Technische Universität München

Fakultät für Informatik

Master's Thesis in Informatik

Modellierung und Anwendung von Redundanz

Clemens Paul

Technische Universität München

Fakultät für Informatik

Master's Thesis in Informatik

Modellierung und Anwendung von Redundanz

Modeling and Application of Redundancy

Autor	Clemens Paul
Aufgabensteller	Prof. Dr.-Ing. Georg Carle
Betreuer	Nadine Herold, M. Sc., Dipl.-Inf. Stephan-A. Posselt
Datum	15. Juli 2015

Informatik VIII
Lehrstuhl für Netzarchitekturen und Netzdienste

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Garching b. München, 15. Juli 2015

Unterschrift

Zusammenfassung

Redundanzen sind für Netzwerke, die eine hohe Zuverlässigkeit aufweisen sollen, essenziell. Aus diesem Grund wurde ein Algorithmus entwickelt, der bei der Bestimmung der Zuverlässigkeit eines Netzwerks mit k-Terminals die Redundanzen der k-Terminals berücksichtigt.

Basierend auf der Berechnung der Zuverlässigkeit wird ein Vorschlagsystem, bestehend aus vier Algorithmen, beschrieben, das dem Nutzer ermöglicht, kritische Entitäten des Netzwerks zu bestimmen, die Zuverlässigkeit des Netzwerks zu erhöhen oder zu verringern und das Vorschläge für die Positionierung einer Redundanz für ein k-Terminal unterbreitet.

Notwendige Grundlagen für das Verständnis der Arbeit werden erläutert. Dies betrifft die Definitionen der Begriffe Zuverlässigkeit und Redundanz sowie die Beschreibung der verwendeten Datenstruktur Binary Decision Diagram (BDD). Des Weiteren wird eine Analyse der Anforderungen für die entwickelten Algorithmen sowie die Betrachtung verwandter Arbeiten durchgeführt.

Beispiele für die entworfenen Algorithmen werden veranschaulicht und deren Implementierung beschrieben. Zusätzlich wird die Benutzbarkeit des Programms, bestehend aus der Berechnung der Zuverlässigkeit sowie des Vorschlagsystems, erklärt.

Des Weiteren wird eine qualitative (bezogen auf die Anforderungen an das System) und quantitative Evaluierung sowie eine Komplexitätsanalyse der entwickelten Algorithmen durchgeführt.

Abschließend wird ein Ausblick für mögliche Verbesserungen und Erweiterungen in zukünftigen Arbeiten gegeben.

Abstract

Redundancy is crucial in networks that are supposed to exhibit high reliability. Therefore, an algorithm was developed that, during determination of reliability of networks including k-terminals, considers the redundancy of the k-terminals.

A system of proposal that comprises four algorithms and that is based on the calculation of reliability is described. The system allows the user to determine critical entities of the network, to increase as well as decrease reliability of the network, and to provide propositions for positioning a redundancy for a k-terminal.

Requisite basic principles to understand the Thesis are explained. This applies to the definition of the terms reliability and redundancy as well as the specification of the used data structure, that is, Binary Decision Diagram (BDD). Furthermore, an analysis of the requirements for the developed algorithms as well as a review of related publications is performed.

Examples for the developed algorithms are illustrated and their implementation is described. Additionally, the usability of the program that comprises the calculation of the k-terminal-reliability as well as the system of proposal is explained.

Furthermore, a qualitative (in relation to the system requirements) and a quantitative evaluation as well as an analysis of the complexity for the developed algorithms is performed.

In the concluding chapter, an outlook on possible improvements and enhancements in future research is provided.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis verwendeter Algorithmen

Kapitel 1

Einleitung

In der heutigen, stark vernetzen Welt spielt es eine entscheidende Rolle eine Vorhersage bezüglich der Funktionsfähigkeit eines Systems, wie z. B. eines Netzwerks, treffen zu können. Eine dafür geeignete Metrik ist die Zuverlässigkeit, die von IEEE in ihrem Standard [4] folgendermaßen definiert wird:

> "The ability of a system or component to perform its required functions under stated conditions for a specified period of time."

Zuerst wurden bestimmte Anforderungen und Evaluierungen der Zuverlässigkeit nur für Systeme entworfen, die bei einem Fehler zu massivem Schaden oder zum Verlust eines Menschenlebens führten. Beispiele hierfür sind Space Shuttles (z. B. Challenger Crash 1986), Flugzeugsysteme, Systeme zur Steuerung von Kernreaktoren, Verteidigungssysteme eines Landes, kritische Import-/Export-Systeme, Transportsysteme, Automobilindustrie, Systeme für militärische Zwecke, Energiesysteme und Systeme für den medizinischen Bereich.

Heutzutage werden auch für Systeme, die im privaten Bereich vertrieben werden, eine entsprechende Analyse und Evaluierung der Zuverlässigkeit durchgeführt. Da diese Systeme eine hohe Zuverlässigkeit aufweisen sollen, um bei Veränderungen der Netztopologie (z. B. Ausfall von Komponenten oder Verbindungen) oder des Verkehrsaufkommens weiter die gleiche Qualität und vollständige Funktionsfähigkeit bereitzustellen, ist der Einsatz von Redundanzen von großer Bedeutung.

Aus diesem Grund befasst sich diese Thesis mit der Berechnung der Zuverlässigkeit eines Netzwerks unter der Berücksichtigung redundanter Komponenten. Das Netzwerk wird dabei als Graph-Modell mit statischen Eigenschaften abstrahiert, wodurch sämtliche Systeme, die auf einer solchen Struktur basieren, herangezogen und ihre Zuverlässigkeit bestimmt werden kann.

Das zugrunde liegende Problem für die Bestimmung der Zuverlässigkeit wird als k-Terminal-Problem bezeichnet. Ein Terminal beschreibt eine Komponente des Netzwerks,

bei deren Ausfall das Netzwerk nicht länger funktionsfähig wäre. Die k-Terminal-Zuverlässigkeit wird durch die Wahrscheinlichkeit, dass mindestens ein funktionsfähiger Pfad zwischen den k-Terminals (Komponenten) existiert, definiert. Dabei können sowohl Komponenten als auch die Verbindungen zwischen den Komponenten ausfallen. Da dieses Problem NP-schwer [5, 6] ist, werden Algorithmen vorgestellt, die dies effizient lösen.

Basierend auf der Berechnung der Zuverlässigkeit von k-Terminals wird ein Vorschlagsystem beschrieben, das vier Algorithmen vorstellt. Mit einem dieser Algorithmen können kritische Netzentitäten bestimmt werden. Dadurch kann die Relevanz einzelner Entitäten für die Zuverlässigkeit des Netzes berechnet werden. Auf Basis dieser Informationen werden zwei Algorithmen definiert, um die Zuverlässigkeit eines Netzwerks sowohl erhöhen als auch verringern zu können. Ein weiterer Anwendungsfall des Vorschlagsystems besteht darin, dem Nutzer Vorschläge für die Positionierung einer Redundanz eines k-Terminals zu unterbreiten, um dadurch die Zuverlässigkeit zu maximieren.

In diesem Kapitel folgt nun die Beschreibung der Forschungsfragen und der Aufbau der Thesis.

1.1 Forschungsfragen

Wie können redundante Netzentitäten, d. h. Komponenten und Pfade, modelliert werden?
Diese Frage betrifft das Entwickeln eines Modells, welches redundante Netzentitäten darstellen und von entsprechenden Algorithmen verarbeitet werden kann. Dabei sollte beachtet werden, welche Arten von Redundanzen es gibt und wie spezifisch oder allgemein das Modell sein muss.

Wie kann ein solches Modell umgesetzt werden?
Dies bezieht sich in erster Linie darauf, wie ein solches Modell effizient umgesetzt werden kann, da eine ineffiziente Lösung keine Herausforderung darstellt. Dabei muss eine etwaige Lösung enstprechend mit der theoretischen Komplexität in Verbindung gebracht werden, um beantworten zu können, ob die Lösung bzw. das Modell, das umgesetzt wurde, effizient ist.

Wie können diese Ergebnisse für die Verbesserung der Zuverlässigkeit eingesetzt werden?
Hierfür sollte überlegt werden, wie durch das Einfügen redundanter Netzentitäten die Zuverlässigkeit verbessert werden kann. Dabei ist eine Betrachtung der Informationen, aufgrund derer eine Redundanz hinzugefügt werden soll, wichtig.

1.2 Aufbau der Thesis

Das erste Kapitel enthält die Einleitung zur Thematik dieser Thesis. Dabei werden unter anderem die aufgestellten Forschungsfragen und der Aufbau der Thesis beschrieben.

Im zweiten Kapitel werden notwendige Grundlagen für das Verständnis der vorliegenden Arbeit erläutert. Dieses Kapitel ist in zwei Abschnitte unterteilt. In 2.1 wird sowohl eine Definition für die Begriffe Zuverlässigkeit und Redundanz gegeben als auch beschrieben, welche der verschiedenen Formen der Zuverlässigkeit und der Redundanz in dieser Thesis eingesetzt werden. Weiterhin werden die Begriffe Verfügbarkeit und Resilienz eines Netzwerks erklärt, um sie von der Zuverlässigkeit eines Netzwerks abzugrenzen. Bezüglich Redundanz werden verschiedene Klassifizierungen erläutert. Der zweite Abschnitt in Kapitel 2 fokussiert sich auf die Datenstruktur Binary Decision Diagram (BDD), die von den in dieser Arbeit entwickelten Algorithmen verwendet wird. Dabei wird eine allgemeine Beschreibung der Datenstruktur gegeben und mögliche Operationen eines BDDs werden angeführt.

Anschließend wird in Kapitel 3 eine Analyse der System-Anforderungen durchgeführt. Dabei werden 11 Anforderungen an das System, die identifiziert wurden, beschrieben. Diese Anforderungen sind für den Vergleich mit verwandten Arbeiten und für die qualitative Evaluierung entscheidend.

Kapitel 4 befasst sich hinsichtlich der Thematik dieser Arbeit mit verwandten Arbeiten. Das Kapitel ist in vier Abschnitte unterteilt. Die ersten drei Abschnitte befassen sich jeweils mit verwandten Arbeiten zur Berechnung der Zuverlässigkeit von k-Terminals, zum Identifizieren kritischer Entitäten und zur Verbesserung der Zuverlässigkeit. Im vierten Abschnitt wird in einer Tabelle eine Gegenüberstellung der Arbeiten, bezogen auf die System-Anforderungen durchgeführt.

Kapitel 5 ("Entwurf") ist das Zentrale-Kapitel dieser Thesis. Zu Beginn des Kapitels wird in Abschnitt 5.1 der Aufbau der algorithmischen Struktur erläutert. Dieser Abschnitt ist in Eingabe, Verarbeitung und Ausgabe unterteilt, um einen Überblick über das entwickelte System zu geben. Der nächste Abschnitt des Kapitels beschreibt die Datenstrukturen, die von den entwickelten Algorithmen eingesetzt werden. Dies sind das BDD und ein aufgestelltes Graph-Modell, bestehend aus Knoten und Kanten.

Des Weiteren werden in den Abschnitten 5.3 und 5.4 die entwickelten Algorithmen beschrieben. Dabei wird in Abschnitt 5.3 erklärt, wie unter Berücksichtigung von Redundanzen die Zuverlässigkeit von k-Terminals eines Netzwerks berechnet werden kann. Basierend auf dieser Berechnung wird in Abschnitt 5.4 das Vorschlagsystem, bestehend aus vier Algorithmen, vorgestellt.

Der letzte Abschnitt dieses Kapitels umfasst Annahmen und Einschränkungen, die getroffen wurden, sowie mögliche Anwendungsfälle für die entwickelten Algorithmen.

In Kapitel 6 wird die Implementierung des Programms beschrieben. Dabei werden die eingesetzte Entwicklungsumgebung und die Datenstruktur sowie verwendete Bibliotheken angeführt. Weiterhin werden Implementierungsdetails zu den entwickelten Algorithmen zur Berechnung der Zuverlässigkeit und des Vorschlagsystems beschrieben. Im letzten Abschnitt wird die Benutzbarkeit des Programms erläutert. Dies betrifft sowohl die Ausführung des Programms als auch die Spezifikation des Dateiformats, die der einzugebende Netzplan aufweisen muss. Zusätzlich wird das konsolenbasierte User Interface beschrieben.

Kapitel 7 beinhaltet die Durchführung einer qualitativen und quantitativen Evaluierung sowie einer Analyse der theoretischen Komplexität der entwickelten Algorithmen. Die qualitative Evaluierung bezieht sich auf die System-Anforderungen. Darin wird beschrieben, ob und wie die Anforderungen durch den Einsatz bestimmter Algorithmen erfüllt werden konnten. Abschnitt 7.3 beschreibt die quantitative Evaluierung, in der Benchmarks für die entwickelten Algorithmen aufgezeigt werden, um den Praxisbezug besser zu verdeutlichen.

Im letzten Kapitel werden die wichtigsten Punkte der vorliegenden Arbeit zusammengefasst. Zusätzlich wird ein Ausblick für zukünftige Arbeiten gegeben. Dieser Ausblick richtet sich auf mögliche Verbesserungen hinsichtlich der Laufzeit des Algorithmus zur Berechnung der Zuverlässigkeit und mögliche Erweiterungen des Graph-Modells.

Kapitel 2

Grundlagen

Dieses Kapitel soll die Grundlage für das Verständnis der Thesis schaffen. Dabei werden im ersten Abschnitt die Begriffe Zuverlässigkeit und Redundanz beschrieben. Der zweite Abschnitt fokussiert sich auf die Datenstruktur Binary Decision Diagram (BDD), die von den entwickelten Algorithmen eingesetzt wird.

2.1 Zuverlässigkeit und Redundanz

Dieser Abschnitt wird in zwei weitere Punkte unterteilt und liefert dabei eine Beschreibung für die Begriffe Zuverlässigkeit und Redundanz. Zusätzlich werden in dem Abschnitt, der die Zuverlässigkeit definiert, noch die Begriffe Verfügbarkeit und Resilienz kurz erläutert, um sie voneinander abzugrenzen.

2.1.1 Zuverlässigkeit, Verfügbarkeit und Resilienz

Dieser Punkt fokussiert sich in erster Linie auf die Beschreibung der Zuverlässigkeit eines Netzwerks. Zusätzlich werden die Begriffe Verfügbarkeit und Resilienz eines Netzwerks erklärt.

Zuverlässigkeit
Ma [7] stellt fest, dass die Funktion zur Berechnung der Zuverlässigkeit $R(t)$ eine gute Definition für die Zuverlässigkeit selbst liefert:

> "Reliability function [$R(t) = P(T > t)$] (where T is the lifetime or failure time, and P is the probability) serves as the definition of reliability very successfully in the sense that (i) it acts as an excellent pedagogical model; (ii) its fundamental concept, although often extended extensively, still holds in complex real-world reliability analysis."

Sowohl diese Definition als auch die in der Einleitung erwähnte Definition von IEEE können für das Verständnis des Begriffs Zuverlässigkeit im Sinne dieser Arbeit verwendet werden In dieser Thesis stellt ein Netzwerk, das aus Entitäten – Komponenten und Verbindungen – besteht und als Graph-Modell dargestellt werden kann, das System dar, dessen Zuverlässigkeit bestimmt werden soll.

Dabei können alle Entitäten mit einer gegebenen Wahrscheinlichkeit ausfallen. Es gibt für eine Entität nur zwei Zustände, entweder sie ist funktionsfähig oder nicht. Zusätzlich wird angenommen, dass die Ausfälle von Entitäten unabhängig voneinander sind. Auf die Frage der Reparatur von Komponenten wird nicht eingegangen.

Da im Allgemeinen Kanten eine hohe Ausfallsicherheit aufweisen und es viele redundante Pfade zwischen den Knoten gibt, haben Netzwerke eine hohe Zuverlässigkeit [8, ch. 6.2]. Durch diese hohe Anzahl an parallelen Pfaden ergibt sich eine hohe Komplexität für die effiziente Berechnung der Zuverlässigkeit. Dies ist eines der Kernprobleme der Analyse, des Designs und der Darstellung von Netzwerken [8, ch. 6.2].

Es gibt die folgenden zu unterscheidenden Arten der Zuverlässigkeit eines Netzwerks [8, ch. 6.3]:

- 2-Terminal: Betrachtet die Kommunikation zwischen einem Quellknoten s und einem Zielknoten t. Die Zuverlässigkeit zwischen zwei Knoten (2-Terminal-Problem genannt) wird durch die Wahrscheinlichkeit angegeben, dass mindestens ein funktionierender Pfad zwischen s und t existiert.

$$R_{st} = P(\text{Knoten } s \text{ und } t \text{ sind verbunden})$$

- All-Terminal: Betrachtet die Kommunikation zwischen einem Quellknoten s und den $(n-1)$ anderen Knoten, wobei n die Anzahl der Knoten des Netzwerks bezeichnet. Sobald ein Pfad von s zu den $(n-1)$ anderen Knoten existiert, gibt es auch einen Pfad zwischen den $(n-1)$ Knoten untereinander. Die Zuverlässigkeit zwischen den n Knoten wird durch die Wahrscheinlichkeit angegeben, dass mindestens ein funktionierender Pfad zwischen allen n Knoten exisitert.

$$R_{all} = P(\text{alle } n \text{ Knoten sind verbunden})$$

- k-Terminal: Betrachtet die Kommunikation zwischen k-Knoten, wobei $2 \leq k \leq n$ gilt. Die Zuverlässigkeit zwischen k-Knoten (k-Terminal-Problem genannt) wird durch die Wahrscheinlichkeit angegeben, dass mindestens ein funktionierender Pfad zwischen den k-Knoten existiert.

$$R_k = P(\text{alle } k\text{-Knoten sind verbunden})$$

Diese Thesis fokussiert sich auf die Berechnung der Zuverlässigkeit von k-Terminals unter Berücksichtigung der Redundanzen dieser Terminals. Es gibt verschiedene Methoden, wie die Zuverlässigkeit eines Netzwerks bestimmt werden kann. Hierfür bietet der Abschnitt 4.1 des Kapitels "Verwandte Arbeiten" einen Überblick und verweist dabei auf die entsprechenden wissenschaftlichen Arbeiten, die sich mit dieser Thematik im Detail beschäftigen.

Verfügbarkeit
Obwohl keine Verfügbarkeitsanalyse in dieser Arbeit durchgeführt wird, soll diese Metrik kurz erläutert werden, um sie von der Zuverlässigkeit abzugrenzen.

Die Verfügbarkeit wird als Metrik eingesetzt, um ein Reparatursystem zu beschreiben. Sie wird als die Wahrscheinlichkeit definiert, dass zu jedem Zeitpunkt t das System funktionsfähig und somit verfügbar ist. Als Notation wird $A(t)$ für die Verfügbarkeit eingesetzt. Verfügbarkeit und Zuverlässigkeit können durch die Vereinigung folgender Ereignisse miteinander in Relation gesetzt werden [8, ch. 6.3]:

$$\begin{aligned} A(t) = P(&\text{Kein Fehler im Intervall 0 bis } t \\ &+ 1 \text{ Fehler und 1 Reparatur im Intervall 0 bis } t \\ &+ 2 \text{ Fehler und 2 Reparaturen im Intervall 0 bis } t + \ldots) \end{aligned} \tag{2.1}$$

Da alle Ereignisse der Gleichung 2.1 voneinander unabhängig sind, kann sie als die Summe der Wahrscheinlichkeiten folgendermaßen erweitert werden:

$$\begin{aligned} A(t) = P(&\text{Kein Fehler im Intervall 0 bis } t) \\ &+ P(1 \text{ Fehler und 1 Reparatur im Intervall 0 bis } t) \\ &+ P(2 \text{ Fehler und 2 Reparaturen im Intervall 0 bis } t) + \ldots \end{aligned} \tag{2.2}$$

Daraus kann Folgendes abgeleitet werden:

- Der erste Term der Gleichung 2.2 stellt die Zuverlässigkeit $R(t)$ dar.
- $A(t) = R(t) = 1$, falls $t = 0$ gilt.
- Für $t > 0$ gilt $A(t) > R(t)$.
- $R(t) \to 0$, falls $t \to \infty$

Resilienz
Eine weitere wichtige Metrik für Netzwerke ist die Resilienz bzw. Robustheit. Im Jahr 1972 haben Van Sylke und Frank die Kenngröße Resilienz vorgestellt, um die geschätzte Anzahl Knoten-Paare anzugeben, die miteiannder kommunizieren können [8, ch. 6.5]. D. h. Knoten-Paare zwischen denen mindestens ein Pfad existiert. Sei s und t ein Knoten-Paar: Die Anzahl der Knoten-Paare in einem Netzwerk mit N Knoten entspricht der

Anzahl der Kombinationen von N über 2.

$$\text{Anzahl der Knoten-Paare } = \binom{N}{2} = \frac{N!}{2!(N-2)!} = \frac{N(N-1)}{2} \tag{2.3}$$

Die Notation $\{s,t\} \subseteq N$ drückt aus, dass die Menge des Knoten-Paars s und t im N Knoten-Netzwerk enthalten ist. Die Resilienz des Graphen G wird nun durch den folgenden Ausdruck beschrieben [8, ch. 6.5]:

$$\text{res}(G) = \sum_{\{s,t\} \subseteq N} R_{st} \tag{2.4}$$

Aus der Gleichung 2.4 geht hervor, dass für sehr zuverlässige Netzwerke die Resilienz annähernd dem Ausdruck $N(N-1)/2$ entspricht. Daher kann es sinnvoll sein, die Resilienz zu normalisieren, indem sie durch den Ausdruck $N(N-1)/2$ dividiert wird. Dadurch können unterschiedlich große Netzwerke besser miteinander verglichen werden. Die "normalisierte" Resilienz entspricht der durchschnittlichen Zuverlässigkeit aller 2-Terminal-Paare des Netzwerks [8, ch. 6.5].

