variablen, u.a. die Systemlast (CPU-Auslastung). Diese Messgröße wurde vom IS in verschiedenen Zeitauflösungen bereitgestellt. Neben der aktuellen Last, die im Minutentakt aktualisiert wurde, existierten Durchschnittswerte für die gemittelte Last über fünf Minuten bzw. fünfzehn Minuten. Die nicht gemittelte Lastmessung wurde als eine verständliche Messgröße angesehen und bildete daher die Datenbasis der Evaluierung. Zur Veranschaulichung wurde eine Messkurve der aufgezeichneten Last (CPU-Auslastung) in Abbildung 6.5 dargestellt.
Auf diese Messdaten (Abbildung 6.5) wurde der Binning-Algorithmus (s. Ka-

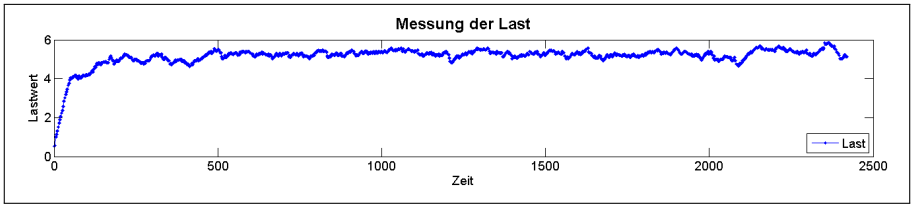


Abbildung 6.5: Informationssystem Lastmessung

pitel 5.4.2) angewandt und in MATLAB [92] implementiert. Im Rahmen der algorithmischen Verarbeitung war die Berechnung eines Histogramms der Daten erforderlich. Das dabei ermittelte Histogramm der Last-Messung zeigt Abbildung 6.6.
Das Histogramm ermöglichte eine Verteilungsanalyse, durch die Änderungen in der Verteilung der einzelnen Variablenausprägungen ermittelt werden konnten.

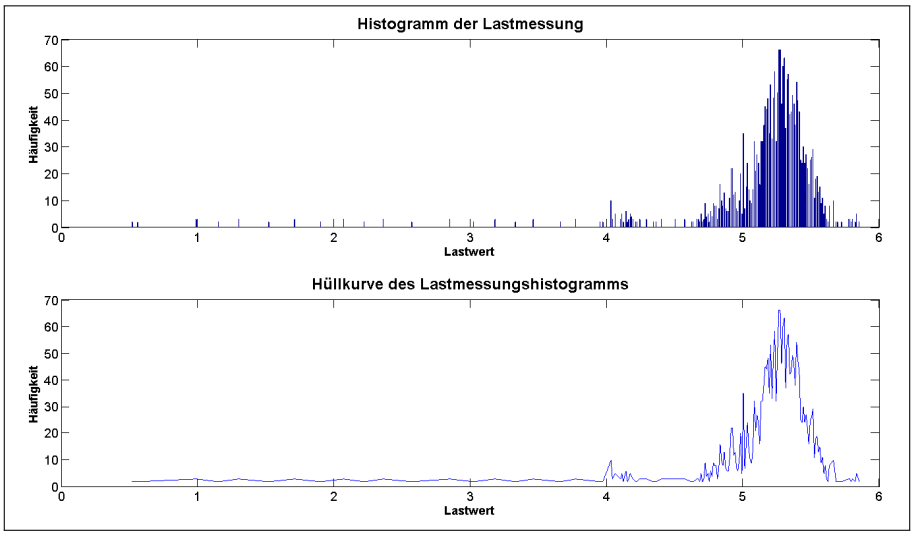


Abbildung 6.6: Histogramm der Lastkurve

Die Änderungen wurden in der vorliegenden Arbeit mit Hilfe von Distanzmaßen, die sich aus im Histogramm benachbarten Ausprägungen ergaben, berechnet. In der nachfolgend dargestellten Untersuchung wurde die Minkowski-Distanz (Gleichung 5.11) verwendet. Die dabei entstandenen Resultate hingen ausschließlich von den zu Grunde liegenden auszuwertenden Daten ab und können somit nicht verallgemeinert werden. Aus dem gleichen Grund kann auch für den Distanzschwellwert, der eine Änderung der Häufigkeit auswertet, kein allgemeiner Wert vorgegeben werden. Im Rahmen der vorliegenden Arbeit wurde der erforderliche Distanzschwellwert empirisch ermittelt. Dieser Distanzschwellwert musste an die Datenlage bzw. das zu analysierende Problem so angepasst werden, dass nur signifikante Veränderungen im jeweiligen Kontext detektiert wurden. Eine Schwellwertgröße von 3 (Minkowski-Distanz) erwies sich für die Auswertung der Frequenz als ein praktikabler Mittelwert, da sich bei der Anwendung dieses Wertes hinsichtlich der Signalform und der Signalqualität wenig Einbußen ergaben. Die Anwendung der Minkowski-Distanz-Berechnung mit dem zuvor benannten Schwellwert auf den o.a. Datensatz führte zu einer Reduktion von 153 Ausprägungen auf 69 Bins. Das Ergebnis zeigt Abbildung 6.7. In der Grafik sind die Bins als Intervalle zwischen den Markierungen (Kreise) zu erkennen. Prinzipiell ist der Diskretisierungsschritt nach der Bin-Berechnung beendet, jedoch ist es für eine geeignete numerische Darstellung erforderlich, z.B. bei einer Weiterverarbeitung, jeden Bin mit einer eindeutigen Nummer zu versehen.

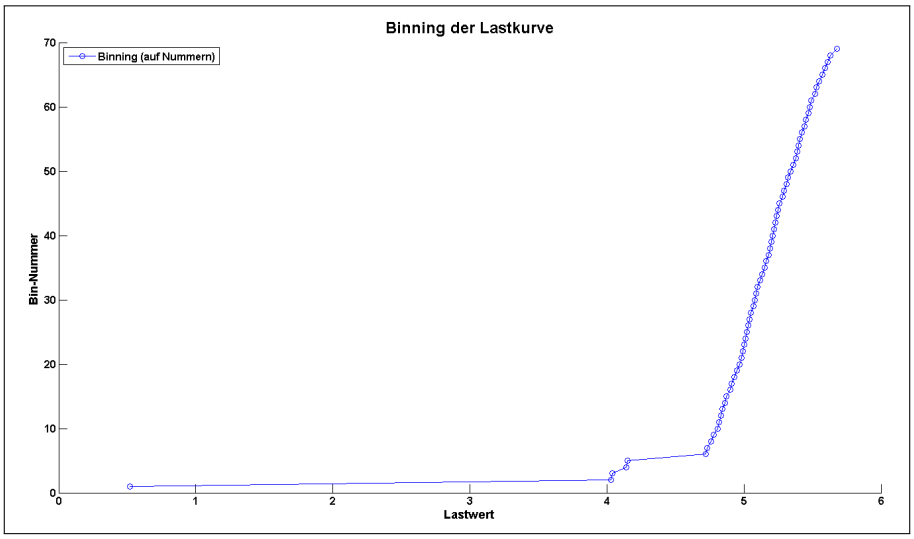


Abbildung 6.7: Binning der Lastkurve

6.4.2 Initiale Wahrscheinlichkeit

Die in der vorliegenden Arbeit eingesetzten Bayes'schen Netze mit kontinuierlicher Zeit (CTBN) erfordern für präzise Berechnungen die Spezifikation von exakten initialen Wahrscheinlichkeiten. Die initiale Wahrscheinlichkeit ist für die Ausprägungen (Instanzen) aller im CTBN genutzten Variablen notwendig. Bei den üblicherweise verwendeten ganzzahligen Variablen können diese Wahrscheinlichkeiten durch normierte Häufigkeitsverteilungen, welche den kompletten Wertebereich der Variablen abdecken, berechnet werden. Bei kontinuierlichen Variablen gestaltet sich das meist schwierig, mitunter ist es sogar unmöglich, aus aufgezeichneten Messdaten eine Verteilung, die den kompletten Wertebereich abdeckt, abzuleiten. Dies ist mitunter dadurch bedingt, dass nicht für alle Ausprägungen des kompletten Wertebereichs einer Variablen Messdaten existieren und somit der Funktionsverlauf nicht vollständig abgebildet ist. Dies stellt dann ein Problem dar, wenn Sprünge in der Verteilung vorkommen und somit eine Unterteilung erfolgen müsste. Bei kontinuierlichen Variablen stellt die Approximation der Dichtefunktion eine Alternative bei der Berechnung der Häufigkeitsverteilung dar. Im Gegensatz zum Histogramm, das aus den einzelnen Ausprägungen der Variablen besteht, wird bei der Dichtefunktion die Dichte für den kompletten Variablen-Wertebereich abgeschätzt. Es ist dabei nicht erforderlich, eine Wahrscheinlichkeit für jede Ausprägung zu berechnen. Diese Methode ist in Kapitel 5.5 detailliert beschrieben und wurde im Rahmen meiner Untersuchungen auf der Basis einer MATLAB-Implementierung des MoTBF-[83] Ansatzes, angewandt. Den aus der Simulation resultierenden Graphen der Dichteschätzung zeigt Abbildung 6.8, die mit der Simulation korrespondierende Gleichung für die Wahrscheinlichkeitsdichte entspricht Gleichung 6.1.

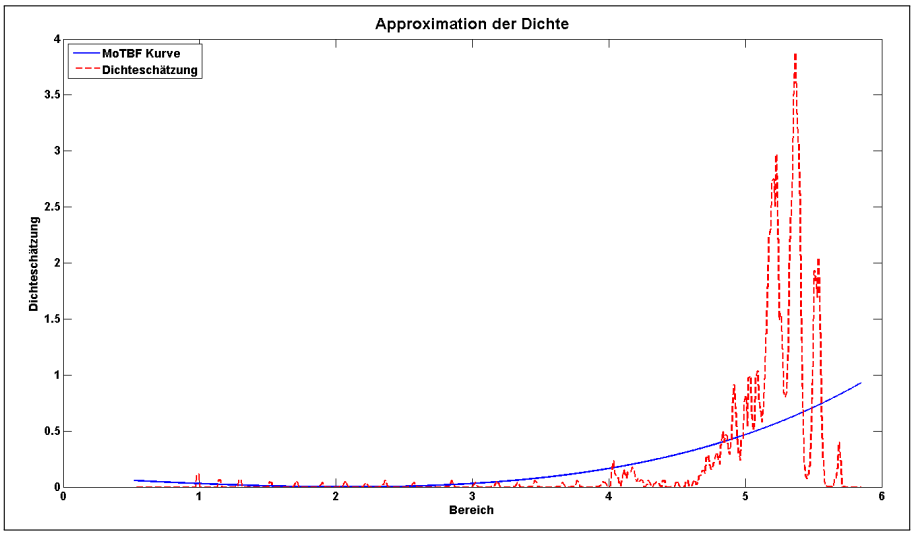


Abbildung 6.8: Dichteschätzung und ermittelte MoTBF-Kurve

$$\begin{aligned} f(x) &= 2.0420/2.3087 - 3.0437/2.3087 \\ &*exp((x - 0.52)/5.33) + 1.1342/2.3087 \\ &*exp(2(x - 0.52)/5.33) \end{aligned} \tag{6.1}$$

Die Wahrscheinlichkeit (PDF) für jeden in der Grafik 6.7 dargestellten Bin lässt sich mit Hilfe von Gleichung 6.1 ermitteln, indem partiell über die PDF integriert wird. Es ist dabei zu beachten, dass die Wertebereichsgrenzen des entsprechenden Bins auch die Intervallgrenzen bei der Integration bilden. Das Verfahren wird exemplarisch am ersten Bin aus Grafik 6.7 veranschaulicht. Durch die Anwendung des Binning-Verfahrens (Kapitel 6.4.1) wurden die Grenzen des ersten Bins als Intervall [0.52 , 4.03] ermittelt, die auch der Abbildung 6.7 entnommen werden können. Diese Werte beschränken den Integrationsbereich und werden wie bereits erwähnt als Integrationsgrenzen bei der PDF-Funktion aus Gleichung 6.1 eingesetzt. Die Integration liefert für den ersten Bin eine Wahrscheinlichkeit von 0.1332. Im Kontext des CTBN ist somit $P^0_{x_1}$ bestimmt. Hierbei gibt P^0 die initiale Wahrscheinlichkeit, X die Nummer der involvierten Variablen und der Index 1 die korrespondierende Bin-Nummer (Instanz) der Variablen an. Durch diese Zuordnung ist die direkte Anwendung bei CTBN möglich.

6.5 Modellbildung und Fehlervorhersage auf Basis von Bayes'schen Netzen mit kontinuierlicher Zeit

Für die Abschätzung von Ausfällen bzw. bei der Zuverlässigkeitsanalyse wird häufig ein System als mathematisches Modell dargestellt. Eine entsprechende Modellierung basierend auf Bayes'schen Netzen mit kontinuierlicher Zeit (CTBN) wird nachfolgend erläutert. Deren realistische Umsetzung wurde mit einer Auswahl an Daten, die das Systemverhalten repräsentieren, sichergestellt. Da die Daten für die Auswertung von einem Server-Cluster im operativen Betrieb stammen, wird Wirklichkeitstreue gewährleistet. Bei den Daten handelt es sich um Systemmesswerte der Systemressourcen und der aufgetretenen Fehlerzustände mit korrespondierenden Zeitinformationen.
Von den aufgezeichneten Rohdaten des Informationssystems wurden die kontinuierlichen Variablen, wie in Kapitel 5.4 beschrieben, in wertdiskrete Daten überführt. Dabei war zu berücksichtigen, dass die im Rahmen der Systemüberwachung gewonnenen Zeitinformationen, die im Datensatz als separate, korrespondierende Zeitstempel vorhanden sind, bei der Umwandlung der kontinuierlichen Werte in diskrete Werte erhalten blieben und weiterhin in der richtigen Beziehung zum neuen Wert stehen. Die in den Daten enthaltenen Fehlerereignisse wurden mit der Methode aus Kapitel 5.6 signalisiert und in den Datenbestand integriert.
Der zuvor erwähnte erweiterte Datensatz bildete die Datenbasis für den Modellierungsprozess. Eine detaillierte Beschreibung des Modellierungsprozesses ist Kapitel 5.7 zu entnehmen. Auf dieser Grundlage erfolgte die entsprechende Umsetzung in eine Software, welche die Basis für Experimente bildete.
Die Funktionsweise des bei der Evaluierung zum Einsatz gekommenen Software-Prototyps basiert auf dem algorithmischen Ansatz, der sich in drei Schritte gliedert, siehe Kapitel 5.7. Vor der Ausführung des Software-Prototyps muss die Granularität der Daten und somit die Beschaffenheit des Datensatzes berücksichtigt werden. Hierbei gilt: Je höher eine Auflösung der zu Grunde liegenden Messdaten ist, desto höher werden die Anforderungen an den Arbeitsspeicher bei der Auswertung.
Nachfolgend werden die in der o. a. Software eingesetzten Komponenten inklusive deren Umsetzung beschrieben:
Im ersten Schritt des Algorithmus wurde die Struktur des Bayes'schen Netzes (BN) durch Anwendung des Max-Min Hill-Climbing (MMHC) -Algorithmus [144] ermittelt. MMHC unterstützt die Verarbeitung diskreter (ganzzahliger) und kontinuierlicher (reellwertiger) Variablen, jedoch stellte sich im Rahmen einiger Experimente mit den o.a. Rohdaten des Informationssystems heraus, dass die Bearbeitung reellwertiger Variablen deutlich mehr Zeit in Anspruch nahm als dies bei wertdiskreten Variablen der Fall war. Deshalb wurde dazu übergegangen, sämtliche realen Variablen vorab in ganzzahlige zu wandeln. Dieses Vorgehen kann auch damit begründet werden, dass das eingesetzte CTBN-Framework zum Zeitpunkt der Erstellung der Arbeit ausschließlich nur ganzzah-

lige Variablen verarbeiten konnte. Dies impliziert, dass bei einer Verarbeitung der reellwertigen Daten mit dem CTBN-Framkework eine Umwandlung erforderlich geworden wäre. Daher wurde die Umwandlung und das Konditionieren aller im Datensatz enthaltenen kontinuierlichen Variablen, wie zuvor in Kapitel 6.4.1 beschrieben, durchgeführt und die resultierende Struktur des Bayes'schen Netzes aus ganzzahligen Daten ermittelt. Eine Implementierung von MMHC war nicht erforderlich, da diese im Bnlearn- [123] Paket, einer etablierten Bibliothek für die in der Statistik gebräuchliche Software R [109], enthalten ist. Es bot sich daher an, diese Kombination für das Lernen der Struktur zu nutzen. Im zweiten Schritt sollen die Parameter der Bayes'schen Netze mit kontinuierlicher Zeit (CTBN) gelernt und das Verhalten des CTBN ermittelt werden. Auch für diesen Zweck existiert eine frei verfügbare Implementierung, die in C++ realisiert ist. Es handelt sich dabei um die "Continuous Time Bayesian Network Reasoning and Learning Engine"(CTBN-RLE) [128], die die gebräuchlichen Algorithmen zum Lernen der Parameter und der Demonstration der CTBN-Modellfähigkeiten beinhaltet. Da die Software bei der Ausführung Berechnungen performant durchführte und nur geringfügige Anpassungen erforderlich waren, wurde diese eingesetzt.
Der Datensatz der IS-Rohdaten vom Server-Cluster besteht aus aufgezeichneten Systemüberwachungswerten mit dazugehörigen Zeitstempeln. Die Systemvariablen sind der LINUX vmstat (virtuelle Speicherstatistiken) [147] entnommen und umfassen die folgenden Messgrößen: CPUWait, MajorPageFaults, MajorPageFaultsFlagged, CPUIdle, CPUSystem, LoadOne, LoadFive, MemoryUsed, PageFaults.
Die Variablen sind wie folgt definiert:

- CPUWait: Wartezeit, in der die CPU auf eine Datenanforderung eines blockorientierten Gerätes (I/O) wartet. Diese wird in prozentualer Wartezeit angegeben.
- MajorPageFaults: Anzahl der Seitenfehler (Page faults) seit dem letzten Systemstart. Ein Seitenfehler tritt auf, wenn ein Programm auf einen Speicherbereich zugreift, der auf die Festplatte ausgelagert wurde und sich nicht im Hauptspeicher (RAM, Cache) befindet.
- CPUIdle: Zeit, in der sich die CPU im Leerlauf befindet.
- CPUSystem: Zeit, in der die CPU Betriebssystem-Kern-Funktionalitäten ausführt
- LoadOne: Einminütiger Lastdurchschnitt des Systems; Die Zahl muss in Relation zu der Anzahl der im System vorhandenen CPUs gestellt werden.
- LoadFive: Fünf-minütiger Lastdurchschnitt; Die Zahl muss in Relation zu der Anzahl der im System vorhandenen CPUs gestellt werden.
- MemoryUsed: Belegter Arbeitsspeicher im MB.

- PageFaults: Anzahl der Seitenfehler seit dem letzten Systemstart, wenn der Betriebssystemkern einen Speicherbereich, der nicht im RAM / Cache verfügbar ist, anfordert. Die Größe wird in Einheiten pro Sekunde angegeben.

Die IS-Rohdaten enthalten einen mehrfach vorkommenden Fehlerzustand, den OOM-Killer. Dieser kann nicht direkt als Systemvariable erfasst werden, sondern ist eine dem System-Log entnommene Ereignisbenachrichtigung. Diese Event-Nachrichten können bei Bayes'schen Netzen nicht direkt mit kontinuierlicher Zeit modelliert werden, da CTBN nur numerische Daten unterstützt. Dagegen kann eine direkte Weiterverarbeitung der aufgezeichneten Systemvariablen erfolgen. Somit bilden deren Messwerte die Basis für den endgültigen Datensatz. Beim OOM-Killer jedoch ist vor der Integration in den Datensatz eine Adaption der Signalisierung nötig. Diese wird durch eine entsprechende Modellierung des OOM-Killer-Zustandes, wie in Kapitel 5.6 beschrieben, umgesetzt. Zu diesem Zweck wird eine Hilfsvariable, die den Fehlerzustand durch eine Markierung anzeigt, siehe hierzu Grafik 5.4, eingeführt und in den Datensatz eingebettet. Der so erweiterte Rohdatensatz bildet die Grundlage für alle weiteren Schritte der Auswertung.
Danach werden sämtliche reellen Zahlen bzw. die entsprechenden Variablen in Ganzzahlen überführt, s. Kapitel 5.4. Es ist zu beachten dass nur eine geringe Reduktion der Ausprägungen erfolgt. Die im Datensatz enthaltenen Zeitinformationen sind für das Lernen der Struktur des Bayes'schen Netzes irrelevant und müssen temporär, d.h. für den Lernprozess, ausgeblendet bzw. eliminiert werden. Durch Anwenden der MMHC-Implementierung lässt sich nun die Struktur bestimmen. Hierbei ist besonders ein Merkmal von MMHC zu erwähnen, das es gestattet, Vorgaben (Zwangsbedingungen) zu spezifizieren, die beim Lernvorgang unbedingt zu berücksichtigen sind.
Eine solche Funktionalität ist hilfreich, wenn Fehler bzw. Fehlerzustände und deren auslösende Ereignisse bekannt sind und verknüpft werden sollen. Im zu Grunde liegenden Datensatz ist dies der Fall: Hier wurde ein Vorkommnis, der OOM-Killer, signalisiert, welches zeitunabhängig auftritt und zwar genau dann, wenn der Speicherverbrauch aus dem Rahmen läuft. Der OOM-Killer [42] ist bei aktuell gebräuchlichen Linux-Betriebssystemen in deren Kern (Kernel) integriert. In der praktischen Anwendung vergleicht der OOM-Killer die frei verfügbare Speichermenge [17] gegen einen festen Schwellwert und wird bei zu geringen freien Resourcen aktiv. Dies bedeutet, dass erst dann der OOM-Killer zur Ausführung kommt, wenn der Schwellwert passiert wurde. Es sollte berücksichtigt werden, dass der Mechanismus des OOM-Killers eine präventive Maßnahme ist mit dem Ziel, Systemausfälle zu vermeiden. Die Funktion des OOM-Killers ist es, eine zu hohe Speicherauslastung, die den Kern des Betriebssystems in Bedrängnis bringen könnte, zu verhindern.
Beim überwachten Server konnte einem separaten System-Log (Logdatei) entnommen werden, dass meist prototypische Softwareumsetzungen, welche zu Testzwecken gestartet wurden, die Ursache für ein OOM-Killer-Event waren. Eine Analyse der Logs ergab, dass entweder eine Unterschätzung des Speicherbe-

darfs oder Memory-Leaks ein solches Ereignis auslösten. Der Zusammenhang zwischen der OOM-Killer-Aktivität und dem Speicherverbrauch lässt sich in Form einer Relation abbilden, die als Zwangsbedingung für MMHC spezifiziert wurde. Möglich ist dies, wenn der zur Strukturbestimmung genutzte Datensatz als Tabelle dargestellt ist, bei der die enthaltenen Variablen die Spalten vorgeben. Die Vorgabe an MMHC erfolgte dabei durch die Übergabe der entsprechenden Spaltennummer der Variablen im Datensatz. Hierdurch wurde die Berücksichtigung der Relation während des Struktur-Lernvorgangs und somit die Abbildung im Bayes'schen-Netz-Modell gewährleistet.
Aus dem MMHC-Lernvorgang resultiert das in Abbildung 6.9 dargestellte Strukturmodell eines Bayes'schen-Netzes. Hier korrespondieren die Zustände X_i mit den folgenden Messgrössen (inklusive der Hilfsvariablen): CPUWait, MajorPageFaults, MajorPageFaultsFlagged, OOM-Killer, CPUIdle, CPUSystem, LoadOne, LoadFive, MemoryUsed, PageFaults.

Die Bayes'sche Netzstruktur aus Grafik 6.9 wird als Gerüst für das CTBN-Modell genutzt. Prinzipiell ist auf Basis der Bayes'schen Netzstruktur und des Datensatzes die Ermittlung der Zeitcharakteristik des CTBN-Modells möglich. Zur Erzielung präziser Ergebnisse der Zeitcharakteristik des CTBN-Modells sind neben dem Systemdatensatz nun noch die initialen Wahrscheinlichkeiten zu jeder Variablen-Ausprägung erforderlich.
Der zur Strukturbestimmung genutzte Datensatz wurde bei praktischen Experimenten für die CTBN-Parameter-Berechnung genutzt. Dabei stellte sich heraus, dass der Speicherbedarf während des Lernvorgangs sehr hoch (mehrere 100GB) war. Hierdurch wurden die vorhandenen Ressourcen komplett ausgeschöpft, was zu Abbrüchen bei den Berechnungen führte. Auf Grund dieser Tatsache war eine Reduktion des erweiterten Datensatzes zwingend erforderlich, um die vorgesehenen Berechnungen mit den vorhandenen moderaten Hardware-Ressourcen, besonders im Bereich des Speichers (ein TB) durchführen zu können.
Durch eine nachträglich durchgeführte geringere Auflösung der Variablen des erweiterten Datensatzes wurde eine höhere Reduktion erreicht. Das hatte den Vorteil, dass für die Berechnungen nun deutlich weniger Speicherkapazität (ca. 70GB) benötigt wurde, aber Ungenauigkeiten beim Ergebnis der Parameterberechnung mit sich brachte. Die Reduktion der Daten erfolgte mit Hilfe der in Kapitel 5.4.2 beschriebenen Methode. Diese führte dabei auch eine Domänen-Partitionierung durch, die die für die CTBN Berechnungen benötigten initialen Wahrscheinlichkeiten für jede Variablenausprägung bestimmte.
Mit der BN-Struktur und den initialen Wahrscheinlichkeiten als Grundlage besteht der CTBN-Lernprozess ausschließlich aus der Ermittlung der CTBN-Parameter, den Intensitätsmatrizen. Durch die Vorgabe der BN-Struktur ist sichergestellt, dass sämtliche im Rahmen der Struktur-Lernphase spezifizierten Transitionen im CTBN-Modell enthalten sind.
Die Darstellung des kompletten CTBN-Modells wirkt auf Grund der Größe und der damit verbundenen Vielzahl an Intensitätsmatrizen unübersichtlich. Exemplarisch wird das Verhalten des OOM-Killers, der durch die Variable X_4 repräsentiert wird, aufgezeigt. Um eine übersichtlichere Darstellung zu erreichen,

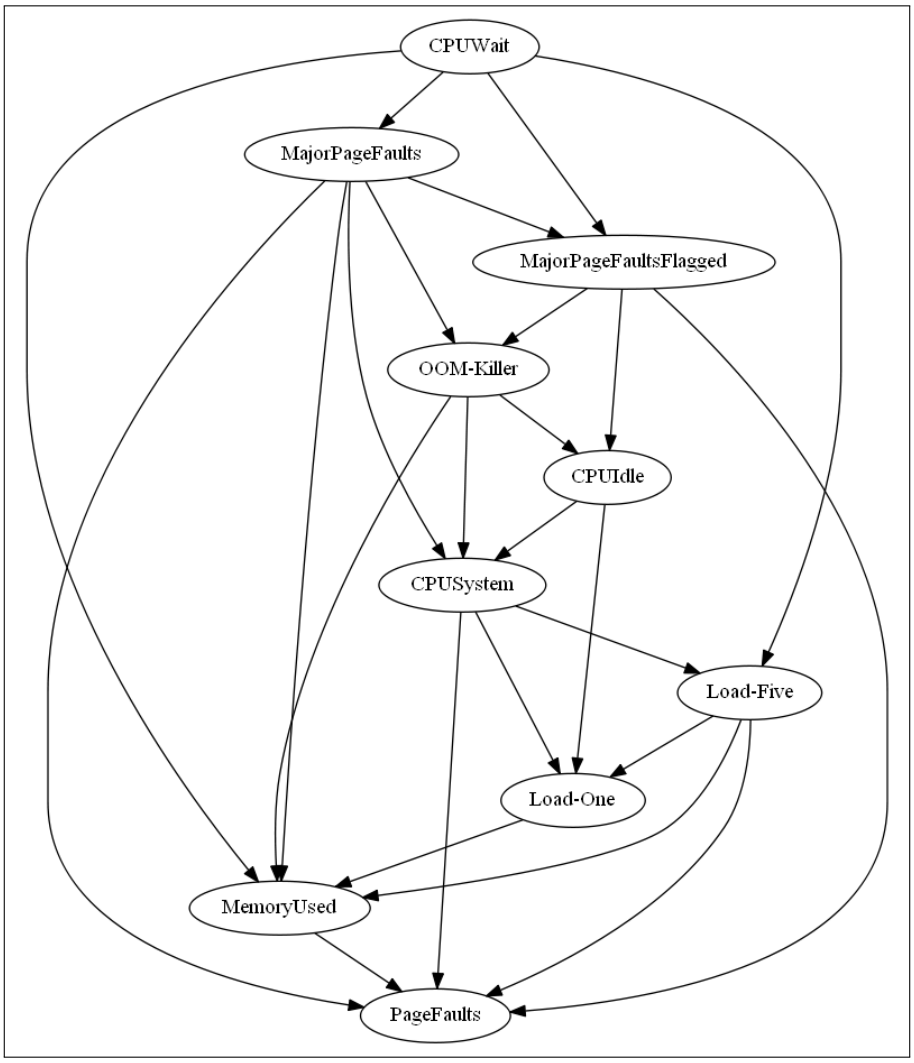


Abbildung 6.9: Bayes'sches Netzwerkmodell des zu Grunde liegenden IS

wird nicht der komplette konditionierte Intensitätsmatrix-Satz (Q_4) dieser Variablen gezeigt, sondern lediglich eine Intensitätsmatrix (Nr. 81), bei der das Umschaltverhalten sehr deutlich zu erkennen ist (siehe Gleichung 6.2). Bei allen anderen im Satz enthaltenen Matrizen ist die Beeinflussung der externen Variablen weniger klar ersichtlich. Der Einfluss auf das Verhalten der Variablen ist eher gering, daher bleiben diese in der Darstellung unberücksichtigt.

$$Q_{4,81} = \begin{bmatrix} -0.00060855 & 0.00060855 \\ 1 & -1 \end{bmatrix} \tag{6.2}$$

Die Bestimmung der Zeit bis zu einem Zustandsübergang und die Wechselwahrscheinlichkeit in einen anderen definierten Zustand kann durch Anwenden nachfolgend angeführter Formeln auf die Intensitätsmatrix ermittelt werden. Exemplarisch wird dies auf der Basis der o. a. Intensitätsmatrix durchgeführt und in Gleichung 6.2 dargestellt mit den aus Kapitel 4.3.2 entnommen Formeln. Aus der dargestellten Matrix in der Gleichung 6.2 ist zu entnehmen, dass der OOM-Killer (siehe Kapitel 5.6) nur zwei Zustände (0 und 1) annehmen kann. Die erwartete Zeit für den Wechsel aus dem Zustand x kann durch Anwendung der Formel $t = 1/q_i^x$ auf die Elemente der Gegendiagonalen bestimmt werden. Für den Wechsel von 0 nach 1 beträgt diese 1643.25 Sekunden.
Des Weiteren kann die Wahrscheinlichkeit, dass der OOM-Killer aus einem Zustand (x) in einen neuen Zustand (x') wechselt, mit $\Theta_{xx'} = q_{xx'}/q_x$ angegeben werden. Im konkreten, recht übersichtlichen Fall ergibt dies die Wahrscheinlichkeiten 0.00060855 für den Wechsel von 0 nach 1. Dem Log-Datensatz war nur die Aktivierung und nicht die Aktivitätszeit des OOM-Killers zu entnehmen. Daher ergibt sich für den Wechsel von 1 nach 0 eine Wahrscheinlichkeit von 1.
Abschließend darf noch einmal darauf hingewiesen werden, dass eine Fehlervorhersage während des laufenden Betriebs nur möglich ist, wenn die Parameter (Variablen: CPUWait, MajorPageFaults, MajorPageFaultsFlagged, CPUIdle, CPUSystem, LoadOne, LoadFive, MemoryUsed, PageFaults), wie sie im vorliegenden Kapitel aufgeführt sind, ständig beobachtet und gegen die Daten aus dem erstellten bzw. vorgegebenen Modell ausgewertet werden.