2.1.2 Redundanz

In diesem Abschnitt werden verschiedene Klassifizierungen sowie Einsatzmöglichkeiten von Redundanz erläutert und der in der Arbeit verwendete Redundanz-Typ beschrieben.

Laut Mili et al. [9] ist Redundanz eine System-Eigenschaft, die sich im Allgemeinen auf die Vervielfältigung von Zustandsinformationen oder System-Funktionen bezieht. Obwohl Redundanz häufig in Verbindung mit Fehlertoleranz betrachtet wird, ist es im Wesentlichen eine System-Eigenschaft, die für sich selbst analysiert werden kann. Aus diesem Grund haben Mili et al. ein Modell aufgestellt, um Redundanz entsprechend analysieren zu können.

Ebrahimipour et al. [10] beschreiben Redundanz als ein sehr wichtiges Konzept, um die Zuverlässigkeit eines Systems zu verbessern. Laut [10] ist es allgemein anerkannt, dass fehlertolerante und sicherheitskritische Systeme ihre angestrebte Zuverlässigkeit nicht ohne das Einführen redundanter Komponenten erfüllen können.

In der Literatur werden viele individuelle Formen der System-Redundanz beschrieben und unterschiedlich klassifiziert. Shooman [8] unterscheidet zwei Ebenen der Redundanz. Dazu gehören "Modular Redundancy" – die Modul-, Komponenten- und Subsystem-Ebene – und die "Microcode-Level Redundancy". Zusätzlich unterscheidet er zwischen zwei Architekturen bei Redundanz. Diese sind: "Parallel Redundancy" – entspricht der "Active Redundancy" – und "Standby Redundancy".

Mili et al. [9] definieren folgende Formen der Redundanz:

- *State Redundancy*
  Diese tritt auf, wenn die Darstellung des Systemzustands ein größeres Intervall an Werten erlaubt als benötigt wird, um alle möglichen Zustände zu repräsentieren. Diese Form der Redundanz kommt z. B. bei Parity-Bit-Schemeta, Fehlerkorrekturcodes sowie Schemata für modulare Redundanz zum Einsatz.

- *Functional Redundancy*
  Diese Form der Redundanz tritt auf, wenn z. B. drei verschiedene Algorithmen verwendet werden, um die gleiche Funktionalität zu berechnen und die Auswahl aufgrund des Ergebnisses getroffen wird. Dabei unterscheiden Mili et al. nicht zwischen paralleler und serieller Ausführung der Berechnung.

- *Temporal Redundancy*
  Angenommen, ein Zustand wird durch zwei Variablen definiert: Die Höhe Z und die vertikale Geschwindigkeit V_Z eines Flugzeugs. Dabei existiert keine Redundanz zwischen den Werten Z und V_Z zu einem bestimmten Zeitpunkt t, d. h. $Z(t)$ und $V_Z(t)$ können unterschiedliche Werte für ein gegebenes t aufweisen. Allerdings besteht eine Redundanz zwischen den Variablen innerhalb eines kleinen Zeitintervall mit z. B. $Z' = Z + V_Z \times dt$.

- *Control Redundancy*
  Diese tritt in Kontroll-Anwendungen auf, wenn der gleiche Effekt des Systems durch mehrere unterschiedliche Kontroll-Einstellungen erzielt wird. Hierfür geben Mili et al. [9] ein Beispiel bezüglich eines Flugzeugs an, das unter bestimmten Bedingungen flugfähig bleibt, obwohl einige Bedienelemente ausfallen. Dies wird aufgrund der Redundanz zwischen Bedienelementen, die für die Funktionsfähigkeit des Systems verantwortlich sind, gewährleistet. Obwohl Elemente wie Gashebel, Landeklappe, Höhen-, Quer- und Seitenruder unterschiedliche Funktionen besitzen, können manche vielleicht den Ausfall eines anderen Elements ausgleichen.

Ebrahimipour et al. [10] unterteilen Redundanz in zwei wesentliche Typen:

- *Active Redundancy* – alle intakten redundanten Komponenten sind ständig in Betrieb, obwohl das System nur einige wenige zu einem bestimmten Zeitpunkt benötigt, um funktionsfähig zu sein. Bei diesem Redundanz-Typ ist es essenziell, dass eine Komponente bei Ausfall durch eine funktionsfähige ersetzt wird.

- *Standby Redundancy* – die redundanten Komponenten sind nur im Betrieb, falls durch das Versagen einiger Komponenten nur noch wenige übrig sind. Dieser Typ an Redundanz wird oft verwendet, um die Zuverlässigkeit sowie Verfügbarkeit eines Systems zu verbessern. "Standby Redundancy" kann in drei weitere Typen unterteilt werden:

  – Cold – impliziert, dass inaktive Komponenten nicht ausfallen können, so-

lange sie im Standby-Modus sind.

- Hot – impliziert, dass inaktive Komponenten die gleiche Ausfallwahrscheinlichkeit wie aktive Komponenten aufweisen
- Warm – liegt zwischen Cold- und Hot-Standby. D. h. inaktive Komponenten haben eine Ausfallwahrscheinlichkeit, die zwischen der für Cold- und Hot-Standby liegt.

Dieser Überblick zeigt, dass Redundanz sowohl unterschiedlich klassifiziert als auch verschieden eingesetzt wird. Der in dieser Arbeit eingesetzte Redundanz-Typ entspricht der "Active Redundancy", da bei der Zuverlässigkeitsanalyse der entwickelten Algorithmen angenommen wird, dass jede funktionsfähige Redundanz immer erreichbar ist. Allerdings wird durch das aufgestellte Modell keine direkte Verbindung zwischen einer Komponente und ihrer Redundanz abgebildet. Somit erfolgt keine Betrachtung eines Informationsaustauschs der Komponenten mit ihren Redundanzen.

2.2 Binary Decision Diagram

Die in dieser Arbeit entwickelten Algorithmen (siehe Kapitel 5) bauen auf der Datenstruktur Binary Decision Diagram (BDD) auf, die in diesem Abschnitt beschrieben wird. Der Abschnitt ist in zwei Punkte gegliedert, wobei der erste Punkt eine allgemeine Beschreibung der Datenstruktur und der zweite Punkt mögliche Operationen eines BDDs beinhaltet.

2.2.1 Allgemeine Beschreibung

Ein BDD ist eine Datenstruktur, die eine n-dimensionale boolesche Funktion

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

abbildet. Ein BDD speichert für alle booleschen Variabeln das dazugehörige Ergebnis in einer komprimierten Form und kann jegliche boolesche Funktion repräsentierten.

Die Datenstruktur ist als gerichteter azyklischer Graph (DAG) aufgebaut und besitzt eine Menge N an Entscheidungs-Knoten und zwei Blätter bzw. Terminal-Knoten, die mit 0 (False) und 1 (True) gekennzeichnet werden. Es gibt einen internen Knoten, der keine eingehenden Kanten besitzt und als "root" bezeichnet wird. Jeder dieser Entscheidungs-Knoten $v \in N$ wird als eine boolesche Variable $b := var(v)$ beschriftet und ist mit zwei Kinder-Knoten verbunden, die als "low" und "high" bezeichnet werden. Bei der Darstellung der Kanten, die auf die Kinder-Knoten "low"/"high" zeigen, wird häufig eine strichlierte bzw. durchgezogene Linie verwendet. Diese Kinder-Knoten werden erreicht, indem der Variablen b 0/1 zugewiesen wird [1].

Die Funktion f_{b_i} wird als Einschränkung von f bezeichnet, falls b_i für $1 \leq i \leq n$ durch eine Konstante $c \in \{0,1\}$ ersetzt wird:

$$f_{b_i=c}(b_1,...,b_n) = f(b_1,...,b_{i-1},c,b_{i+1},...,b_n). \tag{2.5}$$

Jeder BDD-Knoten v ist für sich eine boolesche Funktion f, die entsprechend ihrer booleschen Variable b zerlegt werden kann. Dies wird als "Shannon-Decomposition" von f mit Berücksichtigung auf b bezeichnet:

$$f_b = b \cdot f_{b=1} + \overline{b} \cdot f_{b=0}, \tag{2.6}$$

wobei $high(v) := f_{b=1}$ und $low(v) := f_{b=0}$ die Kinder-Knoten "high" und "low" des BDD-Knotens v darstellen [1].

Die Größe eines BDDs entspricht der Anzahl an Entscheidungs-Knoten und ist stark von der Variablen-Ordnung abhängig. Ein BDD mit einer entsprechenden fixen Variablen-Ordnung, d. h. dass alle Pfade des Graphen eine gegebene lineare Ordnung der Variablen $x_1 < x_2 < ... < x_n$ folgen, wird als OBDD (Ordered Binary Decision Diagram) bezeichnet [11].

Des Weiteren kann ein BDD auch als ROBDD (Reduced Ordered Binary Decision Diagram) bezeichnet werden, falls dieser als "reduziert" gilt und somit die folgenden Eigenschaften aufweist [1]:

- **Einzigartigkeit**: Es dürfen keine zwei unterschiedlichen Knoten u und v die gleiche Bezeichnung sowie dieselben "low" und "high" Kinder-Knoten haben. D. h. $var(u) = var(v), low(u) = low(v), high(u) = high(v)$ impliziert, dass $u = v$ gilt (siehe den linken BDD-Graph der Abbildung 2.1).
- **Keine redundanten Knoten**: Kein interner Knoten u darf identische "low" und "high" Kinder-Knoten aufweisen, d. h. $low(u) \neq high(u)$ (siehe den rechten BDD-Graph der Abbildung 2.1).

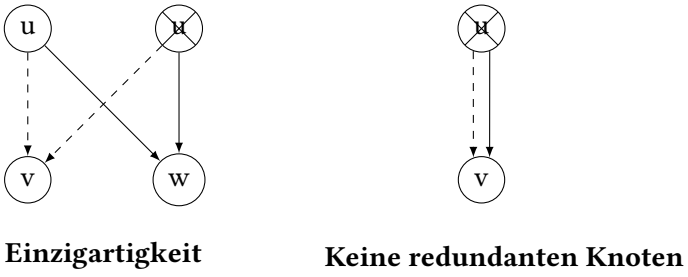


Abbildung 2.1: Eigenschaften eines ROBDDs [1]

Laut [12] ist die Darstellung einer booleschen Funktion mit Hilfe eines ROBDDs kanonisch:

Lemma 2.1 (Canonicity Lemma). *Für jede boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ und eine gegebene Variablen-Ordnung existiert genau ein ROBDD für f, sodass jeder andere OBDD für f mehr Knoten beinhaltet.*

Daraus ergibt sich, dass zwei boolesche Funktionen identisch sind, falls ihre ROBDDs isomorph sind.

2.2.2 Operationen

Operationen sowie Manipulationen können direkt auf ein BDD angewandt werden, ohne vorher eine Dekomprimierung durchzuführen. Einige elementare Operationen sind logische "und"- und "oder"-Verknüpfungen sowie Negierung, Test der "Satisfiability" oder Überprüfung, ob zwei boolesche Funktionen äquivalent sind. Diese Operationen können alle durch BDDs effizient ausgeführt werden, wofür Bryant [12] Algorithmen definiert hat, die eine Zeitkomplexität proportional zur Größe des BDD-Graphens aufweisen.

Die in dieser Thesis sehr häufig verwendete "Apply"-Operation weist eine Zeitkomplexität von $O(|G_f| \cdot |G_g|)$ auf, wobei f und g zwei ROBDDs sind und G_f sowie G_g ihre entsprechenden BDD-Graphen. Die "Apply"-Operation wird in diesem Kapitel noch ausführlicher beschrieben.

Ein weiterer sehr wichtiger Algorithmus ist "Reduce", der einen OBDD-Graphen in einen ROBDD-Graphen mit der gleichen booleschen Funktion transformiert. Dieser Algorithmus basiert auf der Idee des Testens auf Isomorphismus bei Bäumen (vgl. Aho et al. [13]). Die Komplexität des "Reduce"-Algorithmus wurde von Bryant mit $O(|G_f| \cdot log|G_f|)$ angegeben und später von Wegener [14] auf linearen Aufwand reduziert.

Apply: Die "Apply"-Operation verwendet als Eingabe zwei Funktionen f und g, die als ROBDD repräsentiert werden, und erzeugt einen ROBDD für den Ausdruck $f \circ g$, wobei $\circ$ eine beliebige binäre boolesche Operation, wie z. B. "und", "oder" oder "xor" darstellt. Des Weiteren kann die "Apply"-Operation auf Implikation testen oder das Komplement einer Funktion erzeugen. Da der "Apply"-Algorithmus in Bezug auf seine Eingabe-Funktionen von der "Shannon-Expansion" abgeleitet wird, können diese Funktionen mit der Bedingung einer booleschen Variable b rekursiv zusammengefasst werden:

$$f \circ g = b \cdot (f_{b=1} \circ g_{b=1}) + \overline{b} \cdot (f_{b=0} \circ g_{b=0}). \tag{2.7}$$

Der Algorithmus nimmt die zwei ROBDD-Wurzelknoten r_f, r_g der beiden Graphen G_f, G_g und arbeitet sich nach unten. Dabei werden Knoten für den resultierenden Graphen $G_{f \circ g}$ an den Verzweigungsstellen der zwei ursprünglichen Graphen erstellt. Hierfür müssen mehrere Fälle unterschieden werden.

Falls r_f, r_g Blätter sind, besteht der resultierende Graph aus einem Blatt mit dem Wert $value(r_f) \circ value(r_g)$. Ohne Beschränkung der Allgemeinheit erhält das resultierende BDD eine Variablen-Ordnung von $x_1 < x_2 < ... < x_n$.

Für den nächsten Fall wird angenommen, dass einer der beiden Wurzelknoten kein Blatt ist. Sollte $var(r_f) = var(r_g) = x_i$ gelten, dann wird ein Knoten u mit $x_i = var(u)$ erstellt und der Algorithmus rekursiv auf $low(r_f)$ und $low(r_g)$ angewandt, um den Teilgraphen mit dem Wurzelknoten $low(u)$ zu erhalten. Zusätzlich wird der Algorithmus rekursiv auf $high(r_f)$ und $high(r_g)$ angewandt, um den Teilgraphen mit dem Wurzelknoten $high(u)$ zu erhalten.

Angenommen $var(r_f) = x_i$ und r_g ist ein Blatt oder $var(r_g) > x_i$, dann ist die vom Graphen dargestellte Funktion mit der Wurzel r_g unabhängig von x_i. Ist dies der Fall, wird ein Knoten u mit $var(u) = x_i$ erzeugt und der Algorithmus rekursiv auf $low(r_f)$ sowie r_g angewandt, um den Teilgraphen mit der Wurzel $low(u)$ zu erhalten. Dasselbe wird für $high(r_f)$ und r_g durchgeführt, um den Teilgraphen mit der Wurzel $high(u)$ zu erhalten. Um alle Fälle abzudecken, müssen nur die Rollen der beiden Wurzelknoten r_f und r_g getauscht werden [1].

Zu weiteren Implementierungsdetails siehe Andersen [11]. Wie die Ausführung der "Apply"-Methode für zwei kleine ROBDDs aussieht, zeigt die Abbildung 2.2 exemplarisch.

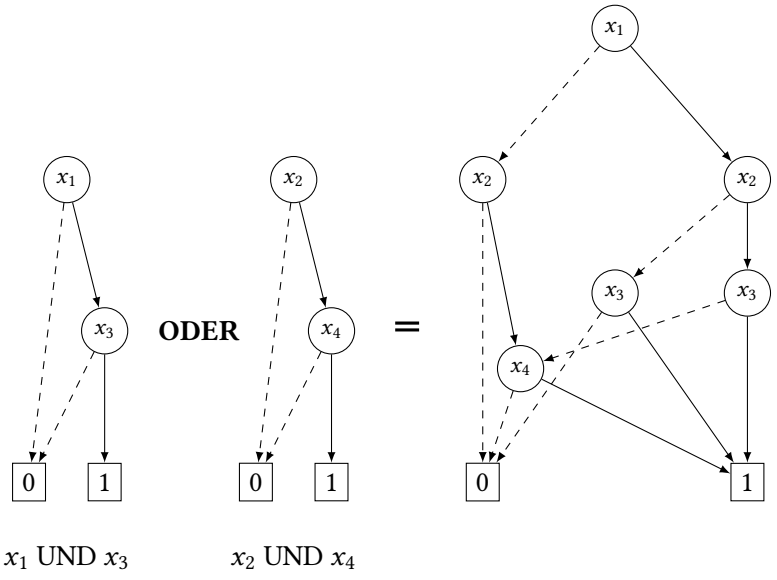


Abbildung 2.2: Beispiel der "Apply"-Methode mit einer "oder"-Operation; die Variablen-Ordnung ist: $x_1 < x_2 < x_3 < x_4$ [1].

Komplexität zur Auffindung der optimalen Variablen-Ordnung: Die Komplexität von Algorithmen zur Manipulation von BDDs hängt stark von der Größe des BDD-

Graphen ab. Aus diesem Grund sollte die Größe des Graphen bei optimaler Variablen-Ordnung möglichst minimal sein. Allerdings ist das Herausfinden einer optimalen Variablen-Ordnung NP-vollständig. Da es $n!$ verschiedene Variablen-Ordnungen für eine Funktion mit n Variablen gibt. Der am effizientesten arbeitende bekannte Algorithmus weist eine Komplexität von $O(n^2 \cdot 3^n)$ für die optimale Variablen-Ordnung auf [15].

Kapitel 3

Anforderungen

Dieses Kapitel beinhaltet eine Analyse der an das System gestellten Anforderungen. Diese Anforderungen – A1 bis A11 – werden sowohl für die qualitative Evaluierung als auch für den Vergleich mit den verwandten Arbeiten des Systems verwendet.

A1 Unabhängigkeit von Topologie
Diese Anforderung verlangt, dass das System für jegliche Netzwerk-Topologie geeignet ist. Dies betrifft sowohl die Datenstruktur als auch die Algorithmen, die vom System verwendet werden.

A2 Abhängigkeiten abbilden
Bei dieser Anforderung handelt es sich um Abhängigkeiten zwischen Komponenten des Netzwerks. Diese Abhängigkeit kann auch als Hierarchie zwischen Komponenten bezeichnet werden. Dies betrifft in erster Linie Virtualisierung und Dienste, da auf einem Computer-System mehrere VMs sowie Dienste laufen können. Diese Abhängigkeiten muss das System entsprechend abbilden können.

A3 Zuverlässigkeit von k-Terminals bestimmen
Das System soll einen Algorithmus zur Berechnung der Zuverlässigkeit von k-Terminals zur Verfügung stellen, wobei $2 \leq k \leq n$ gilt und n der Anzahl der Komponenten des Netzwerks entspricht.