6.6 Probabilistisches Model-Checking von Informationssystemen

In diesem Kapitel wird dargelegt, wie aus einem CTBN-Modell weitere für eine Fehlervorhersage relevante Informationen gewonnen werden können. Im konkreten Fall wurde für die Auswertung der Modelle das Model-Checking [157] [26], eine statische Analyse der Modelle, angewandt.
Ein realistisches Modell wurde aus einem Teil des in Kapitel 6.5 vorgestellten Datensatzes generiert. Prinzipiell hätte das in Kapitel 6.5 ermittelte Modell als Ausgangspunkt für die in diesem Kapitel vorgestellten Ansätze weiterverwendet werden können, jedoch sind im CTBN-Modell aus Kapitel 6.5 einige Redundanzen enthalten, die die weiteren Berechnungen mit den zum Zeitpunkt der Erstellung der vorliegenden Arbeit zur Verfügung stehenden Mitteln unmöglich machten. Die o. a. Redundanzen bewirkten einen übermäßigen Speicherbedarf während der Berechnungen.
Durch Anwenden der Formel 4.9 konnte der Zustandsraum, den die Originaldaten aufspannten, bestimmt werden. Die vereinigte Intensitätsmatrix $(n \times n)$ hatte für n eine Dimension der Länge 34496531784. Die für die CTBN-Modellbildung verwendeten System-Log-Daten (Monitoring-Daten) wiesen eine hohe Auflösung auf, die zu einer Vielzahl von Werte-Ausprägungen pro Variable führte und eine

dichtere Besetzung des Zustandsraumes ergab. Somit war absehbar, dass auf der Basis dieser Daten bei weiteren notwendigen Berechnungen das zur Verfügung stehende Arbeitsspeichervolumen (RAM) von 750 GB überschritten würde. Diese Beobachtungen legten den Schluss nahe, dass das in Kapitel 6.5 ermittelte Modell in der weiteren praktischen Anwendung schwer handhabbar ist.
Es war daher unumgänglich, Reduktionsmöglichkeiten zu identifizieren, die mit den vorhandenen Daten eine weitere Bearbeitung ermöglichten. Im ersten Schritt wurden deshalb Variablen, die entbehrlich waren, aus dem ursprünglichen Datensatz entfernt, da sie z.B. Sachverhalte redundant abdeckten. Dies betraf ausschließlich Variablen und deren zugehörige Messwerte. So wurde in diesem Kontext die Variable X8 $\widehat{=}$ Loadfive, welche einen gemittelten Durchschnittswert von X7 $\widehat{=}$ LoadOne darstellt, aus dem Datensatz entfernt. Diese Maßnahme schränkte den resultierenden Zustandsraum allerdings nicht genügend ein. Um eine Durchführung der Berechnungen zu ermöglichen, war eine weitere Reduktion der Daten erforderlich. Die Reduktion wurde durch empirisches Vorgehen durchgeführt, indem mit Hilfe von Domänenwissen die Anzahl der Ausprägungen von Variablen, die wenig Einfluss auf die Fehlerentstehung haben, reduziert wurde. Dies geschah durch Anwendung des in Kapitel 5.4.2 beschriebenen Algorithmus mit im Hinblick auf die Reduktion speziell angepasster Schwellwerte. Dabei ergab sich die Anzahl der jeweiligen Variableninstanzen wie folgt: CPUWait = 16, MajorPageFaults = 17, MajorPageFaultsFlagged = 2, OOM-Killer = 2, CPUIdle = 9, CPUSystem = 9, LoadOne = 29, MemoryUsed = 39, PageFaults = 8. Auch hier konnte durch das Anwenden der Formel 4.9 der Zustandsraum, den die reduzierten Daten aufspannten, bestimmt werden. Dabei ergab sich für die vereinigte Intensitätsmatrix $(n \times n)$ eine Dimension der Länge 797382144 für n. Der Zustandsraum ist nun mit deutlich geringerer Dichte besetzt.
Generell bringt eine solche Datenreduktion einen Nebeneffekt in Bezug auf die Genauigkeit mit sich. Ein Abwägen wie weit eine Reduktion sinnvoll ist und welches dadurch sich ergebende Ausmaß an Veränderungen akzeptiert werden kann, war in diesem Zusammenhang unumgänglich.
Da der zu diesem Zeitpunkt dominierende Faktor ausschließlich das Volumen des zur Verfügung stehenden Arbeitsspeichers war, wurde eine Reduktion mit der Zielsetzung durchgeführt, weitere Berechnungen zu ermöglichen. Im vorliegenden Falle wurden durch das Auslöschen ursprünglich enthaltener Ausprägungen Ungenauigkeiten beim Verhalten in Kauf genommen. Dies spiegelt sich auch im CTBN-Modell wider.
Eine solch gravierende Veränderung des Datensatzes implizierte eine Neubildung des BN-Modells. Diese Neubildung wurde wieder auf der Basis des in Kapitel 6.5 beschriebenen Datensatzes mit den zuvor erläuterten Reduzierungen durchgeführt. Das damit gebildete BN-Modell wird in Grafik 6.10 dargestellt. Für ein besseres Verständnis wird eine Zuordnung der in Grafik 6.10 dargestellten Variablen X mit den nachfolgend aufgeführten Systemgrößen gegeben: CPUWait, MajorPageFaults, MajorPageFaultsFlagged, OOM-Killer, CPUIdle, CPUSystem, LoadOne, MemoryUsed, PageFaults
Dieses Modell diente nun als Grundlage für den Amalgamationsschritt. Dieser

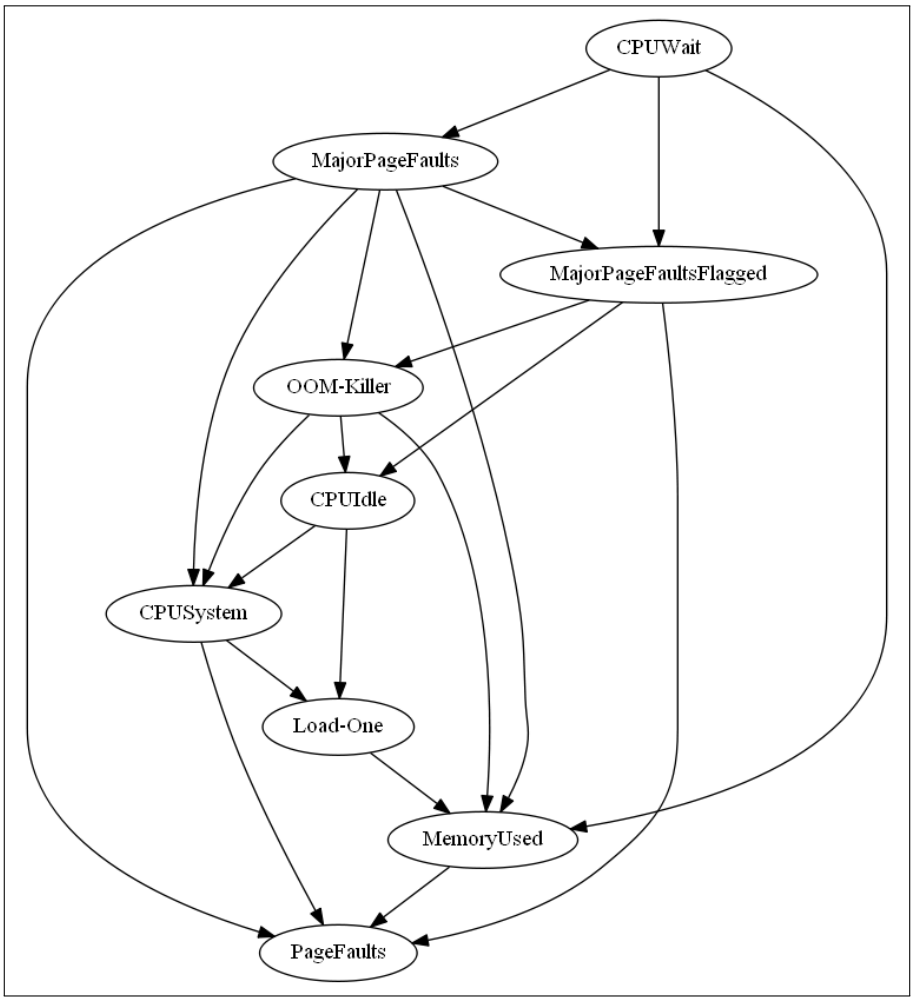


Abbildung 6.10: Bayes'sches Netzwerkmodell aus reduzierten IS Sytem-Logs

ist erforderlich, da das CTBN-Modell aus faktorisierten Markov-Prozessen besteht, welche die Dynamiken beschreiben (s. Kap 4.3.1). Nachfolgend wird die Vorgehensweise detailliert erläutert:
Im CTBN-Modell existiert für jede Variable ein Markov-Subprozess. Eine solche faktorisierte Darstellung gestattet effiziente Berechnungen und ist beim Einsatz im Online-Betrieb von Vorteil, da hier das resultierende individuelle Variablen-Verhalten mit dem beobachteten Prozess abgeglichen werden muss. Jedoch werden dadurch Aussagen über das Verhalten des Gesamtsystems erschwert bzw. sind teilweise auf Grund des fehlenden Gesamtmodells unmöglich. Ein Lösungsansatz, der das Gesamtsystemverhalten abdeckt, ist die Vereinigung aller Teil-Prozesse zu einem Gesamtmodell.

Beim CTBN werden die Teilprozesse ausschließlich durch Intensitätsmatrizen und deren entsprechende Zuordnung zu den Ausprägungen sowohl der konditionierenden als auch der resultierenden Variablen abgebildet. Es ist daher naheliegend, die im CTBN vorhandenen Intensitätsmatrizen zu einer vereinten Intensitätsmatrix zu fusionieren. Hier kommt die in Kapitel 4.3.3 beschriebene Amalgamation zum Einsatz, welche das Kronecker-Produkt der einzelnen Intensitätsmatrizen berechnet. Für die Evaluierung wurde die Amalgamation in C++ implementiert und in CTBN-RLE [128] integriert. Damit wurde die praktische Evaluierung durchgeführt, zu der das in diesem Kapitel beschriebene reduzierte CTBN-Modell den Ausgangspunkt bildete.
Die im Modell durch die konditionierenden Variablen enthaltenen Variablenabhängigkeiten und der recht große Zustandsraum (797382144) ließen dies eine berechnungsintensive Aufgabe werden. Der Rechenaufwand der Amalgamation wurde maßgeblich durch die Verarbeitung der 28424 im CTBN-Modell enthaltenen Intensitätsmatrizen, d.h. die komplette Dynamik des CTBN, bestimmt. Sehr drastisch wirkte sich dabei die Kronecker-Produkt-Berechnung aus, die u.a. eine Dimensionserweiterung der Matrizen durchführt.
Das berechnungsintensive Kronecker-Produkt wurde durch die Implementierung einer parallelisierten Matrixberechnung auf der Basis von OpenMP [104] realisiert. Der Amalgamationsschritt wurde für jede der 28424 Einzelmatrizen separat durchgeführt. Die Fusion, d.h. die Addition der Einzelergebnisse, erfolgte variablenbezogen in einem getrennten nachgelagerten Schritt, so dass für jede Variable eine einzige Matrix resultierte. Auf dieser Basis wurde durch die Fusion der neun Matrizen das Gesamtmodell gebildet, welches aus einer Matrix besteht, die alle Variablen abdeckt. Aus dem vorangegangenen Schritt resultierte somit eine Intensitätsmatrix $(n \times n)$ des Gesamtsystems, die für n eine Dimension der Länge 797382144 hatte. Sowohl die Grösse des Spaltenraums der Quadratmatrizen wie auch deren Vielzahl an Elementen erforderten eine strikte Trennung der zuvor genannten Schritte, um eine Bearbeitung zu ermöglichen. Diese Trennung war besonders bei den letzten beiden Fusionsschritten zwingend erforderlich, da trotz des sehr stark reduzierten Modells ohne Aufspaltung die zur Verfügung stehenden System-Ressourcen für die Berechnungen nicht ausgereicht hätten. Dies wurde hauptsächlich durch die große Menge der 64470375837 von Null abweichenden (non-zero) Elemente, die die Transitionen des Gesamtmodells ausmachten, beeinflusst. Trotz einer effizienten dünnbesetzten Darstellung der Matrix (Sparse), die ausschließlich die von Null abweichenden Elemente enthielt, war für die Speicherung ca. ein Terabyte an Arbeitsspeicher erforderlich.
Die Gesamtsystem-Intensitätsmatrix, die einer Markov-Kette mit kontinuierlichen Zeiteigenschaften (CTMC) entspricht, ließ Analysen und Auswertungen des vollständigen Systemverhaltens zu, allerdings war eine Weiterverarbeitung der CTMC [7], die im Arbeitsspeicher als Sparse-Gesamtsystem-Intensitätsmatrix dargestellt wurde, erst nach einem Export in ein verständliches (lesbares) Format möglich.
Der notwendige Datenexport war unproblematisch durchführbar, weil ausschließlich die von Null abweichenden Transitionsraten mit den jeweiligen zugehörigen