A4 Berücksichtigung von Komponenten
Das System soll bei der Berechnung der Zuverlässigkeit den Ausfall von Netzwerk-Komponenten berücksichtigen. Dafür muss für jede Komponente eine Ausfallwahrscheinlichkeit angegeben werden.

A5 Berücksichtigung von Verbindungen
Das System soll bei der Berechnung der Zuverlässigkeit den Ausfall von Verbindungen zwischen einzelnen Komponenten berücksichtigen. Dafür muss für jede Verbindung eine Ausfallwahrscheinlichkeit angegeben werden.

A6 Zuverlässigkeit pro Komponente bestimmen
Diese Anforderung bedeutet, dass für jede Komponente individuell die Zuverlässigkeit berechnet werden kann. Dafür soll für jede Komponente v eine Menge an K-Komponenten angegeben werden können, die für die Bestimmung der Zuverlässigkeit der Komponente v relevant sind. Diese K-Komponenten bilden zusammen mit der Komponente v die k-Terminals.

A7 Berücksichtigung der Redundanzen von Komponenten, die für die Zuverlässigkeit relevant sind
Zur Bestimmung der Zuverlässigkeit einer Komponente müssen die für sie relevanten Komponenten (k-Terminals) erreichbar sein. Falls eine oder mehrere dieser Komponenten eine oder mehrere Redundanzen besitzen, müssen diese für die Berechnung der Zuverlässigkeit vom System berücksichtigt werden.

A8 Identifizierung kritischer Entitäten
Das System soll die Möglichkeit besitzten, für eine Komponente, deren Zuverlässigkeit bestimmt wurde, kritische Entitäten identifizieren zu können. Entitäten beschreiben sowohl Komponenten als auch ihre Verbindungen untereinander.

A9 Erhöhen der Zuverlässigkeit einer Komponente durch Redundanzen
Durch das Einfügen von Redundanzen für Entitäten soll die Zuverlässigkeit einer bestimmten Komponente auf einen gewünschten Wert angehoben werden können.

A10 Verringern der Zuverlässigkeit einer Komponente
Das System soll durch das Entfernen von Komponenten und Verbindungen die Zuverlässigket einer bestimmten Komponente auf einen gewünschten Wert reduzieren. Dabei sollen jene Entitäten entfernt werden, deren Einfluss auf die Zuverlässigkeit am geringsten ist, d. h. es sollen die unkritischsten Entitäten identifiziert und anschließend entfernt werden bis ein gewünschter Zuverlässigkeitswert erreicht wird.

A11 Vorschläge zur Positionierung einer Redundanz eines k-Terminals
Bei dieser Anforderung soll das System Vorschläge zur Positionierung einer Redundanz für ein k-Terminal einer bestimmten Komponente unterbreiten. Zusätzlich soll zu der jeweiligen Positionierung die Zuverlässigkeit, die sich durch das Einfügen der Redundanz ergibt, angegeben werden. Dabei muss angegeben werden, welche Komponente eine Menge von k-Terminals besitzt und von welchem k-Terminal die Redundanz erzeugt werden soll. Das Ergebnis soll eine Liste sein, die die Positionen (mit welcher Komponente die Redundanz verbunden werden soll) und die entsprechenden Zuverlässigkeitswerte beinhaltet.

Kapitel 4

Verwandte Arbeiten

Dieses Kapitel befasst sich mit verwandten Arbeiten bezüglich der Thematik dieser Thesis. Dafür ist das Kapitel in vier Abschnitte unterteilt. Der erste Abschnitt befasst sich mit Arbeiten zur Berechnung der Zuverlässigkeit von k-Terminals. Im zweiten Abschnitt werden verwandte Arbeiten angeführt, die sich mit den kritischen Entitäten eines Netzwerks beschäftigen. Der dritte Abschnitt führt Arbeiten zum Thema Verbesserung der Zuverlässigkeit an. Im letzten Abschnitt werden die verwandten Arbeiten in einer Tabelle gegenübergestellt und auf Basis der Anforderungen (siehe Kapitel 3) verglichen.

4.1 Berechnung der Zuverlässigkeit von k-Terminals

In diesem Abschnitt werden verschiedene Ansätze beschrieben, wie die Zuverlässigkeit von k-Terminals bestimmt werden kann.

Das 2-Terminal-Problem wurde bereits 1965 von Barlow und Proschan in dem Buch "Mathematical Theory of Reliability" erwähnt [16]. Das Berechnen der Zuverlässigkeit von k-Terminals mit intakten Knoten und Kanten, die unabhängig voneinander ausfallen können, wurde in der Literatur für gerichtete und ungerichtete Graphen bereits vielfach untersucht [17]. Solche Netzwerke, in denen nur die Kanten ausfallen können, werden von Yeh et al. [2] als "link networks" bezeichnet.

Valiant [5] hat gezeigt, dass das 2-Terminal-Problem für gerichtete und ungerichtete "link networks" #P-vollständig ist. Provan und Ball [18] haben dies ebenfalls bewiesen. Ball [6] gibt in dem Paper "Computational Complexity of Network Reliability Analysis: An Overview" einen Überblick zur Komplexität der Berechnung der Zuverlässigkeit eines Netzwerks. Dabei beschreibt er auch die Komplexitätsklassen des 2-, des k- und des All-Terminal-Problems, die alle NP-schwer sind.

Bei der Berechnung der Zuverlässigkeit von k-Terminals gibt es Methoden, die eine exakte Berechnung durchführen, und solche, die durch eine Approximation die Zuverlässigkeit bestimmen. Ayoub et al. [19] beschreiben eine Approximationsmethode zur Bestimmung der Zuverlässigkeit von k-Terminals für ein ungerichtetes Graph-Modell. In dem Paper wird ein Algorithmus beschrieben, der eine Monte-Carlo-Simulation mit einer Breitensuche anwendet. Die Kanten des Netzwerks können unabhängig voneinander ausfallen und die Knoten sind immer intakt.

Da es verschiedene Methoden gibt, um die Zuverlässigkeit von k-Terminals exakt zu berechnen, wird eine Kategorisierung nach Lê et al. [20] vorgenommen.

Kategorie 1: Pfad- und Cut-Enumeration
In dieser Kategorie werden die minimalen Pfade und Cuts enumeriert und anschließend mengentheoretisch durch Techniken zur Bestimmung der Summe disjunkter Produkte manipuliert [21,22], um in Pfad- und Cut-Terme zerlegt zu werden. Cuts werden seit den 1960er Jahren verwendet, um die Zuverlässigkeit eines Netzwerks zu berechnen [23].

In vielen praktischen Systemen ist die Anzahl der Cuts um vieles kleiner als die Anzahl der Pfade [24]. Es wurde gezeigt, dass auf Cuts basierende Algorithmen oftmals wesentlich bessere Performanz erzielen als pfadbasierte Algorithmen [25]. Für große Netzwerke wird diese Methode unpraktisch, da die Anzahl der minimalen Pfade/Cuts im Verhältnis zur Größe des Netzwerks exponenziell zunimmt [2, 20].

Kategorie 2: Faktorisieren mit Reduktions-Techniken
Laut [2] war Moskowitz [26] der erste, der das Faktorisierungstheorem angewandt hat, um die Zuverlässigkeit eines Netzwerks zu bestimmen. Satyanarayana et al. [27] haben gezeigt, dass es effizienter ist, das Faktorisieren mit Reduktions-Techniken zu verwenden. Die Zuverlässigkeit von k-Terminals R_K eines Netzwerks G kann durch das Anwenden des Faktorisierungstheorems mit folgendem Ausdruck rekursiv berechnet werden:

$$R_K(G) = p(e_i) \cdot R_{K'}(G * e_i) + (1 - p(e_i)) \cdot R_K(G - e_i).$$

Dabei stellt $p(e_i)$ die Ausfallsicherheit der Kante $e_i = (u, v)$ und K die Menge der k-Terminals dar. $G * e_i$ ist der Teilgraph von G mit intakter Kante e_i und $G - e_i$ ist jener Teilgraph von G, bei dem die Kante e_i entfernt wurde. Es gilt $K' = K$ falls $u, v \notin K$ oder $K' = K - u - v + u \cup v$ falls $u \in K$ oder $v \in K$ [20].

Die Komplexität kann durch das Anwenden verschiedener Reduktions-Techniken, wie z. B. "series-parallel"- oder "polygon-to-chain"-Reduktion, verringert werden [28]. Dabei wird die Größe des Netzwerks reduziert, während ihre Zuverlässigkeit erhalten bleibt. Satyanarayana und Wood [29] zeigen, wie die Zuverlässigkeit von k-Terminals für "series-parallel"-Graphen in linearer Zeit berechnet werden kann. Die Versuchsergebnisse von Theologou und Carlier [30] haben die Effizienz des Faktorisierens mit

Reduktions-Techniken gezeigt. Zusätzlich betrachten sie den Ausfall von Knoten in einem Netzwerk.

Da die Komplexität des Faktorisierens mit Reduktions-Techniken für größere und dichtere Netzwerke exponenziell wächst, ist die Berechnung bereits für mittelgroße Netzwerke – bestehend aus maximal 30 Kanten – nicht praktikabel. Ein weiterer Nachteil ist, dass eine Refaktorisierung des Netzwerk-Graphen durchgeführt werden muss, sobald sich die Ausfallwahrscheinlichkeit weniger Kanten ändern [20].

Kategorie 3: Binary Decision Diagram (BDD)
Um diesen Nachteil zu umgehen, wurden Methoden entwickelt, die BDDs verwenden [3, 31]. BDDs werden als State of the Art-Datenstruktur bezeichnet, um große boolesche Ausdrücke effizient zu speichern. Laut [20] wurden BDDs von Akers [32] zum ersten Mail eingeführt. Ihr volles Potenzial wurde allerdings erst durch Bryant [33] entfaltet, indem er wichtige Erweiterungen einführte, wie die Variablen-Ordnung und das gemeinsame Benutzen isomorpher Teilgraphen.

Kuo, Lu und Yeh (KLY) verwendeten Ordered BDDs (OBDDs) und zeigten einen Weg, um redundante Berechnungen durch das Erkennen isomorpher Teilgraphen [3, 34] zu umgehen. Sobald das OBDD erzeugt wurde, können beliebige Perfomanz-Messungen durchgeführt werden, indem der OBDD-Graph durchgegangen wird. Gemäß den Ergebnissen von [3, 34] ist die KLY-Methode bezüglich Zeit und Speicher effizienter als die Algorithmen der Kategorie 1 und 2.

Kuo, Yeh und Lin haben in dem Paper "Efficient and Exact Reliability Evaluation for Networks With Imperfect Vertices" [2] eine Methode (CAE – "Composition after Expansion") vorgestellt, um die Zuverlässigkeit von k-Terminals für gerichtete und ungerichtete Netzwerke zu berechnen, in denen sowohl die Kanten als auch die Knoten ausfallen können. Wie diese Methode genau funktioniert, kann dem Abschnitt 5.3.2 entnommen werden.

Für ungerichtete Netzwerke ist die Decomposition-Methode eine weitere sehr effiziente, auf BDDs basierende Methode zur Berechnung der Zuverlässigkeit von k-Terminals. Sie basiert auf der Idee von Rosenthal, in der unterschiedliche Netzwerk-Zustände durch eine Menge von Grenzknoten F ("Frontier Nodes") klassifiziert werden [20]. Die Komplexität der Decomposition-Methode hängt dabei hauptsächlich von der Kardinalität der maximalen Menge der Grenzknoten F_{max} ab [35]. Die Zuverlässigkeit des Netzwerks kann mit einem begrenzten $|F_{max}|$ in einem Zeitraum linear zur Anzahl der Kanten berechnet werden [31].

Im Allgemeinen hängt die Komplexität der BDD-basierten Algorithmen zur Berechnung der Zuverlässigkeit stark von der Größe des BDDs ab. Die Größe des BDDs ist wiederum stark von der gewählten Variablen-Ordnung abhänging, und das Bestimmen der optimalen Variablen-Ordnung ist mit $O(n^2 \cdot 3^n)$ NP-vollständig (siehe Punkt 2.2). Daher werden Heuristiken für die Variablen-Ordnung der BDDs verwendet.

Kuo et al. [3] und Hardy et al. [31] haben durch empirische Ergebnisse gezeigt, dass eine Variablen-Ordnung basierend auf der Breitensuche in den meisten Fällen besser geeignet ist als eine Variablen-Ordnung basierend auf der Tiefensuche.

Lê et al. [20] beschreiben für die Decomposition-Methode eine neue Heuristik zur Bestimmung der Variablen-Ordnung für ein BDD. Durch diese Heuristik kann für eine Vielfalt von Netzwerk-Topologien – im speziellen für unregelmäßige Netzwerke – die Größe des BDDs signifikant reduziert werden. Da dies die Größe des BDDs reduziert, ist diese Verbesserung für alle Methoden, die BDDs verwenden, wie z. B. die KLY-Methode, zur Bestimmung der Zuverlässigkeit von k-Terminals nützlich.

Lê beschreibt in seiner Dissertation "Quantitative evaluation of network reliability" [1] detailliert sowohl exakte Methoden als auch Approximationsmethoden zur Berechnung der Zuverlässigkeit von k-Terminals. Die exakten Methoden, die er beschreibt, sind folgende:

- Pfad- und Cut-Enumeration
- Faktorisieren mit Reduktions-Techniken
- KLY-Methode
- Decomposition-Methode

Zusätzlich beschreibt Lê, wie die KLY- und die Decomposition-Methode verbessert werden können und evaluiert dies.

Wie die KLY-Methode verbessert werden kann, wird auch in dem Paper "Improving the Kuo-Lu-Yeh algorithm for assessing Two-Terminal Reliability" von Lê et al. [36] beschrieben. Da diese Verbesserung in dieser Thesis für die Berechnung der Zuverlässigkeit angewandt wird, kann die detaillierte Beschreibung ihrer Funktionsweise dem Punkt 5.3.1.2 entnommen werden.

Herrmann und Soh [37] beschreiben einen Algorithmus, der die Zuverlässigkeit von k-Terminals – mit intakten Knoten und Kanten, die ausfallen können – speichereffizient berechnen kann durch die Verwendung der Decomposition-Methode mit Augmented Ordered Binary Decision Diagram (OBDD-A). Dabei zeigen sie, dass der Speicherverbrauch für Netzwerke unterschiedlicher Größe, aber gleicher Konnektivitäts-Struktur konstant bleibt, d. h. sie haben denselben Speicherverbrauch für ein 3x12 und ein 3x1000 Grid-Netzwerk.

Um dies zu ermöglichen, speichert das OBDD-A die Ausfallwahrscheinlichkeiten in jedem OBDD-A-Knoten, wodurch die Zuverlässigkeit der jeweiligen OBDD-A-Kinder-Knoten direkt berechnet werden kann. Jeder OBDD-A-Knoten wird nur einmal durchgegangen und anschließend verworfen, wodurch der Algorithmus zu jeder Zeit weniger als zwei komplette Ebenen des OBDD-As speichern muss [37].

Der Nachteil dieser Methode ist, dass das generierte BDD nicht weiterverwendet werden kann, um z. B. den Ausfall von Knoten zu betrachten oder kritische Entitäten des Netzwerks zu bestimmen.

In einem weiteren Paper [38] beschreiben Herrmann und Soh einen Algorithmus, der die benötigte Zeit für die Berechnung der Zuverlässigkeit reduzieren kann. Zusätzlich vergleichen sie die benötigte Zeit für diesen Algorithmus mit dem Algorithmus aus [37].

Mit dem Werkzeug ResiNeT2[1], das in der Master Thesis von Sprotte [39] beschrieben wird, kann sowohl die Zuverlässigkeit von k-Terminals als auch die Resilienz eines Netzwerks bestimmt werden. Dabei kann die Zuverlässigkeit von drei verschiedenen Algorithmen berechnet werden. Einer setzt auf das Faktorisieren ohne Reduktions-Techniken, die beiden anderen sind die Algorithmen von Abraham [40] und Heidtmann, die beide auf der Methode der Pfad- und Cut-Enumeration aufbauen. In diesem Paper [41] vergleicht Heidtmann seinen Algorithmus mit dem von Abraham und zeigt dabei, dass Heidtmanns Algorithmus wesentlich effizienter ist.

Iwashita et al. [42] schildern eine Methode zur Berechnung der Anzahl der Pfade zwischen zwei Knoten s und t eines Graphen G. Diese Methode basiert auf dem von Knuth [43, exercise 225, ch. 7.1.4] entwickelten Algorithmus "SIMPATH". Dieser Algorithmus erzeugt ein Zero-Suppressed Binary Decision Diagram (ZBDD), um alle möglichen Pfade zwischen zwei Knoten in einem Graphen darzustellen. ZBDDs sind eine Variante von BDDs, wobei die Reduktionsregel bezüglich dem Löschen eines BDD- und eines ZBDD-Knotens nicht ident ist.

Ein BDD-Knoten kann entfernt werden, falls beide Kinder-Knoten auf den gleichen BDD-Knoten zeigen. Im Gegensatz dazu kann ein ZBDD-Knoten entfernt werden, falls der Kinder-Knoten "high" auf den Terminal-Knoten 0 (False) zeigt [42]. Die Methode von Iwashita et al. arbeitet sehr effizient, wodurch sie die Anzahl der Pfade eines 15x15 Grid-Netzwerks in wenigen Minuten bestimmt haben. Mit ihrer Methode wird zwar nicht die Zuverlässigkeit eines Netzwerks bestimmt, sie könnte allerdings für diesen Zweck entsprechend erweitert werden.

In [44] beschreibt Lin wie die "Mission"-Zuverlässigkeit eines stochastischen Flussnetzwerks bestimmt werden kann. "Mission"-Zuverlässigkeit wird die Möglichkeit gennant, dass eine bestimmte Menge an multi-commodity durch ein Informations-Netzwerk gesendet werden kann, während die Kosten beschränkt sind. Dabei werden die Knoten und Kanten für die Bestimmung der Zuverlässigkeit berücksichtigt.

In [45] beschreiben Menth et al. das Tool ResiLyzer (Resilience Analyzer) [46], das Risiken in paketvermittelnden Kommunikationsnetzen quantifiziert. Solche Risiken sind z. B. Veränderungen des Verkehrsaufkommens oder der Netztopologie (z. B. Ausfall

[1] Java-Applet, das online auf folgender Seite abrufbar ist:
http://www.informatik.uni-hamburg.de/TKRN/world/tools/ResiNet2/index.html [abgerufen am 14. Juli 2015]

von Komponenten). Durch ein entwickeltes Modell zur Risikoanalyse [47] können gut vergleichbare und aussagekräftige Metriken, die für die Beschreibung der Ausfallsicherheit eines Netzes notwendig sind, bestimmt werden. Mit dem ResiLyzer kann z. B. die 2-Terminal-Zuverlässigkeit sowie eine potenzielle Linküberlast berechnet werden.

4.2 Identifizieren kritischer Entitäten

In diesem Abschnitt werden verwandte Arbeiten zum Thema kritische Entitäten eines Netzwerks vorgestellt.

Wie Hardy et al. [48] beschreiben, war Zygmund Birnbaum 1969 der erste, der ein Konzept zur Wichtigkeit von Komponenten aufstellte. Die Grundlage für das Konzept war, dass manche Komponenten für die Bereitstellung gewisser System-Eigenschaften wichtiger sind als andere. Die beiden bekanntesten Maßeinheiten zur Bestimmung der Wichtigkeit einer Komponenten sind [48]:

- *Birnbaum importance measure* – Sie kann für eine Komponente e_k folgendermaßen angegeben werden:

  $$I_k^B = R(G/e_k \text{ is up}; p) - R(G/e_k \text{ is down}; p).$$

  Wobei R die Zuverlässigkeitsfunktion für das All-Terminal-Problem, G den Graph und p den Vektor für die Ausfallsicherheit der Komponenten angibt.

  Diese Einheit gibt durch die Zuverlässigkeit der einzelnen Komponenten ihre Relevanz bezüglich der Zuverlässigkeit des Systems an. Jene Komponente, deren Abweichung am größten ist, weist die höchste Wichtigkeit auf.

  Dabei ist diese Maßeinheit unabhängig von der Nichtverfügbarkeit der Komponente e_k. Wodurch Komponenten, die sehr wahrscheinlich nicht ausfallen und schwierig zu verbessern sind, eine hohe Wichtigkeit zugewiesen werden kann.

- *Criticality importance measure* – diese Maßeinheit berücksichtigt Komponenten, die nicht nur kritisch bezüglich der Zuverlässigkeit des Systems sind, sondern auch eine höhere Ausfallwahrscheinlichkeit haben. Das Ergebnis ist somit ein kombinierter Wert der Komponente aus der Relevanz für die System-Zuverlässigkeit und ihrer Ausfallwahrscheinlichkeit. Hardy et al. [48] geben zur Berechnung der "Criticality importance measure" folgende Formel an:

  $$I_k^C = I_k^B \cdot \frac{p_k}{R(G; p)}. \tag{4.1}$$

  Schlüssiger erscheint jedoch in der Gleichung 4.1 anstatt p_k die Ausfallwahrscheinlichkeit q_k der Komponente e_k zu verwenden, da dadurch jene Komponen-

ten einen höheren Wert erhalten, deren Ausfall eine höhere Wahrscheinlichkeit aufweist und somit denen eine höhere Wichtigkeit zukommt. Diese Hierarchie entspricht der Reihung von groß nach klein bei der "Birnbaum importance measure".

Da Hardy et al. [48] eine Maßeinheit für die Relevanz einer Komponente benötigen, die die Kosten der jeweiligen Komponente berücksichtigt, haben sie dafür eine Einheit definiert:

$$I_k^N = \frac{1}{c_k} \cdot (R(G;p) - R(G/e_k \text{ is down};p)), \tag{4.2}$$

wobei c_k die Kosten der Komponente e_k angibt.
Diese Maßeinheit berücksichtigt die Kosten von e_k und die Verringerung der Zuverlässigkeit des Netzwerks, falls die Komponente e_k ausfällt.

Rebaiaia und Ait-Kadi [49, Ap. C] definieren die Relevanz der Komponente k mit der Erhöhung der System-Zuverlässigkeit durch die Verbesserung der Ausfallsicherheit von k. Dabei wird ebenfalls die "Birnbaum importance measure" eingesetzt. Die folgende Formel bildet dies ab:

$$I_k^\Delta = (p'_k - p_k) \cdot I_k^B, \tag{4.3}$$

wobei p'_k die verbesserte Ausfallsicherheit angibt. Angenommen, die Komponente k erhält eine Redundanz, die die gleiche Ausfallsicherheit wie k aufweist. Dadurch ergibt sich für die verbesserte Ausfallsicherheit:

$$p'_k = 1 - (1 - p_k)(1 - p_k) = 2p_k - p_k^2 - p_k = p_k q_k$$

Dies kann nun in die Gleichung 4.3 eingesetzt werden:

$$I_k^\Delta = p_k q_k \cdot I_k^B$$

Yeh et al. [2] beschreiben einen Algorithmus, um die "Essential Variable" eines Netzwerks zu identifizieren. Dabei handelt es sich um eine Variable – Knoten oder Kante – des Netzwerks, bei deren Entfernung die Zuverlässigkeit um den höchsten Wert sinkt. Dies bedeutet, dass die "schwächste" Entität (weakest link) hinsichtlich der Zuverlässigkeit des Netzwerk identifiziert wird.

Basierend darauf kann die Zuverlässigkeit des Netzwerks durch das Hinzufügen von Redundanzen gesteigert werden. Zusätzlich können mit diesem Ansatz die für die Zuverlässigkeit am wenigsten kritischen Entitäten bestimmt werden. Diese Information kann verwendet werden, um die Zuverlässigkeit eines Netzwerks schrittweise zu verringern. Daher wird der Ansatz von Yeh et al. (in einer adaptierten Version) in der vorliegenden Arbeit zur Identifikation der kritischen Entitäten in Punkt 5.4.1 verwendet.

4.3 Verbesserung der Zuverlässigkeit

Dieser Abschnitt befasst sich mit Arbeiten, die Methoden zur Verbesserung der Zuverlässigkeit beschreiben.

Abd-El-Barr et al. [50] beschreiben eine Methode, bei der die Topologie eines Rechnernetzes durch die Auswahl einer Teilmenge von möglichen Kanten optimiert wird. Dabei sollen die Fehlertoleranz sowie die Zuverlässigkeit maximiert werden, während die Kosten beschränkt sind. In diesem Paper wird sowohl ein Algorithmus für die Optimierung der 2-Terminal-Zuverlässigkeit als auch ein Algorithmus für die Optimierung der All-Terminal-Zuverlässigkeit vorgestellt, während die Fehlertoleranz des Netzwerks erhöht wird.

In [48] beschreiben Hardy et al. einen Algorithmus – basierend auf einer Heuristik –, der die Netzwerkkosten unter Berücksichtigung einer angegebenen Zuverlässigkeit minimiert. Dabei wird die All-Terminal-Zuverlässigkeit durch einen BDD-basierten Algorithmus bestimmt. Eine Menge an Knoten, die nicht ausfallen können, und deren Topologie sowie eine Menge an möglichen Kanten werden als gegeben angenommen. Für Kanten wird sowohl eine Ausfallsicherheit als auch deren Kosten angegeben.

Der beschriebene Algorithmus startet mit einem theoretisch maximalen (bezogen auf die Zuverlässigkeit mit den gegebenen Netzwerk-Informationen) Netzwerk und entfernt so lange Kanten, bis die Zuverlässigkeit ihren minimal zulässigen Wert erreicht. Die zu entfernenden Kanten werden auf Grundlage der zuvor beschriebenen Maßeinheit I_k^N aus Gleichung 4.2 bestimmt. Wobei jene Kanten, die den geringsten Wert aufweisen, ausgewählt werden. Im nächsten Schritt werden Kanten durch bereits entfernte Kanten, deren Kosten geringer sind und die die minimal Zuverlässigkeit erfüllen, ersetzt [48].

Elshqeirat et al. [51] beschreiben ebenfalls einen Algorithmus, der die Kosten minimiert unter Berücksichtigung einer angegebenen Zuverlässigkeit, die das Netzwerks mindestens aufweisen soll. Der Algorithmus basiert auf dem Ansatz der dynamischen Programmierung. Das Paper zeigt, dass die Methode von Elshqeirat et al. immer eine praktikable Lösung bereitstellt.

Im Paper [52] beschreiben Elshqeirat et al. einen Algorithmus, der die Zuverlässigkeit maximiert, während die Kosten beschränkt sind. Bei dieser Methode setzen sie ebenfalls auf den Ansatz der dynamischen Programmierung.

4.4 Gegenüberstellung der Arbeiten

In diesem Abschnitt werden die verwandten Arbeiten in Tabelle 4.1 gegenübergestellt und auf Basis der Anforderungen an das entwickelte System verglichen.

Quelle	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11
Yeh et al. [2, 3, 34]	+	−	+	+	+	−	−	+	−	−	−
Ayoub et al. [19]	+	−	+	−	+	−	−	−	−	−	−
Lê [1] und Lê et al. [20, 36]	+	−	+	+	+	−	−	−	−	−	−
Hardy et al. [31, 48]	+	−	+	−	+	−	−	+	+	−	−
Herrmann et al. [37, 38]	+	−	+	−	+	−	−	−	−	−	−
Sprotte [39]	+	−	+	−	+	−	−	−	−	−	−
Lin [44]	+	−	−	+	+	−	−	−	−	−	−
Menth et al. [45–47]	+	−	−	+	+	−	−	+	+	−	−
Abd-El-Barr et al. [50]	+	−	−	−	+	−	−	−	+	−	−
Elshqeirat et al. [51, 52]	+	−	−	−	+	−	−	−	+	−	−

Tabelle 4.1: Vergleich verwandter Arbeiten mit den Anforderung an das entwickelte System

Kapitel 5

Entwurf

Dieses Kapitel beschreibt den Algorithmus für die Berechnung der Zuverlässigkeit eines Netzwerks sowie das Vorschlagsystem. Des Weiteren werden die dafür benötigten Datenstrukturen beschrieben.

5.1 Aufbau der algorithmischen Struktur

In diesem Punkt wird der Aufbau der algorithmischen Struktur beschrieben. Dies wird in drei Punkte - Eingabe, Verarbeitung, Ausgabe - unterteilt.

5.1.1 Eingabe

Als Eingabe dient ein Netzplan mit unterschiedlichen Komponenten und entsprechenden Verbindungen untereinander. Dafür soll der Netzplan als Graph mit Knoten und Kanten als Komponenten und Verbindungen dargestellt werden. Des Weiteren müssen benötigte Zusatzinformationen für die einzelnen Netzentitäten angegeben werden. Für jede Entität muss eine Ausfallsicherheit $p(x)$ gegeben sein.

Bei einer Komponente kann angegeben werden, ob diese eine oder mehrere Redundanzen besitzt und welche dies sind. Des Weiteren kann für jede Komponente v eine Menge an K-Komponenten angegeben werden, die für die Bestimmung der Zuverlässigkeit der Komponente v erreichbar sein müssen. Falls die Zuverlässigkeit bestimmter Komponenten einen entsprechnenden Wert aufweisen soll, muss dies durch Zusatzinformationen angegeben werden.

5.1.2 Verarbeitung

In diesem Abschnitt wird der eingegebene Netzplan entsprechend verarbeitet und für die Ausgabe bereitgestellt. Dies entspricht dem Kern des Systems. Dabei wird zuerst die Zuverlässigkeit jeder Komponente, die eine Menge an K-Komponenten besitzt, berechnet, wodurch die Gesamtzuverlässigkeit des Netzwerks bestimmt werden kann.

Anschließend können verschiedene Algorithmen auf das Netzwerk angewandt werden, um z. B. kritische Entitäten zu bestimmen oder die Zuverlässigkeit bestimmer Komponenten auf einen gewünschten Wert zu erhöhen oder zu verringern.

5.1.3 Ausgabe

Die Ausgabe des Systems entspricht wieder einem Netzplan, der je nach Auswahl der anzuwendenden Algorithmen zusätzliche Redundanzen zur Erhöhung der Zuverlässigkeit besitzt. Andererseits kann die Zuverlässigkeit durch Entfernung von Komponenten bzw. Verbindungen reduziert werden. Weiterhin wird pro Komponente, die eine Menge an K-Komponenten besitzt, die berechnete Zuverlässigkeit sowie die Gesamtzuverlässigkeit des Netzwerks angegeben.

5.2 Datenstruktur

Für die Algorithmen, die in diesem Kapitel beschrieben werden, werden zwei Datenstrukturen benötigt, um die Anforderungen erfüllen zu können. Es wird eine Datenstruktur für die Darstellung des Netzwerks als Graph sowie die BDD-Datenstruktur benötigt. In den zwei folgenden Punkten werden beide beschrieben.

5.2.1 BDD

Die Beschreibung der BDD-Datenstruktur inklusive Beispiele befindet sich im Grundlagen-Kapitel in Punkt 2.2. Für die verschiedenen beschriebenen Algorithmen zur Berechnung der Zuverlässigkeit und für das Vorschlagsystem werden in erster Linie die logischen Operationen "und" und "oder" verwendet, um BDD-Knoten miteinander zu verknüpfen. Implementierungsdetails können Punkt 6.2.2 entnommen werden.

5.2.2 Netzwerk-Graph

Diese Datenstruktur wird für die Darstellung des Netzwerks als Graph benötigt. Ein Graph $G = (V, E)$ besitzt eine Menge von Knoten V und eine Menge von Kanten

$E \subset VxV$. Da das Netzwerk Zyklen aufweisen kann und ungerichtet ist, handelt es sich um einen ungerichteten Graphen.

Die Datenstruktur ist dafür geeignet Mehrfachkanten abzubilden, da dies in realen Netzwerken vorkommen kann. Allerdings wird bei der Abbildung einer Redundanz für eine Kante keine Mehrfachkante hinzugefügt, sondern nur die Zuverlässigkeit der ursprünglichen Kante erhöht. Dies erspart Rechenaufwand bei der Berechnung der Zuverlässigkeit eines Netzwerks, da pro Mehrfachkante ein logisches "oder" für den jeweilig folgenden BDD-Knoten ausgeführt werden muss. Dies wird mit folgendem Beispiel veranschaulicht:

Angenommen, die Kante e_0 des ursprünglichen Netzwerk-Graphen G der Abbildung 5.7 soll eine Redundanz e'_0 erhalten. Abbildung 5.1 zeigt drei BDD-Graphen inklusive der Berechnung der Zuverlässigkeit von einem Quellknoten s zu einem Zielknoten t. Der obere BDD-Graph repräsentiert den ursprünglichen Netzwerk-Graphen G. Der linke bzw. rechte BDD-Graph zeigt die Auswirkung des Einfügens der Redundanz e'_0 der Kante e_0, wobei beim linken beide verknüpft werden, um die Ausfallsicherheit zu erhöhen. Beim rechten BDD-Graphen wird e'_0 als Mehrfachkante eingefügt, wodurch ein weiterer Pfad entsteht und somit ein logisches "oder" ausgeführt werden muss. Dadurch erhöht sich die Anzahl der Knoten des BDDs.

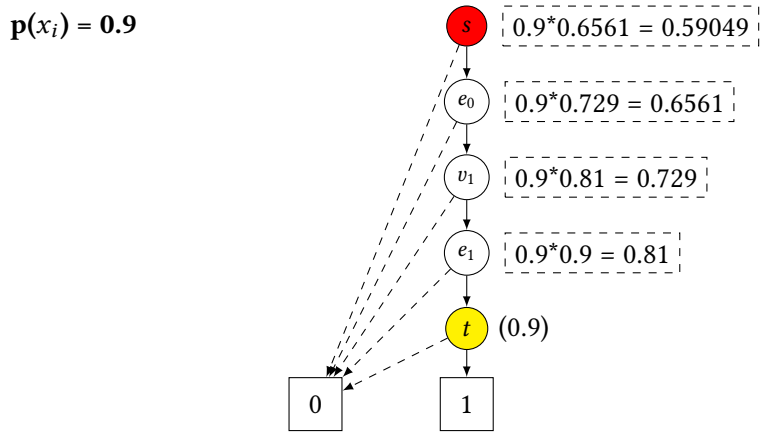
Die Knoten und Kanten des Graphen besitzen jeweils die Information über ihre Ausfallsicherheit $p(x)$. Falls für einen Knoten die Zuverlässigkeit bestimmt werden soll, muss dieser eine Menge an K-Knoten aufweisen, wobei der Knoten selbst in jener Menge enthalten ist. Aus diesem Grund besitzt die Datenstruktur des Knotens die Möglichkeit eine solche Menge $K \subseteq V$ zu speichern.

Eine Kante besitzt entsprechende Referenzen zu ihren Quell- und Zielknoten. Im Gegensatz dazu besitzt ein Knoten v keine Information über seine Kanten. Diese Information wird in der Graph-Datenstruktur gespeichert. Dadurch müssen keine Änderungen am Knoten direkt vorgenommen werden, falls Kanten hinzugefügt oder entfernt werden.

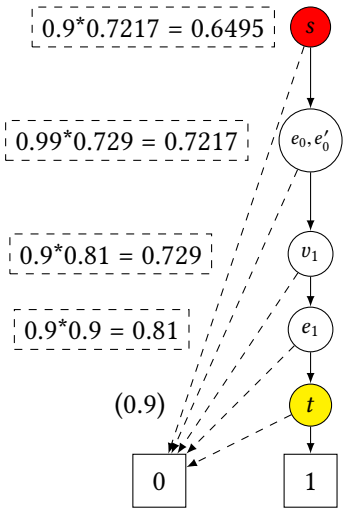
Des Weiteren ist die Abbildung von redundanten Knoten ebenfalls in der Datenstruktur des Graphen gespeichert und nicht im Knoten selbst. Dies hat den gleichen Vorteil wie die Abbildung der Kanten zuvor. Die Datenstruktur des Knotens muss beim Hinzufügen bzw. Entfernen von Redundanzen nicht verändert werden.

Bei der Berechnung der Zuverlässigkeit bzw. im Algorithmus 5.2 "PathConstruct" werden alle möglichen Wege von einer Quelle s zu einem Ziel t bestimmt. Hierfür wird mit dem Ausdruck $sub_G = G * e$ ein neuer Graph gebildet, der den momentanen Quellknoten s sowie alle seine Kanten aus dem Graphen entfernt.

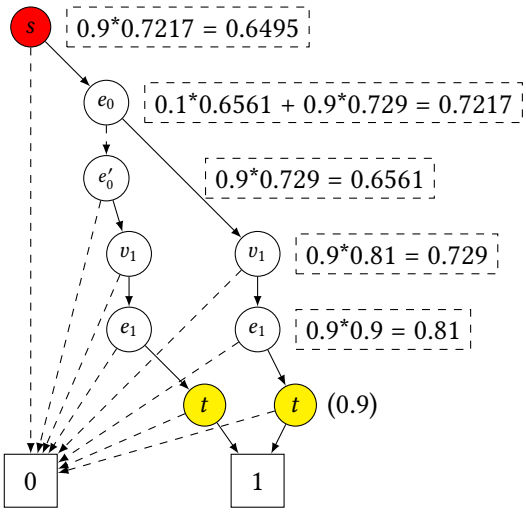
Der Algorithmus "PathConstruct" läuft rekursiv. Dies bedeudet, dass, falls nun die oben beschriebenen Informationen wie Redundanzen und Kanten eines Knotens in der Datenstruktur des Knotens abgespeichert werden würden, entweder die zu entfernenden



BDD-Graph des ursprünglichen Graphen G



BDD-Graph nach Einfügen der Redundanz e'_0 durch die Verknüpfung mit e_0, wodurch die Ausfallsicherheit der Kante e_0 erhöht wird.



BDD-Graph nach Einfügen der Redundanz e'_0 als Mehrfachkante, wodurch ein logisches "oder" ausgeführt wird und ohne "Reduce"-Funktion der BDD-Graph fast doppelt so groß wird.

Abbildung 5.1: Drei BDD-Graphen inklusive der Berechnung der Zuverlässigkeit von s nach t mit der Variablen-Ordnung $s < e_0 < v_1 < e_1 < t$. Der obere BDD stellt den ursprünglichen Netzwerk-Graphen G der Abbildung 5.7 dar. Das linke bzw. rechte BDD zeigt die Auswirkung auf das obere BDD beim Einfügen der Redundanz e'_0 der Kante e_0 als Verknüpfung bzw. Mehrfachkante. Die Zahlen repräsentieren die Zuverlässigkeitswerte der BDD-Knoten auf den entsprechenden Ebenen.

Knoten kopiert oder alle Änderungen abgespeichert werden müssten, um diese nachher wieder hinzufügen zu können. Da allerdings die Graph-Datenstruktur diese Informationen sammelt, kann entweder der Graph kopiert oder die Änderungen, die im Graph vorgenommen wurden, können abgespeichert werden.

Ein redundanter Knoten kann keine Redundanzen besitzen, d. h. falls von einem redundanten Knoten eine Redundanz erstellt werden soll, wird diese immer vom ursprünglichen Knoten aus erstellt.

Da in einem Netzwerk Hierarchien zwischen Komponenten vorkommen können, müssen diese von der Datenstruktur abgebildet werden können. Eine solche Hierarchie kann zwischen einem Computer-System und mehreren VMs, die auf dem Computer-System laufen, bestehen. Um dies abzubilden werden, virtuelle Kanten zwischen den Komponenten erstellt und in der Graph-Datenstruktur abgespeichert. Eine virtuelle Kante e_0 wird dadurch definiert, dass sie nicht ausfallen kann. D. h. ihre Ausfallsicherheit beträgt $p(e_0) = 1$. Dadurch müssen keine zusätzlichen Informationen im Graph gespeichert werden, um die Hierarchie zwischen Komponenten abbilden zu können.

Angenommen, der Knoten v_0 stellt ein Computer-System dar und der Knoten v_1 stellt eine VM, die auf dem Computer-System läuft, dar. Dann würde eine Kante e_0 zwischen v_0 und v_1 erstellt werden mit $p(e_0) = 1$.