Zustandspaaren extrahiert werden konnten. Beim Export wird, wie in Kapitel 5.8 beschrieben, das „PRISM-Explicit-Model-Format“ verwendet, bei dem neben den Transitionen auch die Variablenbelegungen der Zustände enthalten sind.
Diese Art des Datenaustausches ermöglicht eine unkomplizierte Weiterverarbeitung der Daten mit unterschiedlichen Software-Applikationen und schafft die Voraussetzung für die vollautomatische Modellprüfung (Model-Checking). Im Fall der CTMC kann eine solche Modellprüfung mittels des probabilistischen Model-Checking [7], einer speziellen Klasse der formalen Verifikationstechniken, erreicht werden. Grundsätzlich bestehen bei Model-Checking keine Einschränkungen hinsichtlich der Größe des zu bearbeitenden Zustandsraumes. Bei der praktischen Anwendung stellt die Grösse des Zustandsraumes bei etablierten Software-Werkzeugen, z.B. PRISM [76], eine Einschränkung dar, daher wurde wie in Kapitel 4.5.2 ausgeführt der Markov Rewarded Model Checker (MRMC) [65] für die weitere Evaluierung genutzt.
Beim Model-Checking wird überprüft, ob das Modell (CTMC) gewisse zuvor spezifizierte Bedingungen erfüllt. Bei MRMC werden diese Eigenschaften mittels Formeln in Continuous Stochastic Logic (CSL) angegeben. Die aktuelle CSL-Implementierung unterstützt sowohl Zustands- wie auch Pfad-Formeln. Des Weiteren ist eine Auswertung von zeitlosen, zeitbegrenzten und zeitunbegrenzten (steady-state / long run) Eigenschaften möglich.
Bei der Frage nach der Prädiktion von Fehlern beziehen sich die oben genannten Bedingungen in erster Linie auf Fehlersituationen im Systembetrieb. Im zu Grunde liegenden Modell ist bekanntes Fehlverhalten in Form einer Variablen (OOM-Killer) enthalten. Diese Hilfsvariable signalisiert die Aktionen des OOM-Killers, der das Auftreten einer Fehlersituation modelliert. Im Rahmen der vorliegenden Studie ist die Fehlersituation ein Speicherüberlauf, d.h. es wird mehr Hauptspeicher angefordert als das Betriebssystem zur Verfügung stellen kann. Diese Präventivmaßnahme ist üblicherweise sehr selten. In einem solchen Fall ist es sinnvoll, das Problem schon bei Überschreitung eines Schwellwertes für die maximale Speicheranforderung zu signalisieren, da ein Speicherüberlauf in der Regel zu einem Betriebssystem-Ausfall und damit zur massiven Beeinträchtigung der Funktion des Gesamtsystems führt.
Im Rahmen der praktischen Model-Checking-Anwendung traten, bedingt durch die Modellgröße und die damit verbundenen Speicheranforderungen, Probleme bei der Bearbeitung auf. Eine weitere Verarbeitung der CTMC mit MRMC war ausschließlich nach einer Reduktion des Modells möglich. Es wurde daher eine systematische Modell-Bereinigung durchgeführt, indem Transitionen, d.h. Raten, mit einem Wert kleiner-gleich $1e^{-06}$ entfernt wurden. Zustandsübergänge mit sehr geringen Wahrscheinlichkeiten hatten keinen großen Einfluss auf das Systemverhalten; die Ungenauigkeit durch ihr Weglassen wurde als tolerierbar angesehen. Dieses Vorgehen implizierte, dass ausschließlich Transitionsraten mit einem Wert größer als $1e^{-06}$ berücksichtigt wurden.
Es zeigte sich, dass nach der o.a. weiteren Reduktion ein Import in MRMC und somit eine entsprechende Weiterverarbeitung möglich war. Die CTMC wies nach der Bereinigung immer noch 7746486047 Transitionen auf.

Der zuvor erwähnte Datenexport beinhaltete neben der CTMC-Repräsentation auch für alle enthaltenen Zustände die korrespondierenden Variablenbelegungen. Auf Grund der großen Anzahl an Zuständen und der damit verbundenen Unübersichtlichkeit lag es nahe, die Zustände in Abhängigkeit der Wertebelegungen mit Bezeichnern zu versehen. Im vorliegenden Fall wurden die Zustände durch eindeutige Bezeichner (INITIAL; GOOD; FAIL) beschrieben. Der beim Model-Checking erforderliche initiale Zustand (INITIAL) ergab sich aus der Belegung bei der alle Variableninstanzen Null (0) sind.
Die Fehlerzustände wurden im vorliegenden Fall durch den OOM-Killer dominiert. Es war daher naheliegend, die Variable, die das OOM-Killer-Verhalten anzeigt, mit einzubeziehen. Alle Zustände, bei denen die OOM-Killer-Variable ($X4$) den Wert 1 hatte, wurden mit dem Bezeichner FAIL versehen.
Das vorliegende Modell weist außer den OOM-Killer-Vorkommnissen keine weiteren relevanten Charakteristika auf. Daher wurden alle übrigen Zustände mit GOOD bezeichnet.
Eine solch einfache und flache Bezeichnungsstruktur kann in diesem Fall, bedingt durch die vom Modell abgedeckten Fehlerszenarien, als ausreichend angesehen werden. Des Weiteren wird durch die geringe Anzahl von Bezeichnern der Speicher-Resourcenbedarf niedrig gehalten. Eine höhere Granularität würde es sicherlich gestatten, Ergebnisse individueller zu kennzeichnen und dadurch spezifischere Analysen zu begünstigen.
Auf dieser Ausgangsbasis, der stark reduzierten CTMC mit entsprechenden Zustandsbezeichnern, wurden einige MRMC-Experimente durchgeführt. Dabei wurden die zu überprüfenden Eigenschaften mittels CTL-Formeln spezifiziert. Die Auswertung dieser Studie geschah vor dem Hintergrund der Fehlerprädiktion mit dem Ziel, die Entwicklung von Fehlern und die Pfade zu einem Fehler zu erkennen.
Exemplarisch werden nachfolgend CTL-Formeln, die solche Eigenschaften abdecken, aufgeführt.
Der Pfad, der mit einer Wahrscheinlichkeit kleiner 0.1 vom initialen Zustand zu einem Fehlerzustand führt, wird durch die Formel 6.3 beschrieben.

$$P\{< 0.1\}[\text{ INITIAL U FAIL }] \tag{6.3}$$

Im Rahmen der Evaluierung wurde die Formel 6.3 mit MRMC überprüft. Es wurde ein entsprechender Pfad des Modells gefunden, der aus einer sehr großen Zustandsfolge, die für sich alleine genommen wenig aussagekräftig war, bestand. Daher wurde darauf verzichtet, dieses Ergebnis detailliert vorzustellen.
Ein weiteres Szenario ist die Erkundung eines Pfades, der mit einer Wahrscheinlichkeit größer als 0.5 in einen Fehlerzustand mündet. Dies wird durch die Formel 6.4 spezifiziert.

$$P\{> 0.5\}[\text{ INITIAL U FAIL }] \tag{6.4}$$

Auch ein solcher Pfad wurde gefunden, jedoch wurde auf eine detaillierte Darstellung verzichtet, da der Pfad aus einer großen Zahl von Zwischenzuständen

besteht.
Es soll nicht unerwähnt bleiben, dass die vorweg beschriebenen ermittelten Zustandssequenzen mit sogenannten Entwicklungs- bzw. Verhaltensmustern gleichgesetzt werden können. Auf Grund der Tatsache, dass sie die Entwicklung in einen Fehlerzustand aufzeigen, kann auf dieser Basis die Durchführung eines Vergleichs gegen den Systembetriebszustand sinnvoll und aussagefähig sein: Sofern bei einem solchen Abgleich ein reales System während des Betriebes die einem Muster entsprechende Zustands-Charakteristik aufweist, ist es naheliegend, von einem bevorstehenden Fehlerzustand auszugehen. Dies trifft auch zu, wenn nicht der vollständige im Muster enthaltene Pfad, sondern lediglich ein repräsentativer Teil davon durchlaufen wird. Mit Hilfe dieser erweiterten Analyse lassen sind eindeutige Schlüsse auf ein bevorstehendes Fehlverhalten ziehen. Durch Auswertung der noch nicht durchlaufenen Pfadwahrscheinlichkeiten im jeweiligen Muster ist auch die Voraussage von einem Fehlverhalten bzw. einer Fehlersituation in Bezug auf Wahrscheinlichkeit und Zeit möglich.
Solche Verhaltenssequenzen ermöglichen neben der Prädiktion auch präventive Maßnahmen im laufenden Betrieb des Systems. Eine zustandsbasierte Früherkennung von anormalem Verhalten impliziert nicht unbedingt eine Fehlervermeidung, vielmehr ist dadurch die frühe Initiierung einer Gegen- bzw. Ausweich-Maßnahme möglich. Diese könnte dann auch weiterhin den reibungslosen Betrieb des Systems gewährleisten. Beispielsweise wäre eine Situation denkbar, bei der die Entstehung eines Fehlerzustandes, z.B. Crash, erkannt wird und durch Einleitung geeigneter Maßnahmen das System auf einer redundanten Instanz weiterbetrieben werden kann.

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden unter Einsatz von ausgewählten Methoden u.a. aus den Bereichen der Statistik und des maschinellen Lernens Ansätze zur Vorhersage von eventuell auftretenden Fehlern, möglichem Fehlverhalten und sich daraus ergebenden Fehlersituationen in Informationssystemen vorgestellt. Die erwähnten Methoden umfassten sowohl Bayes'schen Netze mit kontinuierlichen Zeiteigenschaften, nichtlineare Kalman-Filter, Support-Vektor-Maschinen, probabilistisches Model-Checking als auch datenbasierte Lernverfahren zur Modellbildung und eine unterraumbasierte System-Identifikation.
Zur Erkennung von Anomalien in dynamischen Daten wurde ein statistischer Anomalie-Detektor, der auf einem nichtlinearen Kalman-Filter (Unscented Kalman-Filter /UKF) basiert, vorgestellt. Die nichtlinearen Transfer- und Überwachungsfunktionen des Kalman-Filters wurden mittels einer unterraumbasierten System-Identifikation aus aufgezeichneten Daten ermittelt. Bei einer solchen datengetriebenen Art der System-Identifikation, die keine Systemkenntnisse voraussetzt, war zu bedenken, dass das berechnete Systemverhalten auch nichtlineare Eigenschaften besitzen kann. Dies wurde sowohl bei der Modellbildung in Form von Kernel-Transformationen als auch bei der Modellrepräsentation durch Support-Vektor-Maschinen berücksichtigt. Der komplette Anomalie-Detektor-Ansatz, der eine Systemidentifikation mit Modellbildung und eine statistische Auswertung beinhaltet, wurde mit synthetischen Lastaufzeichnungen von Computersystemen evaluiert. Dabei prädizierte das UKF die geschätzte Lastmenge bzw. das Lastverhalten der Computersysteme. Auf der Basis von Messwert und Schätzung konnten durch den Einsatz statistischer Tests signifikante Abweichungen des Verhaltens detektiert werden, die nicht nur die Erkennung von Fehlzuständen, sondern auch die Identifikation bzw. Extraktion von Fehlverhaltensmustern, z.B. die Entwicklung in einen Fehlerzustand, möglich machen. Solche Muster können als Datenbestand zum Zwecke einer Bewertung des Systemverhaltens entweder als Trainingsdaten einer Fehlerprädiktionsumgebung oder

als Vergleichsparameter zur Systemlaufzeit zur Fehlererkennung genutzt werden.
Des Weiteren wurde in der vorliegenden Arbeit für die gezielte Erzeugung von Fehlern eine Fehler-Instrumentierungsmethode (CGFail), die auf Linux Kontroll-Gruppen (CGroups) basiert, entwickelt. Dieser Ansatz ermöglichte es, mit eigenen Mitteln des Betriebssystems Test-Szenarien für Softwareprogramme zu erzeugen. Im Verlauf dieser Tests war es mittels der CGroups möglich, die Ressourcen, die einer Applikation zur Verfügung standen, zu beschränken bzw. anzupassen. Ein solch individuelles Ressourcen-Management, das das Betriebssystem und weitere Anwendungen nicht beeinflussen sollte, implizierte, dass die zu testende Applikation (SuT) innerhalb einer CGroup isoliert von den anderen Anwendungen (vergleichbar einer Sand-Box) betrieben wird. Wie bereits erwähnt, beinhaltet CGFail Lastgeneratoren, die während der Testausführung innerhalb der isolierten Umgebung zum Einsatz kommen können. Der Fehler-Instrumentierungsansatz CGFail wurde im Rahmen initialer Experimente analysiert. Dabei wurde sowohl der Arbeitsspeicher, der einer Applikation zur Verfügung stand, wie auch die CPU-Leistung mit Hilfe der CGroups beschränkt.
Das Kernthema der vorliegenden Arbeit, die Entwicklung eines konzeptionellen Ansatzes zur Voraussage von Fehlersituationen in Informationssystemen, wurde auf der Basis von Bayes'schen Netzen mit kontinuierlichen Zeiteigenschaften (CTBN) umgesetzt. Der CTBN-Ansatz wurde für diskrete Daten konzeptioniert, die Verarbeitung kontinuierlicher Daten war jedoch nur nach deren Diskretisierung, d.h. Konvertierung in ganze Zahlen, möglich.
Des Weiteren wurden Fehler signalisiert, die im ursprünglichen Datenbestand als Textnachricht (Event) zwar enthaltenen waren, jedoch nicht direkt mit statistischen Methoden verarbeitet werden konnten. Erst durch die Signalisierung, die durch Hilfsvariablen erfolgte, war eine Verarbeitung möglich. Diese Hilfsvariablen wurden in den Datenbestand und somit in das CTBN integriert. Eine realitätsnahe Auswertung des Ansatzes erfolgte mit Messwerten eines Server-Clusters, mit denen sowohl die Struktur als auch die Dynamiken des CTBNs erlernt wurden. Die Darstellung der CTBN-Dynamiken erfolgte in Matrixnotation als Intensitäten / Transitionsraten. Eine manuelle Analyse der Intensitätsmatrizen in Verbindung mit der Anwendung der für CTBN-Intensitäten entsprechenden Formeln ließ eine Bewertung des Zeitverhaltens zu. In diesem Kontext wurde speziell die Zeit bis zum Auftreten eines zuvor bekannten Fehlers ermittelt.
Darüber hinaus wurde die CTBN-Dynamik mit einem Amalgamationsansatz fusioniert; dies ergab eine Markov-Kette mit kontinuierlicher Zeit (CTMC). Die Fusion gestaltete sich auf Grund der Abhängigkeiten zwischen den aufgezeichneten System-Variablen und der Beschaffenheit des Datensatzes schwierig. Die vorhandenen Abhängigkeiten und die hohe Auflösung der Daten ergaben einen sehr großen Zustandsraum, welcher eine Vielzahl von Elementen enthielt. Die resultierende CTMC wurde mit einer wahrscheinlichkeitsbasierten, automatisierten Modellprüfung, einer sogenannten vollautomatischen Verifikation, dem probabilistischen Model-Checking, analysiert. Bei der Auswertung kam eine etablierte Software, der Markov Rewarded Model Checker (MRMC), zum Ein-

satz. MRMC ermittelte die Zustandsfolgen, d.h. Pfade im CTMC-Modell, über die sich ein System entwickelte, beginnend von einem initialen Zustand bis hin zu einem bekannten Fehlerzustand.
Die vorgestellte Fehlerprädiktionsumgebung nutzte eine Konvertierung zur Verarbeitung kontinuierlicher Daten, bei der die reellwertigen Variablen in ganzzahlige Werte überführt wurden. Grundsätzlich brachte diese Umwandlung einen Darstellungsfehler mit sich, der u.a. durch die Reduktion der Daten hervorgerufen wurde. Eine generalisierte Integration reellwertiger Daten in das CTBN, z.B. mit Hilfe von Wahrscheinlichkeitsdichtefunktionen, könnte eine präzisere Darstellung ermöglichen, und der Vorverarbeitungsschritt (Diskretisierung, Konditionierung) würde entfallen. Darüber hinaus wären Anwendungen des CTBN-Ansatzes im Rahmen von Fallstudien möglich, die andere als in der vorliegenden Arbeit verwendete Fehler und Fehlerszenarien beinhalten. Die Transformation solcher CTBN-Dynamiken in Markov-Ketten würde auch weiterführende probabilistische Model-Checking-Analysen ermöglichen. Der Einsatz anderer als in der vorliegenden Arbeit verwendeter Model-Checker-Software, z.B. PRISM oder statistische Model-Checker, würde u.a. Zeitanalysen (z.B. Feststellung der kürzesten Entwicklungszeit zu einem Fehler), aber auch andere wahrscheinlichkeitsbasierte Überprüfungen (z.B. Ermittlung des Pfades mit der maximalen Wahrscheinlichkeit) zulassen. Ein solches Vorgehen würde nicht nur ein größeres Ergebnisspektrum mit sich bringen, sondern könnte auch zum noch besseren Verständnis eines Systems und dessen Fehlverhalten beitragen. Hierdurch wären Ansätze zur Vermeidung von Fehlverhalten oder Systemabstürzen entwickelbar bzw. in bestehende Systeme integrierbar. Solche Maßnahmen gewährleisten üblicherweise einen langfristigen fehlerfreien und reibungslosen Betrieb von Informationssystemen.

Literaturverzeichnis

[1] Ali Abdul-Aziz, Mark Woike, Nikunj Oza, Bryan Matthews, and George Baakilini. Propulsion health monitoring of a turbine engine disk using spin test data. In *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*, pages 76501B–76501B. International Society for Optics and Photonics, 2010.

[2] Noel Abreu. Fault diagnosis with adaptive kalman filters and cmg design for picosatellite acs. Master's thesis, Ryerson University; Department of Aerospace Engineering, 2010.

[3] A. Avizienis and J.-C. Laprie. Dependable computing: From concepts to design diversity. *Proceedings of the IEEE*, 74(5):629–638, May 1986.

[4] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. Technical report, University of Maryland / Institute for Systems Research, 2004.

[5] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert K. Brayton. Verifying continuous time markov chains. In *Proceedings of the 8th International Conference on Computer Aided Verification*, CAV '96, pages 269–276, London, UK, UK, 1996. Springer-Verlag.

[6] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Softw. Eng.*, 29(6):524–541, June 2003.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[8] Gökhan H. Bakir, Jason Weston, and Bernhard Schölkopf. Learning to find pre-images. In *Advances in Neural Information Processing Systems*, pages 449–456. MIT Press, 2004.

[9] Michèle Basseville, Albert Benveniste, Maurice Goursat, Luc Hermans, Laurent Mevel, and Herman van der Auweraer. Output–only subspace–based structural identification˜: from theory to industrial testing practice.

ASME Journal of Dynamic Systems, Measurement, and Control, Special issue on Identification of Mechanical Systems, 123(4):668–676, December 2001.

[10] Bison. *Version 2.5*. GNU, 2011.

[11] Byron Boots and Geoffrey Gordon. Two-manifold problems with applications to nonlinear system identification. In *Proc. 29th Intl. Conf. on Machine Learning (ICML)*, 2012.

[12] H. Boudali and J. Bechta Dugan. A continuous-time bayesian network reliability modeling, and analysis framework. *Reliability, IEEE Transactions on*, 55(1):86–97, March 2006.

[13] Max Breitling. *Formale Fehlermodellierung für verteilte reaktive Systeme*. Dissertation, Technische Universität München, München, 2001.

[14] João Carreira, Henrique Madeira, and João Gabriel Silva. Xception: A technique for the experimental evaluation of dependability in modern computers. *IEEE Trans. Softw. Eng.*, 24:125–136, February 1998.

[15] João Carreira, Henrique Madeira, and João Gabriel Silva. Xception: Software fault injection and monitoring in processor functional units. In *Conference on Dependable Computing for Critical Applications, DCCA-5, Urbana-Champaign, Illinois,*, 1995.

[16] J.V. Carreira, D. Costa, and J.G. Silva. Fault injection spot-checks computer system dependability. *Spectrum, IEEE*, 36(8):50–55, Aug 1999.

[17] Robert Chase. How to configure the linux out-of-memory killer. `http://www.oracle.com/technetwork/articles/servers-storage-dev/oom-killer-1911807.html`, February 2013.

[18] Jie Cheng, Russell Greiner, Jonathan Kelly, David Bell, and Weiru Liu. Learning bayesian networks from data: An information-theory based approach. *Artif. Intell.*, 137(1-2):43–90, May 2002.

[19] David Maxwell Chickering. Learning equivalence classes of bayesian-network structures. *J. Mach. Learn. Res.*, 2:445–498, March 2002.

[20] David Maxwell Chickering. Optimal structure identification with greedy search. *J. Mach. Learn. Res.*, 3:507–554, March 2003.

[21] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of bayesian networks is np-hard. *J. Mach. Learn. Res.*, 5:1287–1330, December 2004.

[22] DavidMaxwell Chickering. Learning bayesian networks is np-complete. In Doug Fisher and Hans-J. Lenz, editors, *Learning from Data*, volume 112 of *Lecture Notes in Statistics*, pages 121–130. Springer New York, 1996.

[23] T.-J. Chin and D. Suter. Incremental kernel pca for efficient non-linear feature extraction. In *Proceedings of the British Machine Vision Conference*, pages 96.1–96.10. BMVA Press, 2006.

[24] Tat-Jun Chin, Konrad Schindler, and David Suter. Incremental kernel svd for face recognition with image sets. In *Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition*, FGR '06, pages 461–466, Washington, DC, USA, 2006. IEEE Computer Society.

[25] A. Choi, A. Darwiche, L. Zheng, and O. J. Mengshoel. Data mining in systems health management: Detection, diagnostics, and prognostics. In A. Srivastava and J. Han, editors, *Machine Learning and Knowledge Discovery for Engineering Systems Health Management*, chapter A Tutorial on Bayesian Networks for System Health Management. Chapman and Hall/CRC Press, 2011.

[26] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[27] Barry R. Cobb, Rafael Rumi, and Antonio Salmeron. Bayesian network models with discrete and continuous variables. In *Advances in Probabilistic Graphical Models*, pages 81–102. Springer Berlin Heidelberg, 2007.

[28] Gregory F. Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9(4):309–347, October 1992.

[29] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[30] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, December 1989.

[31] Richard Dearden and Brenda Ng. Hybrid system diagnosis with parameter estimation using particle filters. In *Workshop on Planning, Learning and Monitoring with Uncertain and Dynamic Worlds; ECAI 2006*, 2006.

[32] Yu Fan. *Continuous Time Bayesian Network Approximate Inference and Social Network Applications*. PhD thesis, University of California Ű Riverside, 2009.

[33] Yu Fan and Christian R. Shelton. Learning continuous-time social network dynamics. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 161–168, Arlington, Virginia, United States, 2009. AUAI Press.

[34] Yu Fan, Jing Xu, and Christian R. Shelton. Importance sampling for continuous time bayesian networks. *J. Mach. Learn. Res.*, 11:2115–2140, August 2010.

[35] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027, 1993.

[36] Antonio Fernández, Helge Langseth, Thomas Nielsen, and Antonio Salmerón. Parameter learning in mte networks using incomplete data. In *Proceedings of The Fifth European Workshop on Probabilistic Graphical Models (PGM 2010)*, pages 137–144., 2010.

[37] Nir Friedman, Iftach Nachman, and Dana Peér. Learning bayesian network structure from massive datasets: The «sparse candidate« algorithm. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 206–215, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[38] Felix Christoph Gärtner. *Formale Grundlagen der Fehlertoleranz in verteilten Systemen*. PhD thesis, TU Darmstadt, Juli 2001.

[39] ELENA GATTI. *Graphical models for continuous time inference and decision making*. PhD thesis, Università degli Studi di Milano-Bicocca, Dottorato di ricerca in INFORMATICA, 23, 2011-02-08.

[40] Ivan Goethals, Luc Hoegaerts, Vincent Verdult, Johan A.K. Suykens, and Bart De Moor. Subspace identication of hammerstein-wiener systems using kernel canonical correlation analysis. *Internal Report 05-46, ESAT-SISTA, K.U.Leuven (Leuven, Belgium) Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference*, 44th:7108 – 7113, 2004.

[41] Ivan Goethals, Laurent Mevel, Albert Benveniste, and Bart De Moor. Recursive output only subspace identification for in-flight flutter monitoring. In *Proceedings of the 22nd International Modal Analysis Conference IMACXXII*, 2004.

[42] Mel Gorman. Understanding the linux virtual memory manager. `https://www.kernel.org/doc/gorman/html/understand/understand016.html`, July 2007.

[43] Jim Gray. Why do computers stop and what can be done about it? In *Symposium on Reliability in Distributed Software and Database Systems (SRDS-5)*, pages 3–12, Los Angeles, CA, 1986. IEEE CS Press.

[44] Martin Groessl. Cgfail: A new approach for simulating computer systems faults. In *DEPEND*, 2012.

[45] Martin Groessl. An unscented kalman filter based statistical failure detector. In *Control and Fault-Tolerant Systems (SysTol), 2013 Conference on*, pages 401–406, Oct 2013.

[46] HammerDB. Hammerdb, version 2.14. http://hammerora.sourceforge.net/, 2013.

[47] B. Haverkort, L. Cloth, H. Hermanns, J. Katoen, and C. Baier. Model checking performability properties. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 103–112, 2002.

[48] Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the use of model checking techniques for quantitative dependability evaluation. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems, SRDS 2000*, pages 228–237, Los Alamitos, 2000. IEEE Computer Society Press.

[49] Ralf Herbrich, Thore Graepel, and Brendan Murphy. Structure from failure. In *Proceedings of the 2Nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, SYSML'07, pages 10:1–10:6, Berkeley, CA, USA, 2007. USENIX Association.

[50] Hoffmann. *Failure Prediction in Complex Computer Systems*. Dissertation, Humboldt-Universität Berlin, 2005.

[51] Oliver C. Ibe. *Markov processes for stochastic modeling*. Elsevier, 2nd ed. edition, 2013.

[52] David Iverson and Ryan Mackey. Integrated system health management (ishm) technology demonstration project. Technical report, NASA Ames Research Center, 2005.

[53] David L. Iverson. Inductive system health monitoring. In *Proceedings of the International Conference on Artificial Intelligence, IC-AI '04, Volume 2 & Proceedings of the International Conference on Machine Learning; Models, Technologies & Applications, MLMTA '04, June 21-24, 2004, Las Vegas, Nevada, USA*, pages 605–611, 2004.

[54] D.L. Iverson. System health monitoring for space mission operations. In *Aerospace Conference, 2008 IEEE*, pages 1–8, March 2008.

[55] D.N. Jansen, J.P. Katoen, M. Oldenkamp, M.I.A. Stoelinga, and I.S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In K. Yohav, editor, *Hardware and Software: Verification and Testing, Proceedings of the Third International Haifa Verification Conference, HVC 2007*, volume 4899 of *Lecture Notes in Computer Science*, pages 69–85, London, February 2008. Springer Verlag.

[56] A. Jazwinski. Filtering for nonlinear dynamical systems. *Automatic Control, IEEE Transactions on*, 11(4):765–766, Oct 1966.

[57] A.H. Jazwinski. *Stochastic Processes and Filtering Theory*. Mathematics in science and engineering. Academic Press, 1970.

[58] S. Julier, J. Uhlmann, and H.F. Durrant-Whyte. A new method for the nonlinear transformation of means and covariances in filters and estimators. *Automatic Control, IEEE Transactions on*, 45(3):477–482, Mar 2000.

[59] Simon J. Julier and Jeffrey K. Uhlmann. Reduced sigma point filters for the propagation of means and covariances through nonlinear transformations. In *In Proceedings of the 2002 American Control Conference*, pages 887–892, 2002.

[60] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, Mar 2004.

[61] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham. Ferrari: A flexible software-based fault and error injection system. *IEEE Transactions on Computers*, 44(2):248 – 260, 1995.

[62] G.A. Kanawati, N.A. Kanawati, and J.A. Abraham. Ferrari: a flexible software-based fault and error injection system. *Computers, IEEE Transactions on*, 44(2):248–260, Feb 1995.

[63] Joost-Pieter Katoen. Modeling and verification of probabilistic systems lecture 15: Transient analysis of ctmcs. VTSA Summerschool, Liège, Belgium, September 2011.

[64] Joost-Pieter Katoen. Modeling and verification of probabilistic systems. Software Modeling and Verification Group (RWTH AACHEN), April 17 2014.

[65] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. A markov reward model checker. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, QEST '05, pages 243–, Washington, DC, USA, 2005. IEEE Computer Society.

[66] Guillaume Mercere Katrien De Cock and Bart De Moor. Recursive subspace identification for in-flight modal analysis of airplanes. *22nd International Modal Analysis Conference (IMAC-XXII), Dearborn, Michigan*, 22nd:1563–1578, 2004.

[67] N. Keshava. Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries. *Geoscience and Remote Sensing, IEEE Transactions on*, 42(7):1552–1565, July 2004.

[68] Florian Knorn and J. Douglas Leith. Adaptive kalman filtering for anomaly detection in software appliances. *IEEE Conference on Computer Communications INFOCOM (Workshops)*, pages 1–6, 2008.

[69] Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Auton. Robots*, 27(1):75–90, July 2009.

[70] Jonathan Ko, J. Daniel Klein, Dieter Fox, and Dirk Hähnel. Gp-ukf: Unscented kalman filters with gaussian process prediction and observationmodels. In *IROS*, pages 1901–1907, 2007.

[71] Alexander V. Kozlov and Daphne Koller. Nonuniform dynamic discretization in hybrid networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, pages 314–325, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[72] Minjia Krüger, Torsten Jeinsch, Peter Engel, and Steven X. Ding. Zustandsüberwachung und performanzprognose. *atp edition*, 56(10):42–51, 2014.

[73] Minjia Krueger, Adel Haghani Abandan Sari, Steven X Ding, Torsten Jeinsch, and Peter Engel. A data-driven maintenance support system for wind energy conversion systems. In *19th IFAC World Congress*, volume 19. International Federation of Automatic Control, 2014.

[74] Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, Ltd., London, UK, UK, 1995.

[75] M. Kwiatkowska, G. Norman, and D. Parker. Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 15(11):1427–1434, 2006.

[76] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[77] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Proceedings of the 7th International Conference on Formal Methods for Performance Evaluation*, SFM'07, pages 220–270, Berlin, Heidelberg, 2007. Springer-Verlag.

[78] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic model checking for performance and reliability analysis. *SIGMETRICS Perform. Eval. Rev.*, 36(4):40–45, March 2009.

[79] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Symmetry reduction for probabilistic model checking. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 234–248, 2006.

[80] J. T.-Y. Kwok and I. W.-H. Tsang. The pre-image problem in kernel methods. *Trans. Neur. Netw.*, 15(6):1517–1525, November 2004.

[81] Helge Langseth, Thomas D. Nielsen, Rafael Rumi, and Antonio Salmeron. Mixtures of truncated basis functions. *International Journal of Approximate Reasoning*, 53(2):212 – 227, 2012. Theory of Belief Functions (BELIEF 2010).

[82] Helge Langseth, Thomas D. Nielsen, Rafael Rumi, and Antonio Salmeron. Mixtures of truncated basis functions. In *The Sixth European Workshop on Probabilistic Graphical Models*, 2012.

[83] Helge Langseth, Thomas Dyhre Nielsen, and Antonio Salmeron. Learning mixtures of truncated basis functions from data. In *The Sixth European Workshop on Probabilistic Graphical Models*, 2012.

[84] Jean-Claude Laprie and Karama Kanoun. Software reliability and system reliability. In Michael R. Lyu, editor, *Handbook of Software Reliability Engineering*, pages 27–69. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[85] U.N. Lerner. *Hybrid Bayesian Networks for Reasoning about Complex Systems*. PhD thesis, Stanford University. Computer Science Dept, 2002.

[86] libev. *Version 4.04*. Marc Lehmann, schmorp.de, 2011.