5.3 Berechnung der Zuverlässigkeit von k-Terminals

Dieser Abschnitt beschreibt, wie die Gesamtzuverlässigkeit eines Netzwerks berechnet werden kann. Die zugrunde liegende Datenstruktur, die das Netzwerk repräsentiert, ist ein Graph $G = (V, E)$ mit V als Menge der Knoten und E als Menge der Kanten. Der Algorithmus zur Berechnung der Zuverlässigkeit ist für ungerichtete Netzwerke ausgelegt, wobei die Anpassungen für gerichtete minimal sind.

Die Idee des Algorithmus beruht auf der KLY-Methode [2] zur Berechnung der Zuverlässigkeit eines Netzwerks mit k-Terminals, wobei $2 \leq k \leq |V|$ gilt, bei dem sowohl Kanten als auch Knoten ausfallen können. Bei der KLY-Methode wird die Zuverlässigkeit von k-Terminals durch die Verknüpfung von $(k-1)$ Terminal-Paare exakt berechnet, d. h. es wird keine Approximation durchgeführt.

Dabei wird zuerst ein fixer Zielknoten t aus den k-Terminals bestimmt, d. h. $t \in K$, wobei $K \subseteq V$ die Menge der Terminals darstellt und $k = |K|$ gilt. Anschließend werden für alle Quellknoten $s \in K \backslash \{t\}$ die Pfade zu t berechnet und miteinander verknüpft. Für die Berechnung der Pfade kommt die Datenstruktur BDD (siehe Abschnitt 2.2) zum Einsatz. Da diese Datenstruktur boolesche Ausdrücke abbilden kann, können die $(k-1)$ Terminal-Paare mit einem logischen "und" verknüpft werden [2].

Falls eines der k-Terminals eine oder mehrere Redundanzen besitzt, müssen diese berücksichtigt werden. Falls ein Knoten $s \neq t$ eine Redundanz $r \notin K$ (gilt für alle Redundanzen) besitzt, müssen sowohl alle Pfade von s nach t als auch von r nach t bestimmt und anschließend mit einem logischen "oder" verknüpft werden. Dieses BDD kann dann mit den anderen Terminal-Paaren mit einem logischen "und" verknüpft werden.

Sollte der fixe Zielknoten t eine oder mehrere Redundanzen besitzen, müssen für diese Knoten ebenfalls alle möglichen Pfade für alle $s \in K \backslash \{t\}$ berechnet und anschließend die generierten BDDs mit einem logischen "oder" verknüpft werden. Die Berücksichtigung der Redundanzen bei der Berechnung aller Pfade der k-Terminals ist durch die Verknüpfung der $(k-1)$ Terminal-Paare sehr einfach zu realisieren. Aus diesem Grund und der effizienten Berechnung der Zuverlässigkeit wird die KLY-Methode eingesetzt.

Da für jeden Knoten $v \in V$ des Graphen die Möglichkeit bestehen soll, die Zuverlässigkeit separat berechnen zu können, kann v eine Menge von K-Knoten besitzen, in der der Knoten v selbst beinhaltet ist. Die Gesamtzuverlässigkeit des Netzwerks ergibt sich aus der Multiplikation der Zuverlässigkeitswerte der Knoten, die eine Menge von K-Knoten besitzen.

Zur Berechnung der Zuverlässigkeit eines Knotens ist der Algorithmus in drei Teilbereiche unterteilt. Im ersten Bereich werden alle möglichen Pfade unter den k-Terminals im Netzwerk berechnet, ohne dabei die Knoten einzubinden, d. h. nur unter Berücksichtigung der Kanten. Im zweiten Teil werden die Knoten mit eingebunden und im dritten Teil wird schlussendlich die Zuverlässigkeit des generierten BDDs unter Berücksichtigung der Ausfallwahrscheinlichkeiten der jeweiligen Knoten und Kanten berechnet. Diese drei Teile werden im Algorithmus 5.1 "KLYMethod" zusammengefasst. In den folgenden Punkten werden diese Teilbereiche detailliert beschrieben.

Algorithm 5.1 KLYMethod – Berechnung der Zuverlässigkeit von k-Terminals

Require: Graph G, List $kNodes$, Target t ▷ $kNodes = K \backslash \{t\}$

1: Finde eine Variablen-Ordnung der Knoten und Kanten für G durch BFS
2: BDD $bddResult$ = Decompose($G, kNodes, t$)
3: $bddResult$ = Compose($bddResult$)
4: $reliability$ = CalcRel($bddResult$)
5: **return** $reliability$

5.3.1 Berechnung der möglichen Pfade

Dieser Teilabschnitt ist in zwei Punkte gegliedert. Der erste Punkt beschäftigt sich mit der detaillierten Beschreibung des Algorithmus, um alle möglichen Pfade von s nach t zu berechnen. Um dies effizienter zu erreichen, werden redundante Biconnected-Komponenten identifiziert und entfernt. Der zweite Punkt beschreibt dies.

5.3.1.1 Allgemeine Beschreibung des Algorithmus

Für die Berechnung aller möglichen Pfade der k-Terminals wird, wie bereits erwähnt, zuerst ein Terminal als fixer Zielknoten bestimmt, um anschließend eine Heuristik für die Ordnung der Variablen (Knoten und Kanten) vom Zielknoten aus zu bestimmen. Dabei kann die Breitensuche als Heuristik verwendet werden [2]. Diese Variablen-Ordnung ist für die Größe (d. h. Anzahl der Knoten) des zu generierenden OBDDs (Ordered BDD) entscheidend. Je "besser" die Ordnung der Knoten und Kanten des Netzwerks, desto geringer ist die Größe des OBDDs.

Als drastisches Beispiel dient das BDD eines Multiplexers mit n Variablen, das bei optimaler Ordnung weniger als $2n$ Knoten und im Worst Case mehr als $2^{(n+1)}/n$ Knoten aufweist. Da der Komplexitätsaufwand für die optimale Ordnung allerdings NP-vollständig bzw. mit dem effizientesten Algorithmus $O(n^2 \cdot 3^n)$ ist, kommen in der Literatur Heuristiken zum Einsatz [2, 34]. Nun werden alle Pfade zwischen den $(k-1)$ Terminal-Paaren berechnet.

Wie bereits erwähnt werden die generierten BDDs jedes Terminal-Paars mit einem logischen "und" verknüpft. Da jeder dieser k-Knoten eine oder mehrere Redundanzen haben kann, müssen diese Pfade für die Berechnung der Zuverlässigkeit ebenfalls berücksichtigt werden (siehe Anforderung A7). Die einzelnen generierten BDDs der Redundanzen eines k-Knotens zum Zielknoten werden mit dem BDD ihres k-Knotens mit einem logischen "oder" verknüpft.

Zur Generierung des BDDs, das alle möglichen Pfade zwischen einem bestimmten Quellknoten und dem fixen Zielknoten abbildet, kann eine rekursive Funktion angewandt werden. Diese Funktion benötigt dafür den Quellknoten sowie die Abbildung des Netzwerks als Graph in einer bestimmten Datenstruktur.

Von dem Quellknoten s ausgehend werden alle seine benachbarten Kanten im ursprünglichen Graphen G in der Reihenfolge der Tiefensuche besucht. Dies wird dadurch erreicht, dass der ursprüngliche Knoten s sowie alle benachbarten Kanten aus dem Graphen entfernt werden und einer der benachbarten Knoten von s über eine zuvor ausgewählte Kante e nun zu s wird. Diese Methode wird "contraction of G with e": $G * e$ genannt.

Durch diese Graph-Manipulation ergibt sich ein neuer Teilgraph sub_G. Von diesem Teilgraphen können anschließend redundante Biconnected-Komponenten, die weder Quell- noch Zielknoten beinhalten sowie nur mit maximal einer weiteren Komponente verbunden sind, entfernt werden [1].

In jedem rekursiven Aufruf wird ein Teilgraph generiert, der von Yeh et al. [3] als G_{node} bezeichnet wird. Dabei besitzt jeder dieser G_{node} ausgehende Kanten zu seinen "Kindern" G_{node}, sodass jede Kante mit der booleschen Variable der besuchten Kante gekennzeichnet wird. Dieser Rekursions-Graph, der ein gerichteter azyklischer Graph

(DAG) ist, wird als "Edge Expansion Diagram" (EED) bezeichnet. Die Rekursion stoppt, sobald der Knoten s den Zielknoten t erreicht hat.

Bei der ersten Rückkehr der Rekursions-Funktion wird das OBDD für die boolesche Variable B, die die zuletzt besuchte Kante e zwischen den Knoten s und t kennzeichnet, erstellt. Dieses soeben erstellte OBDD wird anschließend in der globalen Hashtabelle mit dem Hashwert des Teilgraphen, bestehend aus den Knoten s und t sowie der Kante e, als Schlüssel gespeichert.

Die Hauptidee der KLY-Methode ist das Erkennen von isomorphen Teilgraphen, wodurch redundante Berechnungen vermieden werden können. Dies erfolgt durch den schrittweisen Aufbau der Hashtabelle durch die berechneten Teilgraphen mit ihren entsprechenden OBDD-Werten. Bei jeder Rekursion wird die Hashtabelle danach überprüft, ob diese bereits einen OBDD-Wert für den gegebenen Teilgraphen besitzt. Ist dies der Fall wird das gespeicherte OBDD zurückgegeben.

Das zurückgegebene OBDD wird anschließend mit den anderen zurückgegebenen OBDDs von den, falls existenten, benachbarten Branches des momentanen G_{node} mit einem logischen "oder" verknüpft, wodurch ein neues OBDD entsteht, das zum "Vater" des G_{node} zurückgegeben wird. Das abschließende OBDD repräsentiert die Funktion X_G für den Graphen G und wird erreicht, sobald alle rekursiven Aufrufe zur Wurzel des EEDs zurückkehren. Laut [3] ist die Funktion folgendermaßen definiert:

$$X_G = \bigvee_{i=1}^{l} B_i \wedge X_{G*e_i}. \tag{5.1}$$

Dabei steht l für die Anzahl der benachbarten Kanten von s und B_i für die boolesche Variable der Kante e_i. Die Anzahl von G_{node} in dem EED entspricht der Anzahl an Einträgen der Hashtabelle, in der jeder Wert eine Referenz auf den BDD-Wurzel-Knoten einer booleschen Funktion abbildet. Da alle Pfade von der Wurzel zu den Terminal-Knoten eines OBDDs disjunkt sind, kann die Wahrscheinlichkeit der booleschen Funktion, die dem Wert der Zuverlässigkeit entspricht, durch den dritten Teilbereich des Algorithmus einfach bestimmt werden [1, 3].

Der Algorithmus 5.2 "PathConstruct" zeigt in Pseudocode wie ein BDD für alle Pfade von einem gegebenen s nach t generiert werden kann.

Laut Yeh et. al [3] kann die Zuverlässigkeit für k-Terminals durch die Verknüpfung der $(k-1)$ Terminal-Paare erzielt werden. Dafür muss die Funktion X_G für alle $(k-1)$ Terminal-Paare mit der booleschen Operation "und" verknüpft werden. Dadurch ergibt sich der Ausdruck

$$X_G = \bigwedge_{s \in K \setminus \{t\}} X_G(s,t), \tag{5.2}$$

Algorithm 5.2 PathConstruct – Erzeugung eines BDDs, das alle möglichen Pfade (nur Kanten) von s nach t abbildet [1]

Require: Graph G, CurrentSource s, Target t
1: BDD $bddNode$, $bddResult$
2: **if** $s == t$ **then**
3: **return** $bddTrue$
4: **if** ($bddResult = hashMap.get(G,s,t)$) is a hit **then**
5: **return** $bddResult$
6: $bddResult = bddFalse$
7: **for all** edges-of-s e **do**
8: $sub_G = G * e$
9: RemoveRedundantBiconnectedComponents($sub_G, source_{sub_G}$)
10: $bddNode = bdd(e) \wedge$ PathConstruct($sub_G, source_{sub_G}, t$)
11: $bddResult = bddResult \vee bddNode$
12: $hashMap.put(G,s,t,bddResult)$
13: **return** $bddResult$

wobei $X_G(s,t)$ die Funktion mit den entsprechenden Terminals s,t ist und $s,t \in K, s \neq t$ gilt [1]. Diese Funktion betrachtet jedoch noch keine Redundanzen von s und t. Daher muss sie folgendermaßen erweitert werden:

$$X_G = \bigvee_{t \in R_t \cup \{t\}} \bigwedge_{k \in K \setminus \{t\}} \bigvee_{s \in R_k \cup \{k\}} X_G(s,t). \tag{5.3}$$

Dabei stellen R_t und R_k die Menge der Redundanzen der jeweiligen Knoten t sowie k dar, wobei für alle Redundanzen $t \in R_t, s \in R_k$ gilt $s,t \notin K$.

Der Algorithmus 5.3 "Decompose" zeigt in Pseudocode wie mit Hilfe des Algorithmus "PathConstruct" ein BDD für die $(k-1)$ Terminal-Paare inkl. ihrer Redundanzen generiert werden kann.

Algorithm 5.3 Decompose – Erzeugung eines BDDs, das alle möglichen Pfade (nur Kanten) der k-Terminals inkl. ihrer Redundanzen abbildet

Require: Graph G, List $kNodes$, Target t ▷ $kNodes = K \setminus \{t\}$
1: BDD $bddResult = bddFalse$
2: **for all** $rt \in \{t\} \cup R_t$ **do** ▷ R_t ist die Menge der Redundanzen des Knotens t
3: BDD $rtResult = bddTrue$
4: **for all** $s \in kNodes$ **do**
5: BDD $sResult$ = PathConstruct(G,s,rt)
6: **for all** $rs \in R_s$ **do** ▷ R_s ist die Menge der Redundanzen von s
7: $sResult = sResult \vee$ PathConstruct(G,rs,rt)
8: $rtResult = rtResult \wedge sResult$
9: $bddResult = bddResult \vee rtResult$
10: **return** $bddResult$

Zum Abschluss dieses Punktes zeigen die folgenden Abbildungen 5.2 und 5.3 exemplarisch wie der Algorithmus für ein kleines Netzwerk mit vier Knoten und fünf Kanten funktioniert. Dabei zeigt die Abbildung 5.2 wie der Graph expandiert sowie alle möglichen Pfade besucht werden. Die dabei generierten BDD-Knoten, durch die die Zuverlässigkeit berechnet werden kann, werden in Abbildung 5.3 gezeigt. Dabei repräsentieren die BDD-Knoten nur die Kanten $e \in E$ des Netzwerk-Graphen. Die Zahlen in der Abbildung stellen die Zuverlässigkeitswerte der jeweiligen BDD-Knoten dar, die mit dem Algorithmus 5.8 "CalcRel" berechnet werden.

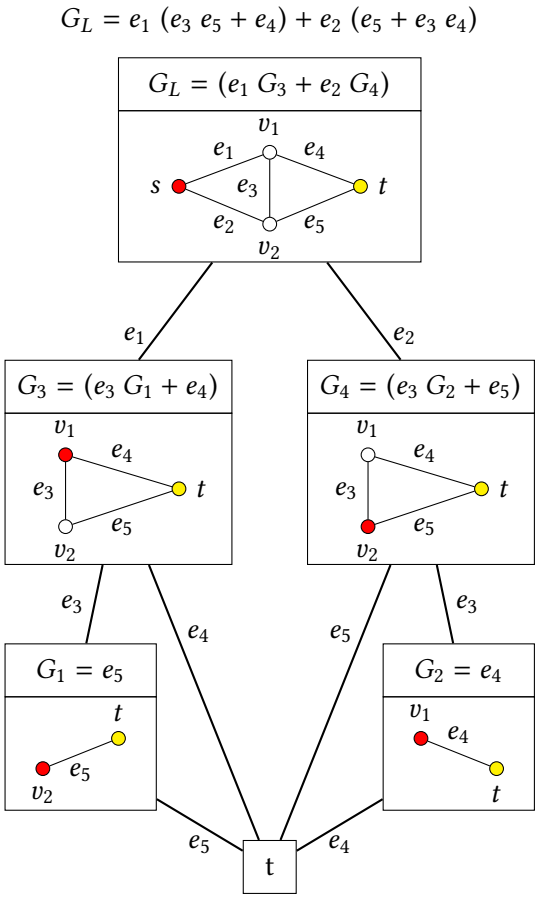


Abbildung 5.2: Darstellung der Funktionsweise der Graph-Expansion von einem Quellknoten s zu einem Zielknoten t [2]

5.3.1.2 Erkennen und Entfernen redundanter Biconnected-Komponenten

Die ursprünglich entwickelte KLY-Methode (vgl. [2, 3, 34]) erkennt und entfernt keine redundanten Biconnected-Komponenten, sondern nur redundante Knoten (Knoten, die weder s noch t sind und nur eine Kante besitzen). Lê [1] hat im Rahmen seiner Dissertation diese Verbesserung entwickelt.

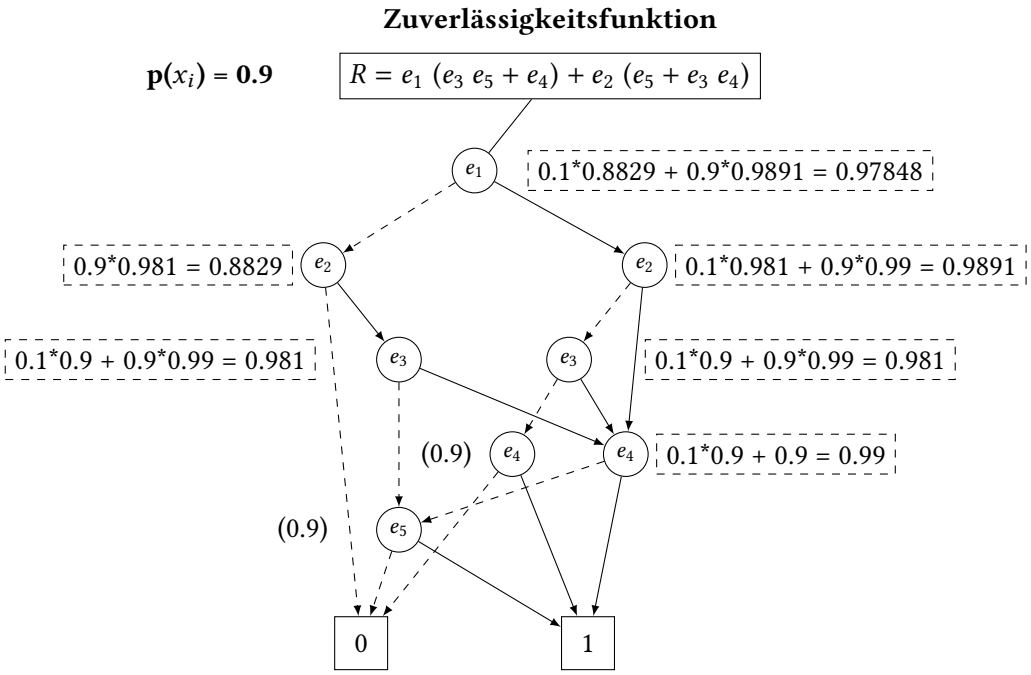


Abbildung 5.3: Zum Netzwerk der Abbildung 5.2 generierter BDD-Graph: Die Zahlen beschreiben die Zuverlässigkeit der jeweiligen BDD-Knoten, die ausschließlich die Kanten $e \in E$ des Netzwerk-Graphen repräsentieren [2].

Wie oben erwähnt werden Biconnected-Komponenten des Netzwerk-Graphen in jedem kompletten Durchlauf des Algorithmus 5.2 "PathConstruct" identifiziert und entfernt, falls sie redundant sind. Damit eine Biconnected-Komponente c als redundant gilt, müssen folgende Bedingungen gelten:

- Sie darf den Quellknoten $s \notin c$ nicht beinhalten.
- Sie darf den Zielknoten $t \notin c$ nicht beinhalten.
- Sie darf mit maximal einer anderen Komponente $\text{degree}(c) \leq 1$ verbunden sein.

Ein "biconnected" ungerichteter Graph ist ein zusammenhängender Graph, der durch das Entfernen eines beliebigen Knotens nicht in mehrere Teile zerfällt. Dies bedeutet, dass ein solcher Graph keinen Gelenkpunkt ("articulation point") besitzt. Ein Gelenkpunkt ist ein Knoten, der beim Entfernen einen zusammenhängenden Graphen in zwei oder mehrere jeweils zusammenhängende Teilgraphen unterteilt [53].

Diese Eigenschaft verhindert, dass ein Graph durch das Entfernen einer beliebigen Kante in Teilgraphen zerfällt, und, somit eine Redundanz besitzt. Es müssen also mindestens zwei Kanten entfernt werden, damit der Graph nicht mehr zusammenhägend ist [1].

Durch das Erkennen und anschließende Entfernen redundanter Biconnected-Komponenten kann der Teilgraph stark reduziert werden, wodurch weniger Zeit für das Cashing

der Teilgraphen benötigt wird. Zusätzlich muss der Algorithmus 5.2 "PathConstruct" weniger oft aufgerufen werden, das bedeutet eine geringere Anzahl an G_{node}, wodurch der Speicheraufwand für die globale Hashtabelle ebenfalls reduziert wird. Allerdings muss der Graph diese Strukturen aufweisen, sonst entsteht ein Mehraufwand ohne Nutzen [1]. Der Mehraufwand ist jedoch linear im Verhältnis zur Größe des Graphen [53].

Hierfür wurden einige Evaluierungen durchgeführt, die in Abschnitt 7.3 beschrieben sind. Abbildung 7.6 zeigt z. B. für Grid-Netzwerke 5x5 bis 5x12 den Unterschied zwischen dem Entfernen redundanter Biconnected-Komponenten oder nur redundanter Knoten (ursprüngliche KLY-Methode). Weitere Evaluierungen zu dieser Thematik können hier [1] gefunden werden.

Umwandlung in einen Biconnected-Komponenten-Baum
Abbildung 5.4 zeigt einen Graph G mit neun Biconnected-Komponenten (eine einzige Kante wird ebenfalls als Biconnected-Komponente betrachtet) und fünf Gelenkpunkten $(a1 - a5)$.

Mit Hopcrofts und Tarjans Algorithmus [53], der in der Funktion $articPointDFS(v)$ abgebildet ist, können diese Komponenten in linearer Zeit durch eine Tiefensuche des Graphen G gefunden werden. Durch die Ausführung von $articPointDFS(s)$ werden für jeden Gelenkpunkt a die entsprechenden Biconnected-Komponenten c als $bcmap$ zurückgegeben (siehe Algorithmus 5.4 "RemoveRedundantBiconnectedComponents"):

$$(a1 : c1, c2), (a2 : c3), (a3 : c4, c5, c6), (a4 : c7), (a5 : c8, c9).$$

Die Zuordnung der einzelnen Komponenten zu den Gelenkpunkten erfolgt aufgrund der Reihenfolge, in der die Gelenkpunkte abgearbeitet werden. Dabei werden Komponenten nur einem Gelenkpunkt zugeordnet und speichern alle entsprechenden Knoten des Graphen G in $c_i, 1 \leq i \leq 9$. Um nun die redundanten Komponenten identifizieren und anschließend entfernen zu können, muss G in einen Biconnected-Komponenten-Baum ($bcTree$), T_G, umgewandelt werden. Dies wird durch den Algorithmus 5.5 "Transform" erzielt.

Der Quellknoten s sowie jede Biconnected-Komponente sind Knoten des $bcTree$. Angefangen beim Quellknoten werden Kanten zu den benachbarten Komponenten gezogen. Dabei wird in jeder dieser Komponenten nach Gelenkpunkten gesucht. Falls solch ein Gelenkpunkt a in der Komponente c gefunden wird, wird vom Knoten c eine Kante zu jeder Komponente in a erstellt. Der Algorithmus 5.6 "DFS" beschreibt die Tiefensuche zur Erstellung des $bcTree$.

Nachdem der ursprüngliche Graph G in den Baum T_G umgewandelt und die Komponente, die den Zielknoten t beinhaltet, identifiziert wurde, können alle redundanten Komponenten – beinhalten weder s noch t und besitzen maximal eine Kante – aus

dem Baum entfernt werden. Die redundanten Komponenten werden in einer Liste abgespeichert, um anschließend alle Knoten, die sich in den Komponenten befinden, aus dem ursprünglichen Graphen G zu entfernen. Abbildung 5.4 zeigt die Umwandlung des Graphen G in den Biconnected-Komponenten-Baum T_G sowie den Graphen G nachdem die sechs redundanten Komponenten entfernt wurden.

In diesem Beispiel stellt der Quellknoten s und der Zielknoten t jeweils selbst einen Gelenkpunkt dar. Dies hat keine Auswirkung auf die Funktionsweise des beschriebenen Algorithmus. Falls s oder t ein Gelenkpunkt im Graphen darstellt, können die Teilgraphen, die keinen Pfad zu t oder s beinhalten, entfernt werden.

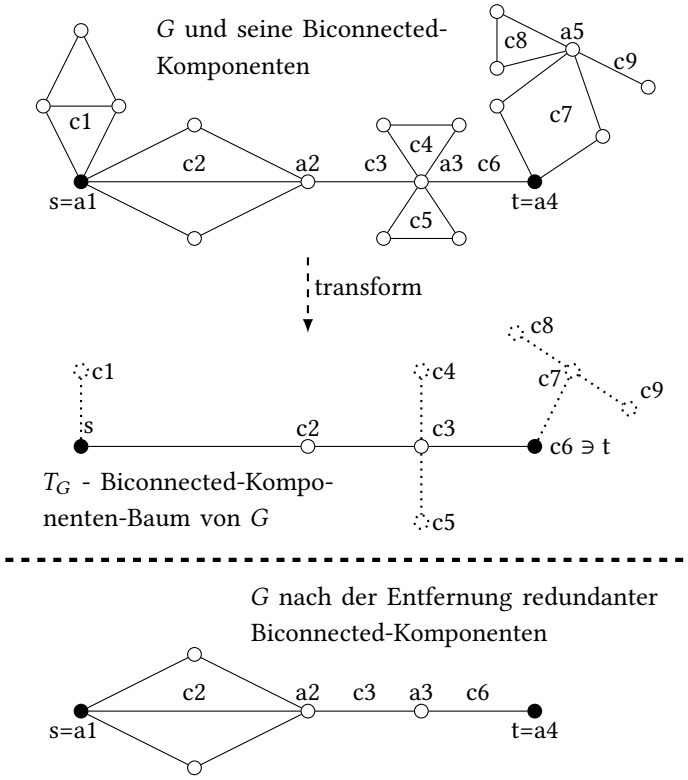


Abbildung 5.4: Entfernung redundanter Biconnected-Komponenten [1]

5.3.2 Hinzufügen der Knoten

Da im ersten Teil des Algorithmus nur die Kanten des Netzwerks in das generierte OBDD aufgenommen wurden, müssen im zweiten Teil zusätzlich die Knoten hinzugefügt werden, um die Anforderung A4 zu erfüllen.

Laut Yeh et al. [2] können mit relativ geringem Aufwand die Knoten mit der Methode CAE ("Composition After Expansion"), die auf der Methode "incident edge substitution"

Algorithm 5.4 RemoveRedundantBiconnectedComponents – Entfernung redundanter Biconnected-Komponenten [1]

Require: Graph G, CurrentSource s, Target t

```
1:  Map bcMap = articPointDFS(s)
2:  if numberOfComponents == 1 then
3:      return                                   ▷ Der Graph G ist "biconnected"
4:  Graph bcTree = Transform(bcMap, s)
5:  Bestimme die Komponente des Graphen bcTree, die den Knoten t enthält
6:  repeat
7:      H := {c ∈ bcTree | c ∉ {s,t} ∧ degree(c) == 1}
8:      for all Component c ∈ H do
9:          bcTree.remove(c)
10:         compsToDelete.add(c)                 ▷ Sammelt c in einer Liste
11: until H = ∅
12: for all Component c ∈ compsToDelete do
13:     for all Node n ∈ c do
14:         G.removeNode(n)
15: return
```

Algorithm 5.5 Transform – Umwandlung der $bcMap$ in den $bcTree$ [1]

Require: Map $bcMap$, CurrentSource s

```
1:  Graph bcTree(numberOfComponents + 1)
2:  Map m = bcMap.get(s)
3:  for all Component c ∈ m do
4:      bcTree.addEdge(s, c)
5:      List ls = m.get(c)                       ▷ Liste aller Knoten in c
6:      for all Node n ∈ ls do
7:          if bcMap.get(n) is not empty then
8:              DFS(bcMap, bcTree, n, c)
9:  return bcTree
```

Algorithm 5.6 DFS – Tiefensuche für die Umwandlung der $bcMap$ in den $bcTree$ [1]

Require: Map $bcMap$, Graph $bcTree$, ArticPoint a, Component $c_{previous}$

```
1:  Map m = bcMap.get(a)
2:  for all Component c ∈ m do
3:      bcTree.addEdge(c_previous, c)
4:      List ls = m.get(a)
5:      for all Node n ∈ ls do
6:          if bcMap.get(n) is not empty then
7:              DFS(bcMap, bcTree, n, c)
8:  return
```

beruht, hinzugefügt werden. Hierfür haben Yeh et al. entsprechende Evaluierungen durchgeführt, die ihrem Paper [2] entnommen werden können.

Für ungerichtete Netzwerke wird die boolesche Variable B_i der Kante $e_i = (n_a^i, n_b^i)$ durch den Ausdruck $n_a^i \wedge B_i \wedge n_b^i$ ersetzt, wobei n_a^i und n_b^i die booleschen Variablen der jeweiligen Endknoten der Kante e_i repräsentieren. Dadurch wird die Funktion X_G aus Gleichung 5.1 in folgender Weise verändert [1]:

$$X_G = \bigvee_{i=1}^{l} (s \wedge B_i \wedge n_b^i) \wedge X_{G*e_i}. \tag{5.4}$$

Um die Knoten nun dem aus dem ersten Teil des Algorithmus generierten BDD hinzuzufügen, wird die Composition-Funktion verwendet, wodurch die "incident edge substitution" rekursiv auf das generierte BDD angewandt wird. Diese Funktion wird als $BDD(G)$ bezeichnet. Mit Rücksicht auf eine vordefinierte Variablen-Ordnung

$$s < e_1 < n_b^1 < n_a^2 < e_2 < n_b^2 < \ldots < n_a^m < e_m < t$$

wird die entsprechende Composition-Funktion als folgender boolescher Ausdruck formuliert:

$$BDD(G) = ((s \wedge B_1 \wedge n_b^1) \wedge BDD(G)_{|B_1=1}) \vee (\overline{(s \wedge B_1 \wedge n_b^1)} \wedge BDD(G)_{|B_1=0}). \tag{5.5}$$

Die zwei Ausdrücke $BDD(G)_{|B_k=1}$ und $BDD(G)_{|B_k=0}$ sind die "high" und "low" BDD-Kinder-Knoten der booleschen Variable B_k [1].

Durch das Verwenden einer globalen Hashtabelle, die einen BDD-Knoten als Schlüssel und den dazu durch die Rekursion errechneten BDD-Knoten als Wert definiert, kann die Effizienz des Algorithmus gesteigert werden (siehe 7.2.1.2). Der Pseudocode für diesen Teilbereich wird in Algorithmus 5.7 "Compose" erfasst. Am Ende der Rekursion wird wieder ein BDD zurückgegeben, der anschließend im dritten Teil des Algorithmus verwendet wird, um die Zuverlässigkeit zu berechnen.

Abbildung 5.5 zeigt den BDD-Graphen für das Netzwerk der Abbildung 5.2 nachdem die Graph-Knoten durch diesen Teil des Algorithmus hinzugefügt wurden. Dabei repräsentieren die BDD-Knoten nun sowohl die Kanten $e \in E$ als auch die Knoten $v \in V$ des Netzwerk-Graphen. Die Zahlen in der Abbildung stellen die Zuverlässigkeitswerte der jeweiligen BDD-Knoten dar, die mit dem Algorithmus 5.8 "CalcRel" berechnet werden.

5.3.3 Berechnung der Zuverlässigkeit

Im letzten Teil des Algorithmus wird mit Hilfe des generierten BDDs und der gegebenen Ausfallwahrscheinlichkeiten der Knoten und Kanten die Zuverlässigkeit berechnet. Dafür wird das BDD rekursiv bis zu seinen Terminal-Knoten evaluiert. Dabei stellt

Algorithm 5.7 Compose – Hinzufügen der Graph-Knoten zu den entsprechenden Graph-Kanten, die das BDD beinhaltet

Require: BDD *bddNode*

1: **if** $(bddNode == bddTrue) \vee (bddNode == bddFalse)$ **then**
2: **return** *bddNode*
3: BDD *bddResult*
4: **if** $(bddResult = hashMap.get(bddNode))$ is a hit **then**
5: **return** *bddResult*
6: Edge e = getEdge($bddNode$) ▷ $e = (v_a, v_b)$
7: BDD $b = bdd(v_a) \wedge bdd(e) \wedge bdd(v_b)$
8: $bddResult = (b \wedge \text{Compose}(bddNode.high)) \vee (\overline{b} \wedge \text{Compose}(bddNode.low))$ ▷ Entspricht der Formel 5.5
9: $hashMap.put(bddNode, bddResult)$
10: **return** *bddResult*

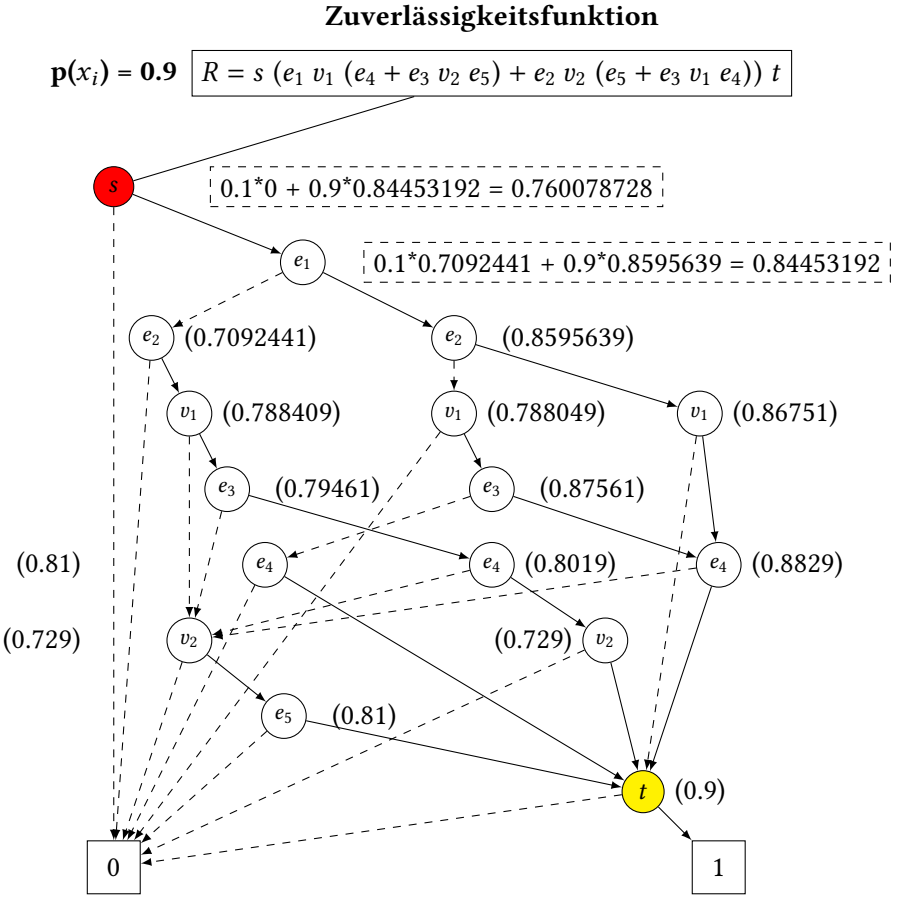


Abbildung 5.5: Zum Netzwerk der Abbildung 5.2 generierter BDD-Graph: Die Zahlen beschreiben die Zuverlässigkeit für die jeweiligen BDD-Knoten, die sowohl die Kanten $e \in E$ als auch die Knoten $v \in V$ des Netzwerk-Graphen repräsentieren [2].

jeder BDD-Knoten, der evaluiert wird, entweder eine Kante oder einen Knoten des Netzwerk-Graphen dar. Die Zuverlässigkeit ergibt sich durch die rekursive Funktion

$$R(BDD(G)) = p(x_i) * R(BDD(G)_{|x_i=1}) + [1 - p(x_i)] * R(BDD(G)_{|x_i=0}), \tag{5.6}$$

wobei x_i ein BDD-Knoten darstellt, der entweder auf eine Kante oder einen Knoten aus dem Netzwerk-Graphen verweist. Die Funktion $p(x_i)$ gibt die Wahrscheinlichkeit an, dass x_i nicht ausfällt. Die zwei Ausdrücke $BDD(G_{|x_i=1})$ und $BDD(G_{|x_i=0})$ bilden die BDD-Kinder-Knoten "high" und "low" des BDD-Knotens x_i. Sobald die Rekursion einen der beiden Terminal-Knoten des BDDs erreicht hat, wird entsprechend für "bddTrue" die Zahl 1 und für "bddFalse" die Zahl 0 zurückgegeben.

Durch das Verwenden einer globalen Hashtabelle, die einen BDD-Knoten als Schlüssel und den dazu errechneten Wahrscheinlichkeitswert als Wert definiert, kann die Effizienz des Algorithmus gesteigert werden. Das Ergebnis der Funktion ist die Zuverlässigkeit von k-Terminals [2]. Dieser Teilbereich wird in Algorithmus 5.8 "CalcRel" dargestellt.

Um die Gesamtzuverlässigkeit des Netzwerks zu bestimmen, muss dieser dreiteilige Algorithmus für alle Knoten, die eine definierte Menge von K-Knoten aufweisen, ausgeführt und die jeweiligen Zuverlässigkeitswerte miteinander multipliziert werden.

Algorithm 5.8 CalcRel – Berechnung der Zuverlässigkeit eines BDDs [1]

Require: BDD *bddNode*

```
1:  ▷ p gibt die Ausfallsicherheit der Komponente an, die von der Variablen bddNode
    repräsentiert wird
2:  if bddNode == bddTrue then
3:      return 1
4:  else if bddNode == bddFalse then
5:      return 0
6:  if (result = hashMap.get(bddNode)) is a hit then
7:      return result
8:  else
9:      result = p · CalcRel(bddNode.high) + (1 − p) · CalcRel(bddNode.low)
10:     hashMap.put(bddNode, result)
11: return result
```

5.4 Vorschlagsystem

In diesem Abschnitt wird das Vorschlagsystem beschrieben. Dafür wurden vier verschiedene Teilsysteme bzw. Algorithmen entwickelt. Der erste Punkt beschreibt, wie kritische Entitäten eines Netzwerks identifiziert werden können. Der zweite und dritte Punkt

nutzen diese Funktionalität, um die Zuverlässigkeit zu erhöhen bzw. zu verringern. Im vierten und letzten Punkt wird ein Verfahren beschrieben, das Vorschläge für die Positionierung einer Redundanz für einen ausgewählten Knoten im Netzwerk unterbreitet. Alle Methoden basieren auf der Berechnung der Zuverlässigkeit des Netzwerks.

5.4.1 Identifizieren kritischer Entitäten

Dieser Abschnitt beschreibt, wie kritische Entitäten – Komponenten und Verbindungen, die eine hohe Relevanz für die Zuverlässigkeit aufweisen – im Netzwerk identifiziert werden können. Die Bestimmung solcher Entitäten wird benötigt, um die Anforderung A8 zu erfüllen.

Diese kritischen Entitäten bilden die Grundlage für das Verbessern der Zuverlässigkeit des Netzwerks. Dabei müssen die kritischen Entitäten des Netzwerks für jeden Graph-Knoten, der eine bestimmte Menge von K-Knoten besitzt, separat berechnet werden, da sich die kritischen Entitäten immer auf die Berechnung der Zuverlässigkeit von k-Terminals beziehen und nicht auf das gesamte Netzwerk. Des Weiteren wird nun beschrieben, wie kritische Entitäten für ein Netzwerk mit k-Terminals bestimmt werden können.