[87] libevent. *Version 2.1*. Nick Mathewson and Niels Provos, 2012.

[88] Jiang Liu and Mingquan Lu. An adaptive ukf filtering algorithm for gps position estimation. *International Conference on Wireless Communications, Networking and Mobile Computing*, 5th:1 – 4, 2009.

[89] Yilu Liu, StantonHadley, Joe Gracia, Travis Smith, and Isabelle Snyder. Development of dynamic models & tools for interconnection wide simulations. Advanced Grid Modeling Peer Review Presentation, June 2014.

[90] Chin-Hsiung Loh, Jerome P. Lynch, Kung-Chun Lu, Yang Wang, and Ping-Yin Lin. Output-only modal identification of a cable-stayed bridge using wireless monitoring systems,. *Engineering Structures, Elsevier*, 30(7)::1820–1830, 2008.

[91] David Marquez, Martin Neil, and Norman Fenton. Improved reliability modeling using bayesian networks and dynamic discretization. *Reliability Engineering & System Safety*, 95(4):412 – 425, 2010.

[92] MATLAB. *version 7.14.0 (R2012a)*. The MathWorks Inc., 2012.

[93] P.M. Melliar-Smith and B. Randell. Software reliability: The role of programmed exception handling. In SantoshKumar Shrivastava, editor, *Reliable Computer Systems*, Texts and Monographs in Computer Science, pages 143–153. Springer Berlin Heidelberg, 1985.

[94] Andrew Moore and Weng keen Wong. Optimal reinsertion: A new search operator for accelerated and more accurate bayesian network structure learning. In *In Proceedings of the 20th International Conference on Machine Learning (ICML Š03*, pages 552–559. AAAI Press, 2003.

[95] Serafín Moral, Rafael Rumí, and Antonio Salmerón. Mixtures of truncated exponentials in hybrid bayesian networks. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, ECSQARU '01, pages 156–167, London, UK, UK, 2001. Springer-Verlag.

[96] Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UNIVERSITY OF CALIFORNIA, BERKELEY, 2002. AAI3082340.

[97] Martin Neil, Manesh Tailor, David Marquez, Norman Fenton, and Peter Hearty. Modelling dependable systems using hybrid bayesian networks. *Reliability Engineering & System Safety*, 93(7):933 – 939, 2008. Bayesian Networks in Dependability.

[98] Brenda Ng, Avi Pfeffer, and Richard Dearden. Continuous time particle filtering. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1360–1365, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[99] U. Nodelman, C.R. Shelton, and D. Koller. Continuous time bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 378–387, 2002.

[100] Uri Nodelman and Eric Horvitz. Continuous time bayesian networks for inferring usersŠ presence and activities with extensions for modeling and evaluation. Technical report, Microsoft Research, 2003.

[101] Uri Nodelman, Christian R. Shelton, and Daphne Koller. Learning continuous time Bayesian networks. In *Proceedings of the Nineteenth International Conference on Uncertainty in Artificial Intelligence*, pages 451–458, 2003.

[102] Uri D. Nodelman. *CONTINUOUS TIME BAYESIAN NETWORKS*. PhD thesis, STANFORD UNIVERSITY, 2007.

[103] J.R. Norris. *Markov Chains*. Number Nr. 2008 in Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.

[104] OpenMP Architecture Review Board. OpenMP application program interface version 3.1, July 2011.

[105] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[106] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA, 2000.

[107] Zhaoguang Peng, Yu Lu, Alice Miller, Chris Johnson, and Tingdi Zhao. A probabilistic model checking approach to analysing reliability, availability, and maintainability of a single satellite system. In *Proceedings of the 2013 European Modelling Symposium*, EMS '13, pages 611–616, Washington, DC, USA, 2013. IEEE Computer Society.

[108] Frank Puppe. *Einführung in Expertensysteme*. Springer, 1991.

[109] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[110] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[111] MichaelM. Richter and RosinaO. Weber. Basic similarity topics. In *Case-Based Reasoning*, pages 113–147. Springer Berlin Heidelberg, 2013.

[112] Vanessa Romero, Rafael Rumí, and Antonio Salmerón. Learning hybrid bayesian networks using mixtures of truncated exponentials. *Int. Journal of Approximate Reasoning*, 42(1-2):54–68, 2006.

[113] David Di Ruscio. Combined deterministic and stochastic system identification and realization: Dsr - a subspace approach based on observations. *Modeling, Identification and Control*, No. 17:pp. 193–230, 1996.

[114] David Di Ruscio. Closed and open loop subspace system identification of the kalman filter. *Modeling, Identification and Control*, 30(2):71–86, 2009.

[115] Evan Russell, Leo H Chiang, and Richard D Braatz. *Data-driven methods for fault detection and diagnosis in chemical processes*. Springer, 2000.

[116] Sage A. and HUSA G. W. Adaptive filtering with unknown prior statistics. *Joint Automatic Control Conference,Boulder, USA: American Society of Mechanical Engineers,*, pages 760–769, 1969.

[117] B. Saha, K. Goebel, S. Poll, and J. Christophersen. An integrated approach to battery health monitoring using bayesian regression and state estimation. In *Autotestcon, 2007 IEEE*, pages 646–653, Sept 2007.

[118] Felix Salfner. *Event-based Failure Prediction*. Dissertation, Humboldt-Universität Berlin, 2008.

[119] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.

[120] J. Schumann, O.J. Mengshoel, and T. Mbaya. Integrated software and sensor health management for small spacecraft. In *Space Mission Challenges for Information Technology (SMC-IT), 2011 IEEE Fourth International Conference on*, pages 77–84, Aug 2011.

[121] Johann Schumann, Ole J. Mengshoel, and Adnan Darwiche. Iswhm: Tools and techniques for software and system health management. Technical report, NASA Ames Research Center; Moffett Field, CA, United States, 2010.

[122] Johann Schumann, Ashok N. Srivastava, Ole J. Mengshoel, and Adnan Darwiche. Towards software health management with bayesian networks. In *Future of Software Engineering Research (FoSER 2010)*, 2010.

[123] Marco Scutari. Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

[124] J. Seitz. *Digitale Sprach- und Datenkommunikation: Netze - Protokolle - Vermittlung ; mit ... 24 Tabellen und 97 Aufgaben.* Fachbuchverl. Leipzig im Carl-Hanser-Verlag, 2007.

[125] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004.

[126] Christian Shelton and Gianfranco Ciardo. Tutorial on continuous-time markov processes. *Conference on Uncertainty in Artificial Intelligence*, page 64, 2012.

[127] Christian R. Shelton and Gianfranco Ciardo. Tutorial on structured continuous-time markov processes. *J. Artif. Intell. Res. (JAIR)*, 51:725–778, 2014.

[128] Christian R. Shelton, Yu Fan, William Lam, Joon Lee, and Jing Xu. Continuous time bayesian network reasoning and learning engine. *Journal of Machine Learning Research*, 11:1137–1140, 2010.

[129] Yong Shi, Chongzhao Han, and Yongqi Liang. Adaptive ukf for target tracking with unknown process noise statistics. *International Conference on Information Fusion, FUSION '09*, 12th:1815 – 1820, 2009.

[130] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems - design and evaluation (3. ed.).* A K Peters, 1998.

[131] Balbir Singh and Vaidyanathan Srinivasan. Containers: Challenges with the memory resource controller and its performance. *Ottawa Linux Symposium*, 2:209–222, 2007.

[132] AlexJ. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[133] Isabelle Snyder and Travis Smith. Dynamic protection - planning simulation. EPRI Grid Ops & Planning Advisory Meeting Presentation, September 10 2014.

[134] L. Song, B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola. Hilbert space embeddings of hidden Markov models. In *Proc. 27th Intl. Conf. on Machine Learning (ICML)*, 2010.

[135] Qi Song and Yuqing He. Adaptive unscented kalman filter for estimation of modelling errors for helicopter. In *Proceedings of the 2009 international conference on Robotics and biomimetics*, ROBIO'09, pages 2463–2467, Piscataway, NJ, USA, 2009. IEEE Press.

[136] P. Spirtes, C.N. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Adaptive computation and machine learning. MIT Press, 2000.

[137] A.N. Srivastava and J. Schumann. The case for software health management. In *Space Mission Challenges for Information Technology (SMC-IT), 2011 IEEE Fourth International Conference on*, pages 3–9, Aug 2011.

[138] Ashok N. Srivastava. Discovering system health anomalies using data mining techniques. In *Proceedings of the Joint Army Navy NASA Air Force Conference on Propulsion*, 2005.

[139] Ashok N. Srivastava. Integrated vehicle health management. Technical report, NASA, 2009.

[140] Ashok N. Srivastava, Claudia Meyer, and Robert W. Mah. *Encyclopedia of Aerospace Engineering*, chapter In-Flight Vehicle Health Management. John Wiley & Sons, Ltd, 2010.

[141] J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*, volume ISBN 981-238-151-1. World Scientific, Singapore, 2002.

[142] Johan Suykens. Support vector machines and kernel based learning. *International Conference on Artificial Neural Networks*, 2007.

[143] TimothyK. Tsai and RavishankarK. Iyer. Measuring fault tolerance with the ftape fault injection tool. In Heinz Beilner and Falko Bause, editors, *Quantitative Evaluation of Computing and Communication Systems*, volume 977 of *Lecture Notes in Computer Science*, pages 26–40. Springer Berlin Heidelberg, 1995.

[144] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Mach. Learn.*, 65(1):31–78, October 2006.

[145] Rudolph Van Der Merwe. *Sigma-point Kalman Filters for Probabilistic Inference in Dynamic State-space Models*. PhD thesis, Oregon Health & Science University, 2004. AAI3129163.

[146] Peter Van Overschee and Bart De Moor. N4sid: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1):75–93, January 1994.

[147] Vmstat. Vmstat. man page,`http://www.linuxcommand.org/man_pages/vmstat8.html`, July 1994.

[148] Dirk Vorberg and Sven Blankenberger. Die auswahl statistischer tests und maSSe. *Psychologische Rundschau*, 50(3):157–164, 1999.

[149] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient software-based fault isolation. *SIGOPS Oper. Syst. Rev.*, 27(5):203–216, December 1993.

[150] E.A Wan and R. Van der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158, 2000.

[151] Eric A. Wan and Rudolph van der Merwe. *The Unscented Kalman Filter*, pages 221–280. John Wiley & Sons, Inc., 2002.

[152] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.

[153] L. Wolf and A. Shashua. Kernel principal angles for classification machines with applications to image sequence interpretation. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–635–I–640 vol.1, June 2003.

[154] Lior Wolf, Amnon Shashua, and Donald Geman. Learning over sets using kernel principal angles. *Journal of Machine Learning Research*, 4:2003, 2003.

[155] Jing Xu and Christian R. Shelton. Intrusion detection using continuous time bayesian networks. *J. Artif. Int. Res.*, 39(1):745–774, September 2010.

[156] Jing Xu and ChristianR. Shelton. Continuous time bayesian networks for host level network intrusion detection. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Computer Science*, pages 613–627. Springer Berlin Heidelberg, 2008.

[157] Ivan S. Zapreev. *Model checking Markov chains : techniques and tools*. PhD thesis, University of Twente, Enschede, March 2008.

Anhang A

Verwendete Abkürzungen

BN Bayessche Netze

CTBN Bayessche Netze mit kontinuierlicher Zeit

CTBN-RLE Continuous Time Bayesian Network Reasoning and Learning Engine

DBN Dynamischen Bayesschen Netze

DAG azyklisch gerichteter Graph

CPD bedingte Wahrscheinlichkeitsverteilung

ML Maximum-Likelihood

MMHC Max-Min Hill-Climbing

MMPC Max-Min Parents and Children

CPC PC-Kandidaten / CandidatesParenstChildren

CI Konfidenzintervall

CIM konditionierte Intensitäts-Matrix

DB Datenbank

HSMM Hidden-Semi-Markov-Modelle

RBF radialen Basisfunktions-Kernel

CSL kontinuierlichen stochastischen Logik

CSRL Continuous Stochastic Reward Logic

PCTL Probabilistic Computation Tree Logic

MTBF Mean Time Between Failures

EKF Erweiterte-Kalman-Filter

CTMC Markov-Kette mit kontinuierlicher Zeit

HMP Hidden-Markov-Prozess

DTMC zeitdiskreten Markov-Ketten

MRMC Markov Rewarded Model Checker

UKF Unscented Kalman Filter

DSR Kombinierten deterministischen und stochastischen System- und Realisierungs- Algorithmus

SVD Singulärwertzerlegung

SAM Spectral Angle Mapper

SVR Support-Vektor-Maschinen-Regression

LS-SVM Least-Square-Support-Vector-Machine

GP Gaußschen-Prozessen

SVM Support Vektor Maschinen

PCA Hauptkomponentenanalyse

IS Informationssystemen

K-SVD Kernel Singulär-Wertzerlegung

JPD vereinigte Wahrscheinlichkeitsverteilung

GOF Goodness-of-Fit Test

AUKF adaptive UKF

MTE Mischung aus beschränkten Exponentialfunktionen

MoTBF Mischung aus beschränkten Basis-Funktionen

PDF Wahrscheinlichkeitsdichtefunktion

OOM Out-Of-Memory

OS Betriebssystem

SuT System under Test

Anhang B

Amalgamation-Matrizen

−34.81	0	0	0	0	0	34.81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7.46	0	0	0	0	0	−22.38	0	0	0	0	0	14.92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4.28	0	0	0	0	0	12.84	0	0	0	0	0	−17.12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−34.81	0	0	0	0	0	34.81	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7.46	0	0	0	0	0	−22.38	0	0	0	0	0	14.92	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4.28	0	0	0	0	0	12.84	0	0	0	0	0	−17.12	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Amalgemations-Matrix $X3,0$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	−26.68	0	0	0	0	0	22.24	0	0	0	0	0	4.45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	−35.94	0	0	0	0	0	35.94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	11.72	0	0	0	0	0	7.82	0	0	0	0	0	−19.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−26.68	0	0	0	0	0	22.24	0	0	0	0	0	4.45	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−35.94	0	0	0	0	0	35.94	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11.72	0	0	0	0	0	7.82	0	0	0	0	0	−19.54	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Amalgemations-Matrix $X3,1$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	−22.73	0	0	0	0	0	15.15	0	0	0	0	0	7.58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	10.26	0	0	0	0	0	−35.90	0	0	0	0	0	25.64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	17.35	0	0	0	0	0	4.34	0	0	0	0	0	−21.68	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−22.73	0	0	0	0	0	15.15	0	0	0	0	0	7.58	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10.26	0	0	0	0	0	−35.90	0	0	0	0	0	25.64	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17.35	0	0	0	0	0	4.34	0	0	0	0	0	−21.68	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Amalgemations-Matrix $X3,2$

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -9.90 & 0 & 0 & 0 & 0 & 0 & 4.95 & 0 & 0 & 0 & 0 & 0 & 4.95 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 16.79 & 0 & 0 & 0 & 0 & 0 & -22.38 & 0 & 0 & 0 & 0 & 0 & 5.60 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9.90 & 0 & 0 & 0 & 0 & 0 & 4.95 & 0 & 0 & 0 & 0 & 0 & 4.95 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16.79 & 0 & 0 & 0 & 0 & 0 & -22.38 & 0 & 0 & 0 & 0 & 0 & 5.60 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Amalgemations-Matrix $X3,3$

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -29.66 & 0 & 0 & 0 & 0 & 0 & 22.25 & 0 & 0 & 0 & 0 & 0 & 7.42 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 10.04 & 0 & 0 & 0 & 0 & 0 & -13.39 & 0 & 0 & 0 & 0 & 0 & 3.35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 13.89 & 0 & 0 & 0 & 0 & 0 & 2.78 & 0 & 0 & 0 & 0 & 0 & -16.67 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -29.66 & 0 & 0 & 0 & 0 & 0 & 22.25 & 0 & 0 & 0 & 0 & 0 & 7.42 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10.04 & 0 & 0 & 0 & 0 & 0 & -13.39 & 0 & 0 & 0 & 0 & 0 & 3.35 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13.89 & 0 & 0 & 0 & 0 & 0 & 2.78 & 0 & 0 & 0 & 0 & 0 & -16.67 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

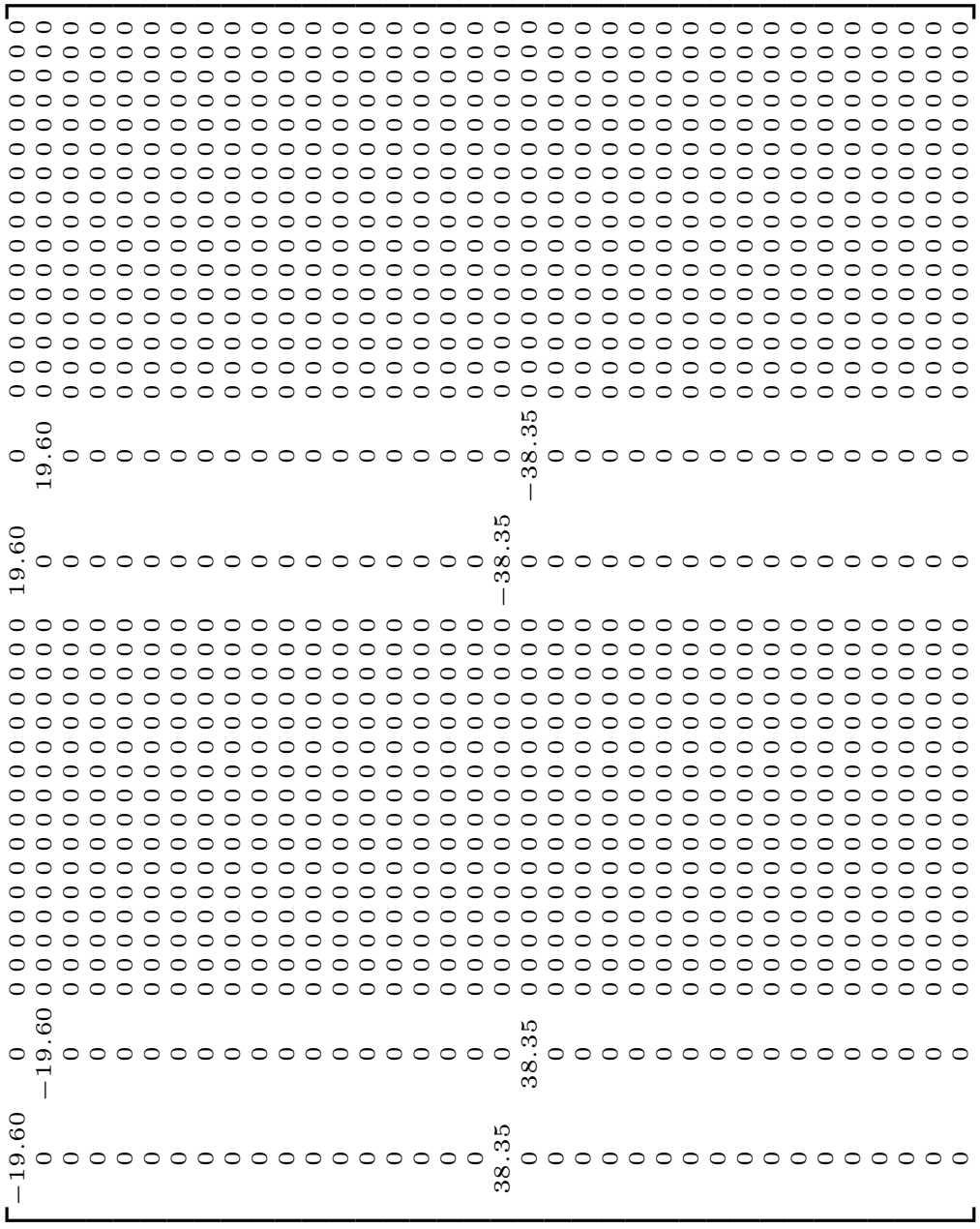
Amalgemations-Matrix $X3, 4$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -29.66 & 0 & 0 & 0 & 0 & 0 & 22.25 & 0 & 0 & 0 & 0 & 0 & 7.42 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 10.04 & 0 & 0 & 0 & 0 & 0 & -13.39 & 0 & 0 & 0 & 0 & 0 & 3.35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 13.89 & 0 & 0 & 0 & 0 & 0 & 2.78 & 0 & 0 & 0 & 0 & 0 & -16.67 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -29.66 & 0 & 0 & 0 & 0 & 0 & 22.25 & 0 & 0 & 0 & 0 & 0 & 7.42 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10.04 & 0 & 0 & 0 & 0 & 0 & -13.39 & 0 & 0 & 0 & 0 & 0 & 3.35 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13.89 & 0 & 0 & 0 & 0 & 0 & 2.78 & 0 & 0 & 0 & 0 & 0 & -16.67 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

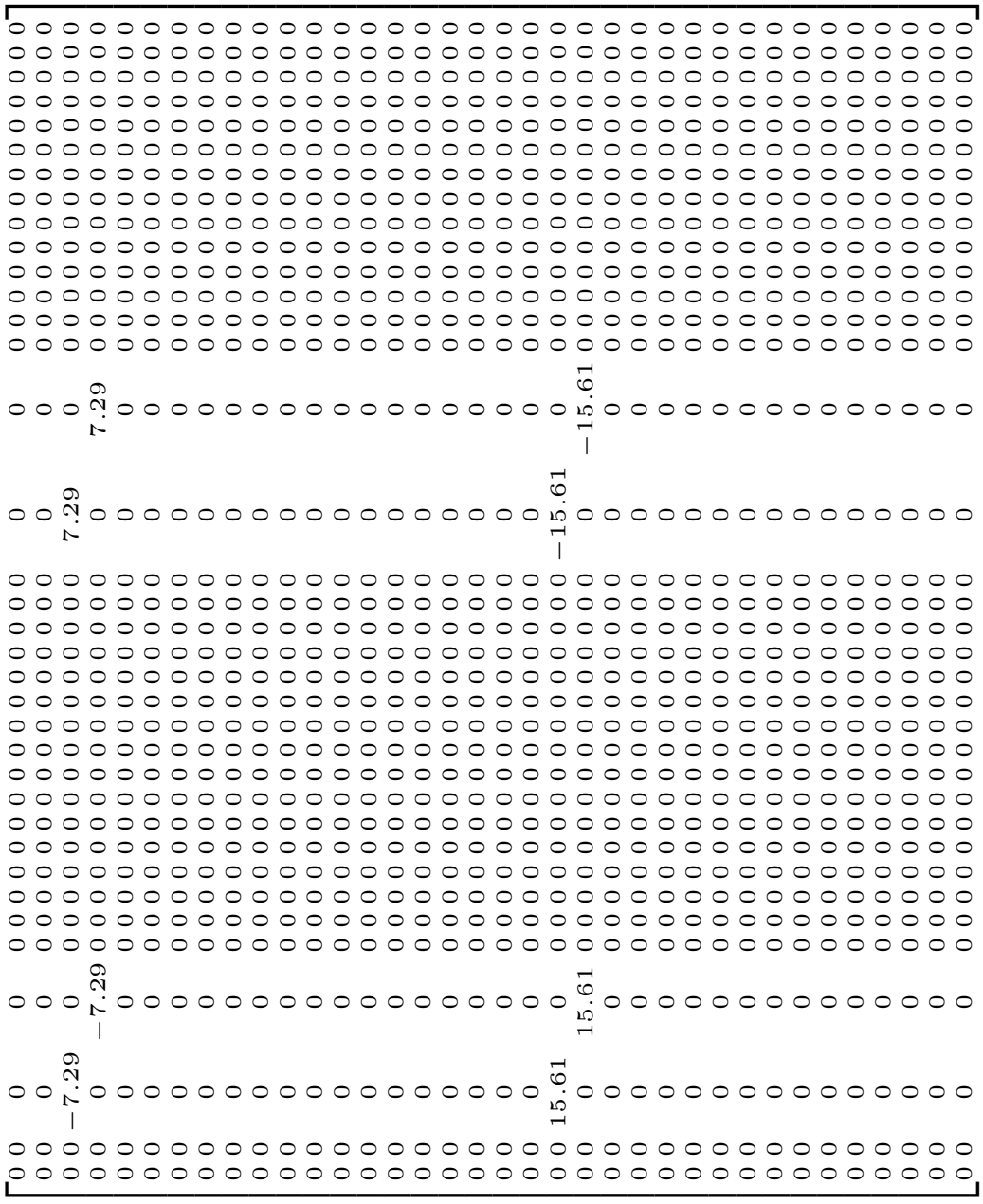
Amalgemations-Matrix $X3, 4$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	−139.86	0	0	0	0	0	139.86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	14.37	0	0	0	0	0	−35.92	0	0	0	0	0	21.55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	52.80	0	0	0	0	0	−52.80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−139.86	0	0	0	0	0	139.86	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14.37	0	0	0	0	0	−35.92	0	0	0	0	0	21.55
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52.80	0	0	0	0	0	−52.80

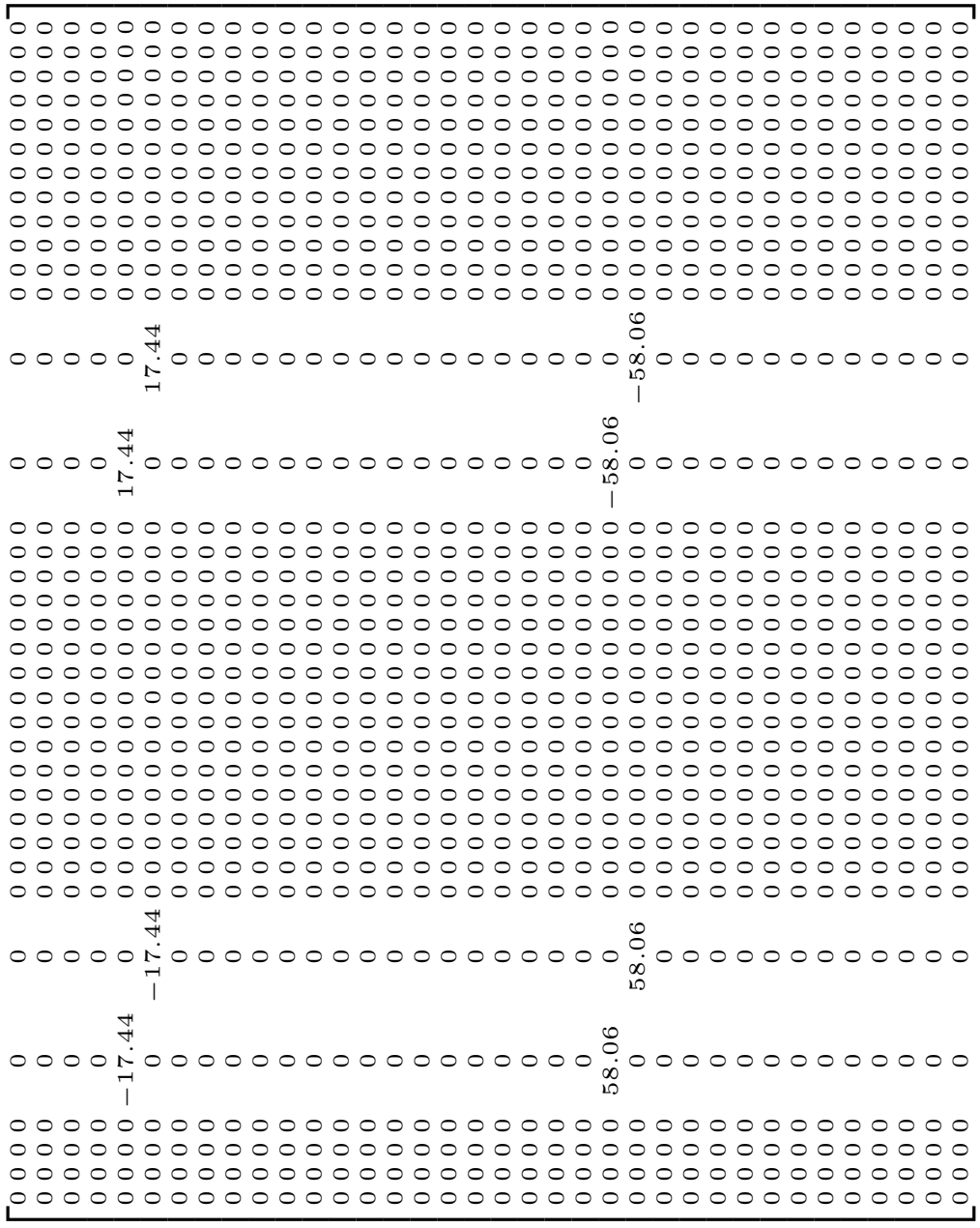
Amalgemations-Matrix $X3,5$



Amalgemations-Matrix $X4,0$



Amalgemations-Matrix $X4,1$



Amalgemations-Matrix $X4,2$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	−35.36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35.36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	−35.36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35.36	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	13.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.14	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	13.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−13.14	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

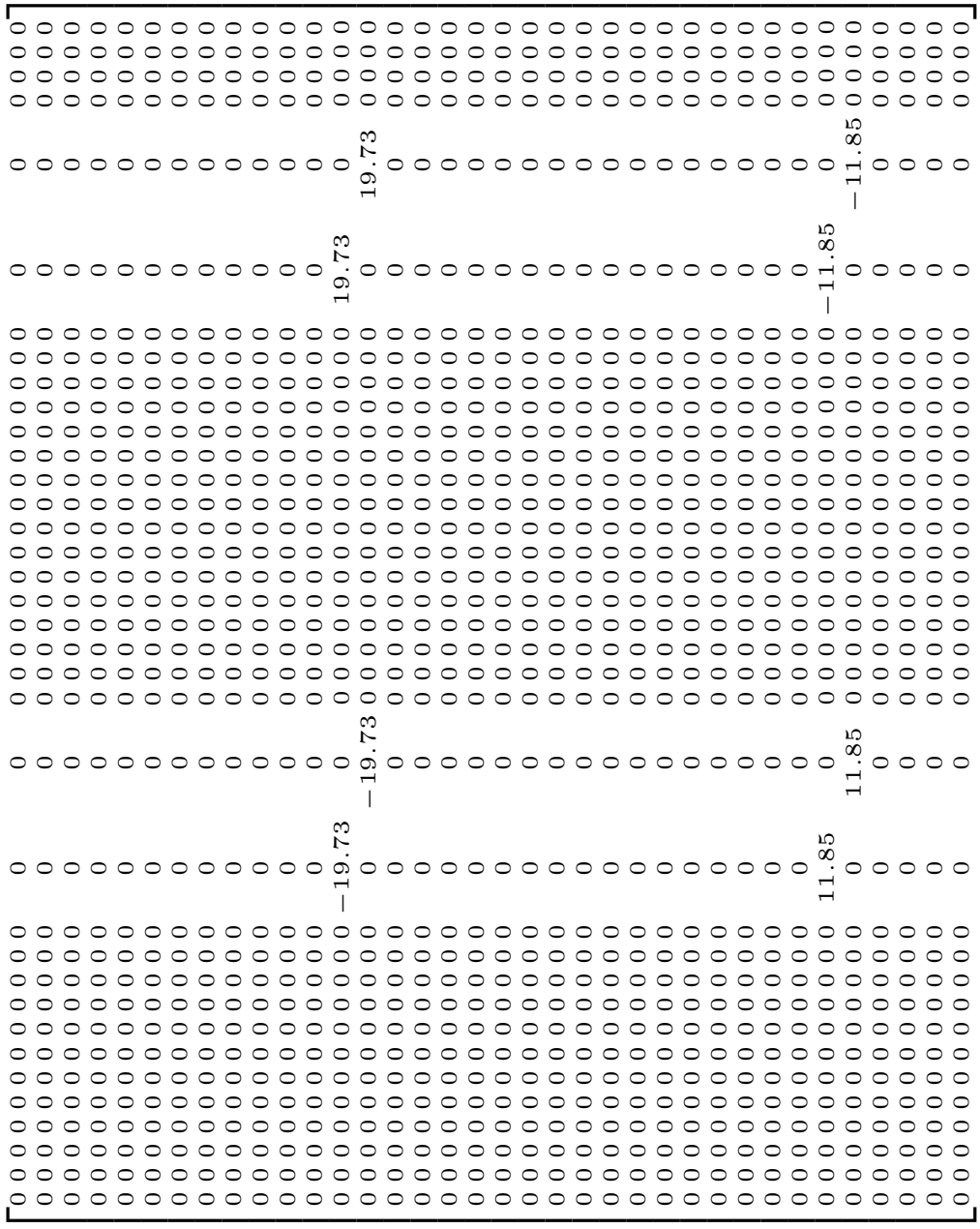
Amalgemations-Matrix $X4, 3$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	−15.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15.10	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	−15.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15.10	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	27.57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−27.57	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	27.57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−27.57	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