Yeh et al. [2] beschreiben die kritischen Entitäten als "Essential Variables", wobei sowohl Knoten wie auch Kanten als Variablen definiert werden. Weiterhin beschreiben Yeh et al. die kritischste Entität mit folgender Formel:

$$R(G_{|x_i=0}) = \text{Minimum}\left(R\left(G_{|x_j=0}\right)\right), \quad x_i, x_j \in V \cup E, x_i, x_j \neq s, t \tag{5.7}$$

Diese Formel beschreibt mit $x_i, x_j \neq s, t$ das 2-Terminal-Problem. Somit muss dieser Ausdruck für das k-Terminal-Problem folgendermaßen verändert werden: $x_i, x_j \notin K$.

In Worten bedeutet diese Formel, dass eine "Essential Variable" x_i eines Netzwerk-Graphs G sowohl eine Kante wie auch ein Knoten, der nicht in der Menge K vorkommt, sein kann. Diese Variable hat dabei den größten Einfluss auf die Zuverlässigkeit des Netzwerks, d. h. falls diese ausfällt, verringert sich die Zuverlässigkeit um den höchsten Wert. Falls mehrere Variablen bei Ausfall den gleichen Zuverlässigkeitswert ergeben, wird jene Variable, die die höchste Ausfallwahrscheinlichkeit $q(x_i)$ aufweist als die kritischste betrachtet.

Die Bestimmung der Zuverlässigkeit eines Netzwerks unter der Voraussetzung, dass eine Variable x_j ausfällt, kann auf unterschiedliche Weise erfolgen. Yeh et al. [2] beschreiben hierfür zwei verschiedene Verfahren. In der ersten Variante wird im Prinzip noch einmal der komplette Algorithmus zur Berechnung der Zuverlässigkeit ausgeführt. Der einzige Unterschied ist, dass die Variable x_j aus dem Graphen G entfernt wird und somit der Graph $G - x_j$ entsteht. Da diese Methode sehr aufwendig ist, wird noch ein weiteres

Verfahren beschrieben.

In der zweiten Methode kommt die "Composition"-Funktion eines BDDs zum Einsatz. Dabei wird für jede Variable $x_i \notin K$ des Graphen G der BDD-Knoten der Variablen x_i durch den BDD-Knoten $bddFalse$ ersetzt. Dies generiert das BDD $G - x_i$, das anschließend verwendet werden kann, um die neue Zuverlässigkeit des Netzwerks zu berechnen. Dadurch wird vermieden, dass, im Gegesatz zur ersten Variante, nicht der komplette Algorithmus zur Berechnung der Zuverlässigkeit ausgeführt werden muss, sondern nur der Algorithmus 5.8 "CalcRel" für das neu generierte BDD $G - x_i$ [2]. Der Pseudocode für diese Methode wird in Algorithmus 5.9 "CalcEV" gezeigt.

Algorithm 5.9 CalcEV – Bestimmung der kritischen Entitäten [2]

Require: List $Variables$ ▷ $Variables = V \backslash K \cup E$
1: List $EssentialVariables$ ▷ Aufsteigend sortierte Liste
2: **for all** $x_i \in Variables$ **do**
3: $bdd(G - x_i) = bdd(G)_{|x_i=0}$
4: $EssentialVariables.add(\text{CalcRel}(bdd(G - x_i)), x_i)$
5: **return** $EssentialVariables$

Eine weitere Methode, die während der Ausarbeitung dieser Thesis entwickelt wurde, ist für Netzwerke effizienter, bei denen viele BDD-Knoten während der Berechnung der Zuverlässigkeit generiert werden. Die Performance-Unterschiede können im Evaluierungskapitel in Abbildung 7.13 betrachtet werden.

Da für Netzwerke mit vielen generierten BDD-Knoten die "Composition"-Funktion aufwendiger wird, wurde darauf verzichtet und stattdessen der Algorithmus 5.8 "CalcRel" entsprechend angepasst. Bei der Anpassung wird nun überprüft, ob die Entität bzw. Variable, die gerade abgearbeitet wird, der Variablen x_i entspricht. Ist dies der Fall, wird anstelle des Ausdrucks

$$p(x_i) * R\left(BDD(G)_{|x_i=1}\right) + [1 - p(x_i)] * R\left(BDD(G)_{|x_i=0}\right),$$

der Ausdruck: $R(BDD(G)_{|x_i=0})$ ausgeführt, da $p(x_i) = 0$ gilt.

Dies bedeutet, dass für die Bestimmung der Zuverlässigkeit des BDD-Knotens der Variablen x_i nur noch der BDD-Kinder-Knoten "low" verwendet wird. Dadurch wird die Zuverlässigkeit so berechnet, als würde die Variable x_i ausfallen.

Eine Erweiterung, die ebenfalls im Zuge dieser Thesis entwickelt wurde, ist die Betrachtung des Falls, dass einer der k-Knoten eine oder mehrere Redundanzen besitzen kann. Trifft dies zu, wird der Ausfall von mehreren Variablen, entsprechend der maximalen Anzahl an Redundanzen von einem der k-Knoten inkl. des k-Knotens, gleichzeitig berechnet. Dadurch verändert sich der Komplexitätsaufwand zum Identifizieren der kritischen Entitäten von linear zu polynomiell, d. h. $O(|VAR|^r)$, wobei $VAR = V \backslash K \cup E$

die Menge der Variablen ist und

$$r = \text{Maximum}\,(|R_{k_i}|) + 1, \quad k_i \in K. \tag{5.8}$$

R_{k_i} bildet die Menge der Redundanzen des Knotens k_i.

Warum diese Betrachtung wichtig ist, wird an folgendem Beispiel erläutert:

Abbildung 5.6 zeigt das Beispielnetzwerk, das als Graph mit neun Knoten und zwölf Kanten dargestellt wird. Für den Knoten v_0 bzw. s sollen die Zuverlässigkeit sowie die kritischen Entitäten bestimmt werden. Dabei besitzt s eine Redundanz s' bzw. den Knoten v_8 sowie zwei k-Knoten k_1 und k_2, die mit s zusammen die Menge K bilden. Die Zuverlässigkeit wird nun durch die möglichen Wege von s nach k_1 und k_2 oder s' nach k_1 und k_2 berechnet. Anschließend können die kritischen Entitäten bestimmt werden.

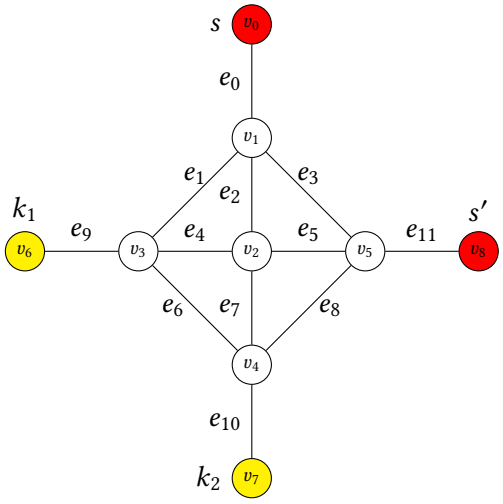


Abbildung 5.6: Netzwerk mit drei k-Knoten sowie einer Redundanz s' des Knotens s

Angenommen, die Kanten e_0 und e_{11} weisen jeweils eine Ausfallwahrscheinlichkeit von $q(e_i) = 0.6$ und die Kanten e_9 sowie e_{10} weisen jeweils eine Ausfallwahrscheinlichkeit von $q(e_j) = 0.1$ auf. Hier werden nun die beiden Fälle – Betrachtung des Ausfalls einer Variable oder zweier Variablen gleichzeitig – unterschieden.

1. **Betrachtung des Ausfalls einer Variablen**
   In diesem Fall sind die kritischsten Entitäten die Kanten e_9 und e_{10} sowie die Knoten v_3 und v_4, da, falls eine dieser Entitäten ausfällt, der jeweilige k-Knoten k_1 oder k_2 nicht mehr erreichbar ist, wodurch die Zuverlässigkeit des Netzwerks den Wert 0 hat.

   Falls eine der beiden Kanten e_0 oder e_{11} ausfallen würde, wäre der Knoten s oder s' immer noch erreichbar, wodurch die Zuverlässigkeit größer 0 ist.

2. **Betrachtung des Ausfalls einer oder mehrerer Variablen**
   Im zweiten Fall wird der Ausfall, beginnend bei jeder einzelnen bis zu r (siehe Gleichung 5.8) Variablen gleichzeitig, betrachtet. In diesem Beispiel sind dies sowohl jede einzelne Variable wie auch immer zwei Variablen gleichzeitig, da der Knoten s eine Redundanz besitzt.

   Wenn nun der Ausfall der zwei Kanten e_0 und e_{11} gleichzeitig betrachtet wird, hat die Zuverlässigkeit, wie bei den im ersten Fall beschriebenen Entitäten, den Wert 0. Da allerdings die Wahrscheinlichkeit, dass beide Kanten gleichzeitg ausfallen, mit $q(e_0 \wedge e_{11}) = 0.36$ größer ist als die Wahrscheinlichkeit, dass entweder die Kante e_9 oder e_{10} mit $q(e_9 \vee e_{10}) = 0.19$ ausfällt, wird in diesem Fall das Kanten-Paar e_0 und e_{11} als kritischste Komponente angeführt.

Die beiden oben beschriebenen Algorithmen können für diese Erweiterung leicht adaptiert werden. Bei dem Algorithmus von Yeh et al. muss die "Composition"-Methode für alle Entitäten, die gleichzeitig ausfallen sollen, ausgeführt und anschließend die Zuverlässigkeit des geänderten BDDs berechnet werden. Beim selbstentwickelten Verfahren muss anstelle der Überprüfung einer Variablen eine Liste von Variablen überprüft werden. Bei einer Übereinstimmung muss immer der BDD-Kinder-Knoten "low" der jeweiligen Variable ausgeführt werden.

5.4.2 Erhöhen der Zuverlässigkeit

Dieser Punkt beschreibt, wie die Zuverlässigkeit eines Netzwerks verbessert bzw. erhöht werden kann, unter der Angabe eines Werts für die gewünschte Zuverlässigkeit. Dabei nutzt das Verfahren die in Punkt 5.4.1 beschriebenen kritischen Entitäten sowie das Einfügen redundanter Entitäten in das Netzwerk. Da dies ein Optimierungsproblem darstellt, muss die Zuverlässigkeit schrittweise erhöht und sich so dem gewünschten Wert angenähert werden. Im Folgenden wird nun ein Verfahren beschrieben, das während der Ausarbeitung dieser Thesis entwickelt wurde.

Bei dieser Methode werden die Ausfallsicherheiten der einzelnen Entitäten, d. h. $p(x_i)$ nicht verändert, sondern redundante Entitäten in das Netzwerk eingefügt. Im ersten Schritt werden die Ausfallsicherheiten der k-Terminals erhöht, indem jedes Terminal, dessen Ausfallsicherheit nicht über der gewünschten Zuverlässigkeit liegt (für $p(k) <= rel$, wobei rel die gewünschte Zuverlässigkeit darstellt und $k \in K$ ist), eine Redundanz erhält. Dies ist notwendig, da für jedes k-Terminal, das keine Redundanz besitzt, die Zuverlässigkeit des Netzwerks maximal Minimum$(p(k))$ sein kann.

Zur Unterscheidung der Ausfallsicherheit einer Entität, die mit der Funktion $p(x_i)$ angegeben wird, und der Ausfallsicherheit einer Entität inklusive ihrer Redundanzen,

wird eine neue Funktion $Rel(x_i)$ eingeführt:

$$Rel(x_i) = 1 - \left[q(x_i) \prod_{r \in R_{x_i}} q(r) \right], \tag{5.9}$$

wobei $q(x_i) = 1 - p(x_i)$, $Rel(x_i) = [0,1]$ $\forall x_i \in V \cup E$ ist und R_{x_i} die Menge aller Redundanzen der Entität x_i beschreibt.

Um die gewünschte Zuverlässigkeit zu erreichen, muss weiterhin für alle $k \in K$ die Funktion $Rel(k) > rel$ gelten. Zusätzlich muss die Verknüpfung aller k-Terminals ebenfalls größer als die gewünschte Zuverlässigkeit sein. Dies bedeutet, dass der folgende Ausdruck ebenfalls gelten muss:

$$\prod_{k \in K} Rel(k) > rel. \tag{5.10}$$

Es müssen solange Redundanzen der k-Terminals erzeugt werden, bis diese Ausdrücke gelten. Falls die Zuverlässigkeit nach Einführen von Redundanzen für die k-Terminals bereits ausreichend ist, kann der Algorithmus beendet werden. Ist dies nicht der Fall, wird mit dem zweiten Schritt fortgefahren.

Im zweiten Teil des Verfahrens werden Redundanzen für die kritischen Entitäten eingefügt. Dafür wird ein Näherungsverfahren angewandt, bei dem solange Redundanzen erzeugt werden, bis die gewünschte Zuverlässigkeit erreicht wird. Hierfür wird eine Liste kritischer Entitäten durchlaufen, die absteigend nach der Kritikalität sortiert ist.

In jedem Durchlauf wird eine Redundanz erzeugt und überprüft, ob das Netzwerk nun die gewünschte Zuverlässigkeit rel aufweist. Hierfür muss zur exakten Bestimmung der Zuverlässigkeit der Algorithmus 5.1 "KLYMethod" des Abschnitts 5.3 ausgeführt werden.

Da dies für größere Netzwerke relativ aufwendig ist, wird eine Methode verwendet, deren Ergebnis für die Zuverlässigkeit mindestens dem Wert der exakten Berechnung gleicht. Mit dieser Methode kann eine schrittweise Annäherung an den gewünschten Wert rel erzielt werden. Falls dieser Wert erreicht wurde, wird mit dem Algorithmus 5.1 "KLYMethod" die exakte Zuverlässigkeit des Netzwerks bestimmt. Sollte dieser Wert dem gewünschten genügen, kann der Algorithmus beendet werden. Ist dies nicht der Fall, werden mit der schrittweisen Annäherung wieder weitere Redundanzen eingefügt und die gerade beschriebenen Punkte wiederholt, bis die exakte Zuverlässigkeit mindestens der gewünschten entspricht. Der Algorithmus 5.10 "ImproveReliability" zeigt den Pseudocode hierfür.

Beim Erzeugen einer Redundanz für eine Entität wird unterschieden, ob diese Entität ein Knoten oder eine Kante ist. Bei einer Kante wird eine zusätzliche Kante mit der gleichen Ausfallsicherheit erzeugt. Für die Berechnung der Zuverlässigkeit kann die

Algorithm 5.10 ImproveReliability – Erhöhung der Zuverlässigkeit

Require: Graph G, BDD $bddResult$, Reliability $currentRel$, Reliability $wantedRel$

1: **if** $currentRel \geq wantedRel$ **then**
2: **return** $currentRel$
3: **for all** $k \in K$ **do**
4: **while** Rel(k) $\leq wantedRel$ **do**
5: CreateRedundancy(k)
6: **while** $\prod_{k \in K} Rel(k) \leq wantedRel$ **do**
7: CreateRedundancy(k)
8: $currentRel$ = Reliability($bddResult$)
9: List EV = CalcEV($V \backslash K \cup E$) ▷ EV ist die Liste der kritischsten Entitäten
10: **for all** $x_i \in EV \wedge currentRel \leq wantedRel$ **do**
11: CreateRedundancy(x_i)
12: $currentRel$ = Reliability($bddResult$)
13: $currentRel$ = KLYMethod($G, K \backslash \{t\}, t$) ▷ Exakte Berechnung der Zuverlässigkeit
14: **if** $currentRel < wantedRel$ **then**
15: **go to** 9
16: **return** $currentRel$

Funktion $Rel(e)$, wobei $e \in E$ gilt, angewandt werden. Bei der Erzeugung einer Redundanz für einen Knoten $v \in V$ wird sowohl ein zusätzlicher Knoten mit der gleichen Ausfallsicherheit erzeugt, als auch eine Redundanz für jede Kante $e \in E_v$ des Knotens v erzeugt.

Da sich der Graph G dadurch entsprechend verändert, beschreibt die Funktion $Rel(v)$ nicht exakt die neue Zuverlässigkeit. Diese Funktion kann allerdings als Annäherung verwendet werden, da der Wert der Zuverlässigkeit durch $Rel(v)$ mindestens genau so groß ist wie der exakte Wert. Um diese Annäherung zu bestimmen, wird der Algorithmus 5.8 "CalcRel" etwas angepasst. Diese Anpassung betrifft nur die Zeile 9 des Algorithmus. Anstatt $p(x_i)$ muss die Funktion $Rel(x_i)$ ausgeführt werden.

Somit muss für die Annährung nur wiederholt der Algorithmus zur Berechnung der Zuverlässigkeit eines bereits generierten BDD-Graphen ausgeführt werden und nicht der komplette Algorithmus 5.1 "KLYMethod". Dieser muss erst zur Berechnung der exakten Zuverlässigkeit durchgeführt werden. Durch diese Methode kann die Laufzeit reduziert werden, wie Abbildung 7.17 im Evaluierungs-Kapitel anhand eines Beispiels zeigt.

Das folgende Beispiel veranschaulicht den Unterschied zwischen dem Näherungsverfahren und der exakten Zuverlässigkeits-Berechnung.

Abbildung 5.7 zeigt drei Graphen, wobei der urspüngliche Graph aus drei Knoten und zwei Kanten besteht. Es wird nun angenommen, dass für den Knoten v_1 eine Redundanz

v_1' eingeführt werden soll, um die Zuverlässigkeit des Netzwerks zu erhöhen. Der linke Graph in Abbildung 5.7 zeigt die Darstellung des Netzwerks durch das Näherungsverfahren. Der Knoten v_1 und seine Redundanz v_1' bilden zusammen einen Knoten, dessen Zuverlässigkeit anstatt $p(v_1)$ der Zuverlässigkeit $p(v_1 \vee v_1')$ entspricht, damit mindestens einer der beiden Knoten funktioniert. Für die Kanten e_0 und e_1 von v_1 gilt dasselbe.

Der rechte Graph in Abbildung 5.7 zeigt die tatsächliche Darstellung des Graphen G nach Einfügen der Redundanz v_1' des Knotens v_1. Diese Darstellung des Graphen wird genutzt, um die exakte Zuverlässigkeit zu bestimmen. In Abbildung 5.7 werden die unterschiedlichen Zuverlässigkeitsfunktionen für die drei dargestellten Graphen aufgezeigt. Angenommen $p(x_i) = 0.9$, dann ergibt sich im Näherungsverfahren die Zuverlässigkeit:

$$\begin{aligned} R &= p(s \wedge (e_0 \vee e_0') \wedge (v_1 \vee v_1') \wedge (e_1 \vee e_1') \wedge t) \\ &= 0.9 * 0.99 * 0.99 * 0.99 * 0.9 \\ &= 0.7859 \end{aligned}$$

Und aus der exakten Berechnung resultiert die Zuverlässigkeit:

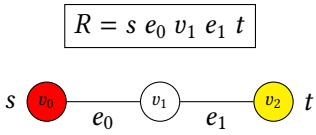
$$\begin{aligned} R &= p\left(s \wedge \left[(e_0 \wedge v_1 \wedge e_1) \vee (e_0' \wedge v_1' \wedge e_1')\right] \wedge t\right) \\ &= 0.9 * \left[1 - (1 - 0.9 * 0.9 * 0.9)^2\right] * 0.9 \\ &= 0.7505. \end{aligned}$$

Je höher der Grad der Dichte (Verhältnis tatsächlicher Kanten zu potenziell möglichen Kanten ohne Mehrfachkanten und ohne Selbstschleifen) des Graphen, desto genauer ist das Näherungsverfahren. Um dies zu zeigen, wurde eine Evaluierung durchgeführt (siehe Abbildung 7.18).

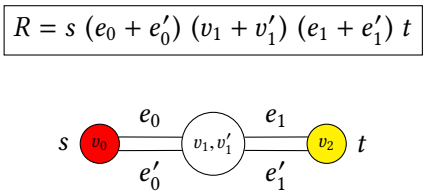
Die Zuverlässigkeit des Netzwerks in Abbildung 5.8 entspricht $R = 0.45879$, wobei $p(x) = 0.9$ gilt. Angenommen, die Zuverlässigkeit soll auf 0.5 gesteigert werden. Der Algorithmus würde eine Redundanz für den Knoten v_2 einfügen, da dieser der kritischste Knoten ist. Durch das Näherungsverfahren ergibt sich ein Wert von $R = 0.514229$ und durch die exakte Berechnung der Zuverlässigkeit ein Wert von $R = 0.508108$. In diesem Beispiel ist die Annäherung wesentlich genauer, da der Graph, der das Netzwerk repräsentiert, eine höhere Dichte aufweist.