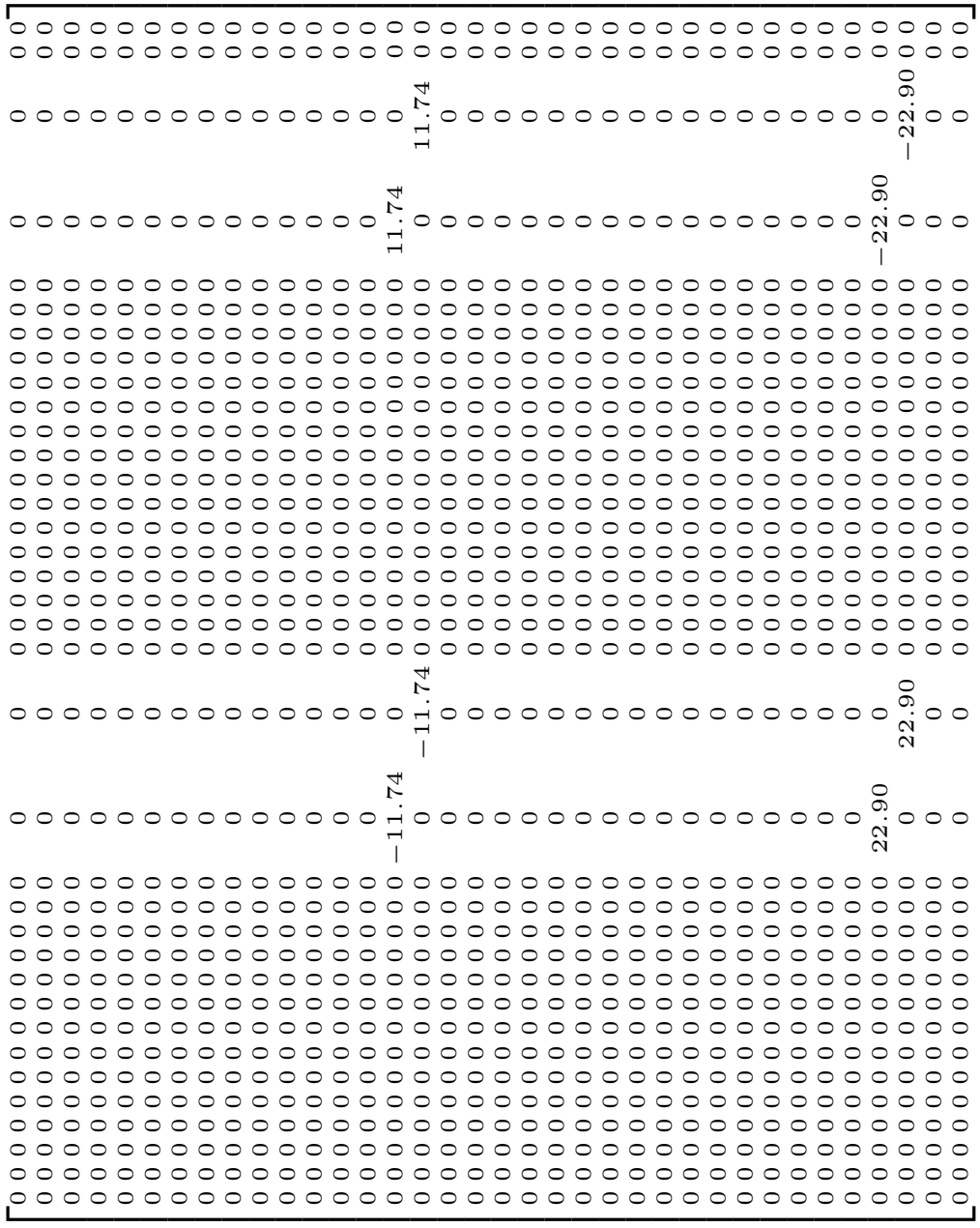
Amalgemations-Matrix $X4, 4$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	−24.29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24.29	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	−24.29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24.29	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10.98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−10.98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	10.98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−10.98	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

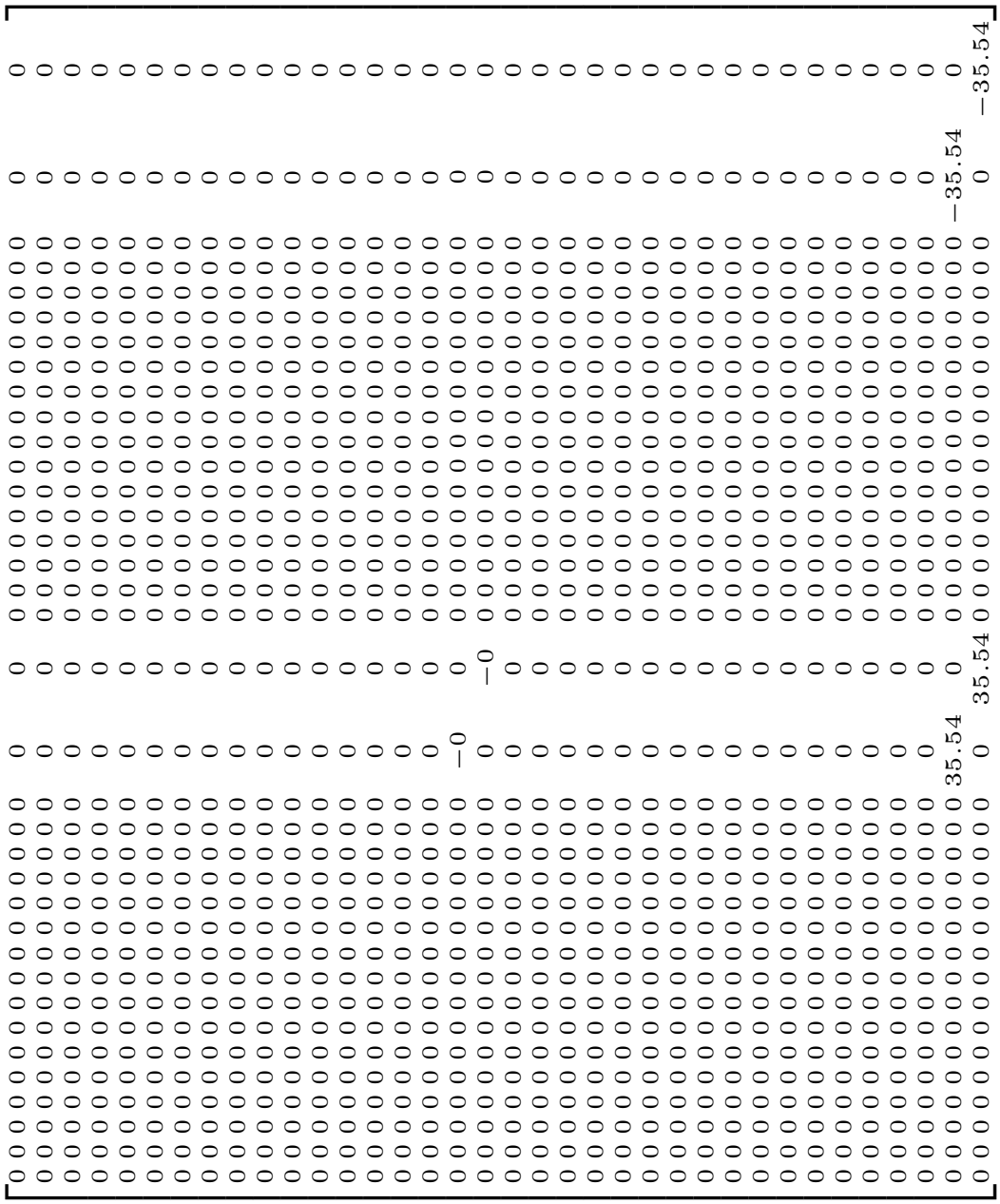
Amalgemations-Matrix $X4,5$



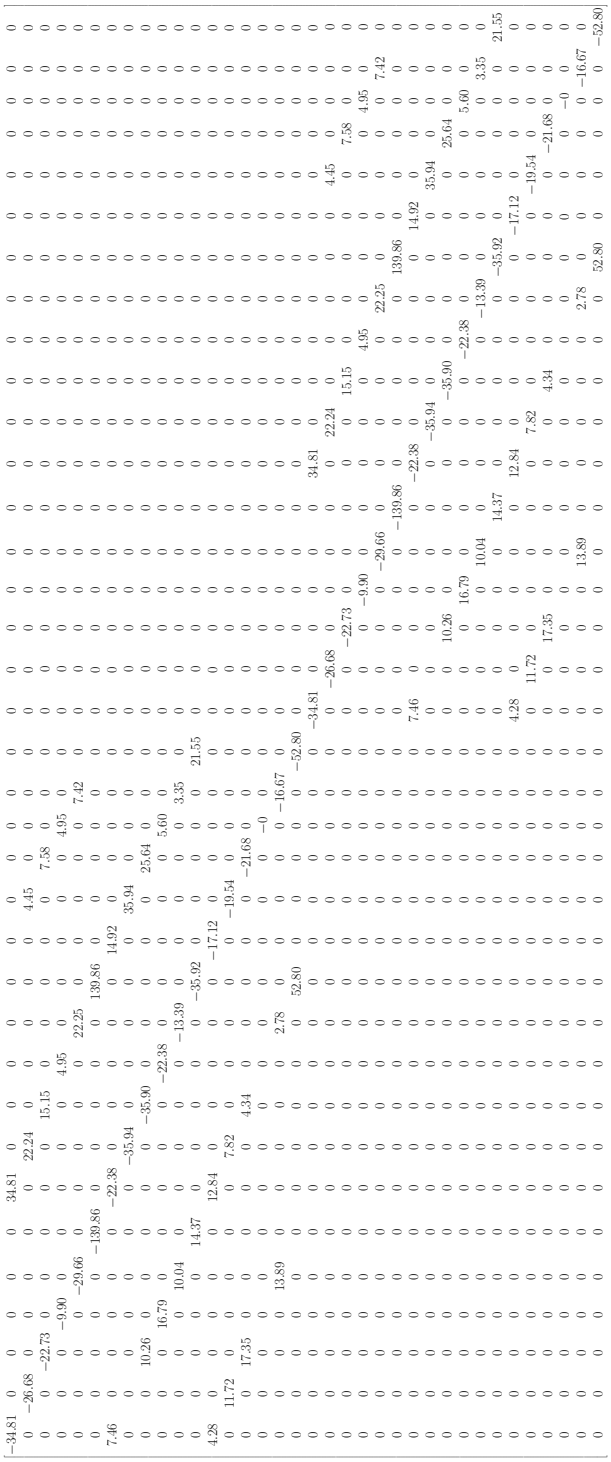
Amalgemations-Matrix $X4,6$



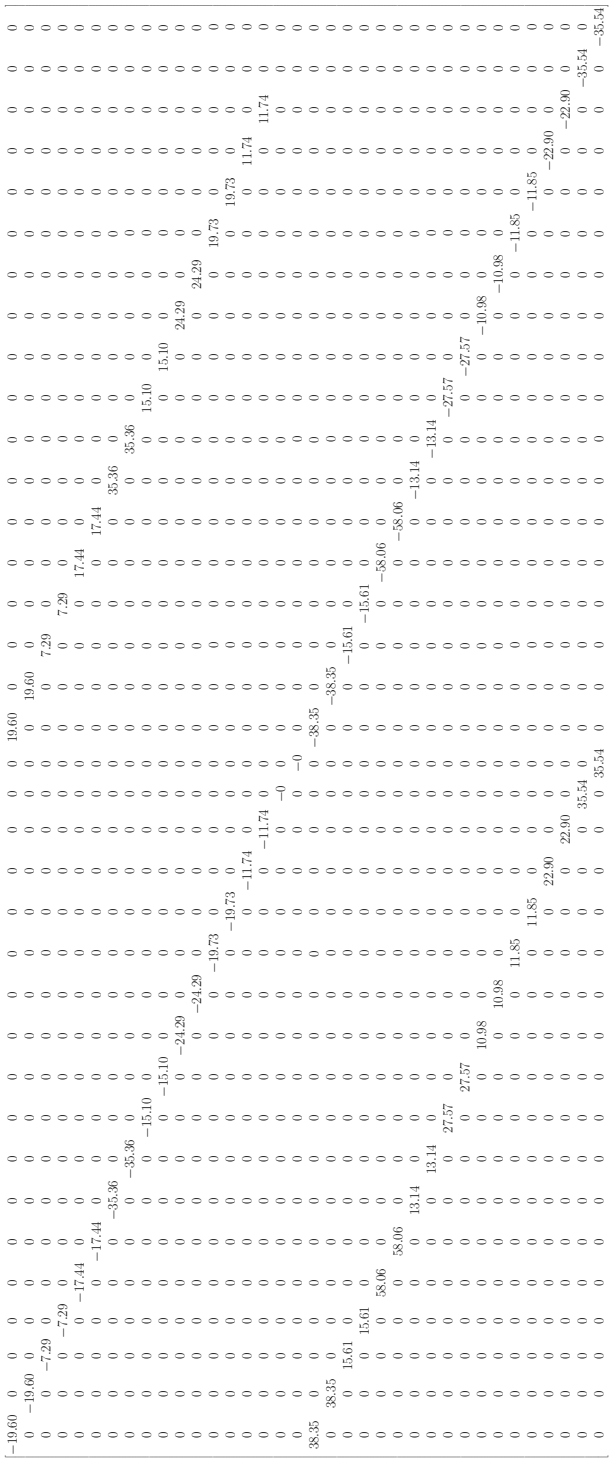
Amalgemations-Matrix $X4,7$



Amalgemations-Matrix $X4,8$



Amalgemations-Matrix (Fusioniert) $X3$



Amalgemations-Matrix (Fusioniert) $X4$