5.4.3 Verringern der Zuverlässigkeit

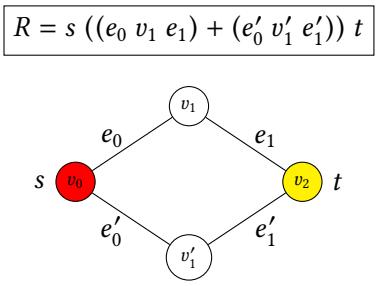
In diesem Punkt wird ein selbstentwickeltes Verfahren beschrieben, mit dem die Zuverlässigkeit eines Knotens, der eine Menge K-Knoten besitzt, auf einen gewünschten Wert reduziert werden kann. Dies kann für Netzwerke sinnvoll sein, die stark vernetzt sind und eine zu hohe Zuverlässigkeit aufweisen, um Kosten zu sparen. Des Weiteren kön-



Ursprünglicher Graph G



G nach Einfügen einer Redundanz v_1' von v_1 durch das Näherungsverfahren



G nach Einfügen der Redundanz v_1' von v_1

Abbildung 5.7: Der oberste Graph stellt den ursprünglichen Graphen G dar. Der linke Graph beschreibt G nach dem Einfügen der Redundanz v_1' durch das Näherungsverfahren. Der rechte Graph zeigt die tatsächliche Darstellung von G nach Einfügen von v_1'. R beschreibt die Zuverlässigkeitsfunktion der verschiedenen Graphen.

nen dadurch Knoten und Kanten, die für die Zuverlässigkeit des ausgewählten Knotens nicht relevant sind, entfernt werden.

Die zugrunde liegende Kenngröße, wie relevant eine Entität ist, wird wieder durch den Algorithmus zum "Identifizieren kritischer Entitäten" aus Punkt 5.4.1 bestimmt. Diesesmal werden jedoch die am wenigsten kritischen Entitäten berechnet, um anschließend entfernt zu werden. Der Algorithmus arbeitet folgendermaßen:

Zuerst wird die Zuverlässigkeit für den angegebenen Knoten mit der Menge K-Knoten errechnet, anschließend werden die unkritischsten Entitäten bestimmt. Dafür wird die Zuverlässigkeit für den Ausfall jeder Entität exkl. der Menge K berechnet und der Wert mit der jeweils entsprechenden Entität in eine Liste, die aufsteigend sortiert wird, gespeichert. Da die Betrachtung des Ausfalls von mehreren Entitäten gleichzeitig für das Verringern der Zuverlässigkeit keinen Mehrwert aufweist, wird darauf verzichtet, um den Rechenaufwand zu reduzieren.

Als nächstes werden solange Entitäten aus dem Graphen G sowie dem dazugehörigen BDD-Knoten entfernt, bis sich der Zuverlässigkeitswert verringert. Dabei wird überprüft, ob der Graph nach der Entfernung einer Entität noch zusammenhängend ist. Ist dies nicht der Fall, wird der Schritt rückgängig gemacht. Um die neue Zuverlässigkeit zu berechnen, muss nur der Algorithmus 5.8 "CalcRel" für den angepassten BDD-Knoten ausgeführt werden.

Sobald sich der Zuverlässigkeitswert verringert, werden die unkritischsten Entitäten neu bestimmt, da das Entfernen einer Entität, die den Zuverlässigkeitswert reduziert, Einfluss auf die Zuverlässigkeitswerte beim Ausfall weiterer Entitäten haben kann.

Anschließend werden wieder Entitäten entfernt, bis sich die Zuverlässigkeit verringert. Diese Schritte werden solange wiederholt, bis das weitere Entfernen einer beliebigen Entität die Zuverlässigkeit unter den gewünschten Wert reduzieren würde. Der Algorithmus 5.11 "ReduceReliability" zeigt das eben Beschriebene in Pseudocode.

Der Beispielgraph aus Abbildung 5.8 veranschaulicht dies. Für die Bestimmung der Zuverlässigkeit von s sind alle Wege von s nach k_1, k_2 und k_3, die zu viert die Menge der K-Knoten bilden, relevant. Daher beeinflussen die Knoten v_{13} und v_{15} sowie die Kanten - e_{16} und e_{18} - abgehend von diesen Knoten die Zuverlässigkeit für s nicht. Somit sind diese vier Entitäten die unkritischsten und können ohne Reduzierung der Zuverlässigkeit entfernt werden.

Angenommen, diese vier Entitäten wurden entfernt und alle Knoten sowie Kanten des Graphen $G = (V, E)$ haben die gleiche Ausfallwahrscheinlichkeit $q(x_j) = 0.1$, wobei $x_j \in V \cup E$ gilt, dann beträgt die Zuverlässigkeit für den Knoten s 0.458793. Die Tabelle 5.1 zeigt nun die sechs unkritischsten Entitäten absteigend sortiert inkl. der Zuverlässigkeit bei Ausfall der jeweiligen Entitäten. Wie der Tabelle entnommen werden kann, wird die Zuverlässigkeit durch das Entfernen der Entitäten v_0, e_0 und e_1 in jeweils gleichem

Algorithm 5.11 ReduceReliability – Reduktion der Zuverlässigkeit

Require: Graph G, BDD $bddResult$, Reliability $currentRel$, Reliability $wantedRel$

```
1:  if currentRel ≤ wantedRel then
2:      return currentRel
3:  List Variables = V\K ∪ E
4:  List EssentialVariables = CalcEV(Variables)
5:  for all x_i ∈ EssentialVariables.reverse do
6:      if x_i.rel ≥ wantedRel then
7:          Graph TempG = G
8:          G.remove(x_i)
9:          if G is connected then
10:             tempRel = currentRel
11:             Variables.remove(x_i)
12:             bddResult.remove(x_i)
13:             currentRel = CalcRel(bddResult)
14:             if currentRel < tempRel then
15:                 go to 4
16:         else
17:             G = TempG
18: return currentRel
```

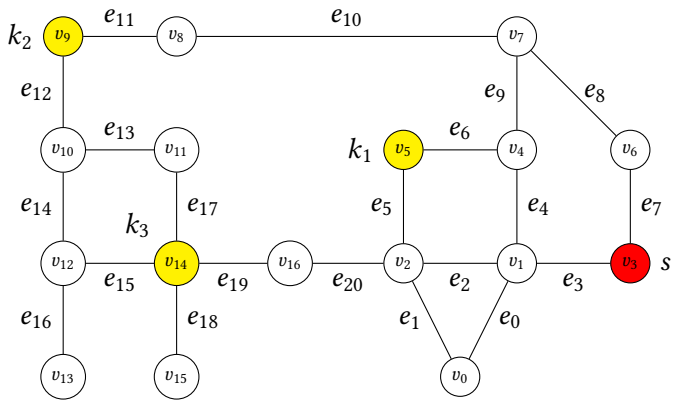


Abbildung 5.8: Graph, bestehend aus 17 Knoten und 21 Kanten mit einem Knoten s (rot markiert) und drei weiteren Knoten k_1, k_2, k_3 (gelb markiert), die zusammen die Menge der K-Knoten für s bilden.

Ausmaß am geringsten reduziert.

Entität	Zuverlässigkeit bei Ausfall
v_0	0.454206402505721
e_0	0.454206402505721
e_1	0.454206402505721
e_2	**0.443446575040151**
e_4	0.438873321251647
e_{15}	0.416981333112781

Tabelle 5.1: Die unkritischsten Entitäten mit den jeweiligen Zuverlässigkeitswerten bei einem Ausfall für den Knoten s des Graphen G (siehe Abbildung 5.8), nachdem die Entitäten v_{13}, v_{15}, e_{16} und e_{18} entfernt wurden.

Des Weiteren ist laut Tabelle die Kante e_2 die viert unkritischste Entität und würde bei Ausfall die Zuverlässigkeit nur minimal verringern. Falls allerdings eine der drei zuvor gennanten Entitäten aus dem Graphen entfernt wird, würde dies die Kante e_2 entsprechend beeinflussen, wodurch diese kritischer wäre. Dies veranschaulicht die Tabelle 5.2, die die unkritischsten Entitäten mit den jeweiligen Zuverlässigkeitswerten bei einem Ausfall zeigt, nachdem die Entitäten v_0, e_0 und e_1 aus dem Graphen entfernt wurden.

Entität	Zuverlässigkeit bei Ausfall
e_4	0.427844029025826
e_{15}	0.412385037965669
v_{12}	0.412385037965669
e_{14}	0.412385037965669
e_{13}	0.412385037965669
e_{17}	0.412385037965669
v_{11}	0.412385037965669
e_2	**0.397575731634299**

Tabelle 5.2: Die unkritischsten Entitäten mit den jeweiligen Zuverlässigkeitswerten bei einem Ausfall für den Knoten s des Graphen G (siehe Abbildung 5.8), nachdem die Entitäten v_{13}, v_{15}, e_{16} und e_{18} sowie v_0, e_0 und e_1 entfernt wurden.

Aus diesem Grund müssen die kritischen Entitäten jedesmal neu bestimmt werden, sobald eine Entität entfernt wurde, die die Zuverlässigkeit reduziert. Dadurch werden immer die Knoten und Kanten mit der geringsten Relevanz hinsichtlich der Zuverlässigkeit entfernt, bis die gewünschte Zuverlässigkeit erreicht wird.

5.4.4 Vorschläge zur Positionierung von Redundanzen

In diesem Punkt wird ein Verfahren beschrieben, das Vorschläge für die Positionierung einer Redundanz für einen ausgewählten Knoten im Netzwerk unterbreitet. Daher muss angegeben werden, für welchen der k-Knoten von einem Knoten, der eine Menge K besitzt, eine Redundanz erzeugt werden soll. Zusätzlich kann angegeben werden wie hoch die Zuverlässigkeit durch das Einfügen der Redundanz mindestens sein soll, damit die Vorschläge in die zurückgebende List aufgenommen werden. Falls dies nicht angegeben wird, nimmt der Algorithmus die vorhandene Zuverlässigkeit, um alle Erhöhungen als Vorschläge zu unterbreiten.

Im ersten Schritt des Verfahrens wird eine Redundanz r für den angegebenen Knoten v erzeugt, wobei für r die gleiche Ausfallsicherheit wie für v mit $p(r) = p(v)$ angenommen wird. Zusätzlich erhält die Redundanz eine Kante, um den Knoten mit dem Graphen verbinden zu können. Damit nicht jede Position bzw. jeder Knoten im Graph überprüft werden muss und somit der Rechenaufwand reduziert werden kann, wird eine Vorauswahl an möglichen Positionen bzw. Knoten getroffen. Diese Vorauswahl wird allerdings nur durchgeführt, falls die Menge K mehr als zwei Knoten beinhaltet ($|K| > 2$). Ist dies nicht der Fall, werden alle Knoten des Graphen als mögliche Position für die Redundanz überprüft.

Der Algorithmus für die Vorauswahl funktioniert folgendermaßen:
Als Eingabe wird der Graph G sowie eine Liste $kNodes$, die die Menge K ohne den Knoten v beinhaltet, benötigt. Als erstes wird die Liste bis zum vorletzten Element durchgegangen und für jedes Element bzw. jeden Knoten k_i inkl. seiner Redundanzen der Dijkstra-Algorithmus ausgeführt, um die kürzesten Pfade zu dem jeweiligen Knoten zu berechnen.

Als Kenngröße für die Distanz beim Dijkstra-Algorithmus wird die Ausfallsicherheit $p(x_i)$, $x_i \in V \cup E$ der einzelnen Entitäten (Knoten und Kanten) verwendet. D. h. der kürzeste Pfad wird als der Pfad definiert, der die höchste Ausfallsicherheit aufweist. Angenommen die Menge P_{n_i,n_j} für $n_i, n_j \in V$ und $n_i \neq n_j$ beinhaltet alle Pfade des Knotens n_i zum Knoten n_j, dann kann der kürzeste Pfad von n_i nach n_j mit der Menge

$$SP_{n_i,n_j} = \left\{ y \in P_{n_i,n_j} \mid \text{Maximum} \left(\prod_{x \in y} p(x) \right) \right\} \tag{5.11}$$

definiert werden.

Weiterhin wird in einem zweiten Schleifendurchgang die Liste $kNodes$, ausgehend von der Stelle nach dem Knoten k_i bis zum Ende, durchgegangen. Dabei werden für jeden Knoten $k_j \neq k_i$ inkl. seiner Redundanzen alle Knoten, die auf dem kürzesten Pfad zu k_i liegen, einer Liste hinzugefügt, die keine Duplikate beinhaltet.

Mit diesem Algorithmus 5.12 "ComputePossibleNodes" werden somit alle Knoten, die auf den kürzesten Pfaden der Knoten k_i der Liste $kNodes$ untereinander liegen, als mögliche Positionen für die Verbindung der Redundanz r mit dem Graphen bestimmt. Dadurch wird garantiert, dass die Positionen, die die Zuverlässigkeit am meisten steigern können, betrachtet werden.

Die Komplexität für den Algorithmus ist abhängig von der Implementierung des Dijkstra-Algorithmus. Mit der weitverbreiteten Implementierung von Dijkstra mit einer Priority-Queue erhält man eine Komplexität von $O(|E| + |V| \cdot log|V|)$ [54, p. 87, ch. 3]. Da der Algorithmus zur Bestimmung der möglichen Positionen den Algorithmus von Dijkstra genau $|K| - 2$ mal inkl. den jeweiligen Redundanzen R_k der k-Knoten für $k \in K$ und $|K| > 2$ aufruft, ergibt sich ein Komplexitätsaufwand von

$$O\left((|E| + |V| \cdot log|V|) \cdot (|K| + red)\right), \quad red = \sum_{k \in K} |R_k|.$$

Algorithm 5.12 ComputePossibleNodes – Bestimmung möglicher Positionen für Redundanzen

Require: Graph G, List $kNodes$

```
1:  Set PossibleNodes                                  ▷ Beinhaltet keine Duplikate
2:  for i = 0 → |kNodes| − 1 do
3:      for all s ∈ {kNodes_i} ∪ R_kNodes_i do
4:                  ▷ R_kNodes_i ist die Menge der Redundanzen des Knotens kNodes_i
5:          Dijkstra.computePaths(G, s)          ▷ Berechne die kürzesten Pfade zu s
6:          for j = i + 1 → |kNodes| do
7:              for all t ∈ {kNodes_j} ∪ R_kNodes_j do
8:                  PossibleNodes.add(Dijkstra.getShortestPath(t))
9:                          ▷ Füge die Knoten des kürzesten Pfads von s nach t hinzu
10: return PossibleNodes
```

Der Beispielgraph in Abbildung 5.8 veranschaulicht dies. Angenommen, für den k-Knoten k_1 soll eine Redundanz erstellt werden und alle Entitäten des Graphen haben eine Ausfallwahrscheinlichkeit von $q(x_i) = 0.1$ mit $x_i \in V \cup E$. Die Liste $kNodes$ beinhaltet somit (in dieser Reihenfolge) die Knoten s, k_2 und k_3 ($K \backslash \{k_1\}$). Ausgehend von s und allen Redundanzen von s (besitzt keine) werden mit dem Dijkstra-Algorithmus die kürzesten Pfade zu allen Knoten berechnet.

Nun werden für k_2 und k_3 inkl. ihrer Redundanzen (besitzen keine) die Knoten auf dem kürzesten Pfad zu s in eine Liste, die keine Duplikate beinhaltet, gespeichert. Dies sind für s nach k_2 die Knoten s, v_6, v_7, v_8 und k_2 sowie für s nach k_3 die Knoten s, v_1, v_2, v_{16} und k_3. Anschließend wird dies für den nächsten Knoten der Liste $kNodes$ wiederholt, welcher in diesem Beispiel k_2 ist.

Nun werden die kürzesten Pfade des Knotens k_2 und seiner Redundanzen mit Hilfe des

Dijkstra-Algorithmus berechnet. Da nur noch der Knoten k_3 übrig ist, müssen die Knoten des kürzesten Pfads von k_2 nach k_3 der Liste, die die möglichen Positionen beinhaltet, hinzugefügt werden. Diese sind: k_2, v_{10}, v_{11} und k_3. Somit schlägt der Algorithmus für die anschließende Berechnung folgende Positionen vor: s, v_6, v_7, v_8, k_2, v_1, v_2, v_{16}, k_3, v_{10} und v_{11}.

Nachdem eine Liste der möglichen Positionen bestimmt wurde, wird diese durchgegangen und die Redundanz r mit jedem Knoten der Liste im Graphen nacheinander verbunden. Anschließend werden die folgenden beiden Fälle unterschieden:

1. **Knoten v entspricht dem Zielknoten t**
   Falls $v = t$ gilt, muss für den Knoten r ein BDD generiert werden, das alle möglichen Pfade aller k-Knoten ohne v zu r abbildet, und anschließend mit einem logischen "oder" mit dem BDD, das für v und seine möglichen Redundanzen generiert wurde, verknüpft werden. Dies bedeutet, dass zuerst der "Decompose"- (siehe Alg. 5.3) und dann der "Compose"-Algorithmus (siehe Alg. 5.7) für r ausgeführt werden muss.

2. **Knoten v entspricht nicht dem Zielknoten t**
   Für diesen Fall ist der Rechenaufwand wesentlich geringer, da lediglich das BDD generiert werden muss, das alle möglichen Pfade von r zu einem gegebenen Zielknoten t abbildet. Dieses BDD kann mit den Algorithmen "PathConstruct" (siehe Alg. 5.2) sowie "Compose" (siehe Alg. 5.7) erstellt werden.

   Anschließend muss dieses BDD mit dem bereits generierten BDD für den Knoten, der die Menge K besitzt, verbunden werden. Dabei muss das BDD für den Knoten r mit einem logischen "oder" mit dem BDD des Knotens v inkl. seiner möglichen Redundanzen verknüpft werden. Das aus dieser Verbindung resultierende BDD muss nun mit den BDDs aller anderen Knoten der Menge $K \backslash \{v, t\}$ mit einem logischen "und" verknüpft werden.

   Dies wird im Algorithmus 5.13 "FindPlacesForRedundancy" in Zeile 14 mit der Funktion *merge* erzielt. Wie dies im Detail gemacht wird, ist von der Implementierung abhängig. Eine Möglichkeit wäre, die BDDs der k-Knoten zum Zielknoten t in einer Liste zu speichern, während die Zuverlässigkeit berechnet wird. Danach könnte diese Liste in der Funktion *merge* verwendet werden.

Sobald das neue BDD erstellt wurde, kann der Algorithmus 5.8 "CalcRel" aufgerufen und dadurch die Zuverlässigkeit inkl. des Knotens r an einer bestimmten Position berechnet werden. Danach erfolgt die Überprüfung, ob die neu berechnete Zuverlässigkeit mindestens der gewünschten Zuverlässigkeit entspricht. Ist dies der Fall, wird jene Position inklusive der berechneten Zuverlässigkeit in eine Liste, die absteigend nach dem Wert der Zuverlässigkeit sortiert wird, abgespeichert. Abschließend wird die Verbindung zwischen dem Knoten r und dem jeweiligen Knoten aus der Liste der möglichen Positionen wieder entfernt.

Der Rückgabewert des Algorithmus 5.13 "FindPlacesForRedundancy" ist die sortierte Liste mit den jeweiligen Positionen und den dazugehörigen Zuverlässigkeitswerten.

Algorithm 5.13 FindPlacesForRedundancy – Unterbreitet Vorschläge für die Positionierung einer Redundanz inkl. der sich dadurch verändernden Zuverlässigkeit

Require: Graph G, Node v, Reliability $wantedRel$, Target t, BDD $bddResult$

```
1:  List places ▷ Sortierte Liste, die zu der jeweiligen Position auch die Zuverlässigkeit speichert
2:  Node r = CreateRedundancy(v)    ▷ Erzeugt Knoten r mit einer Kante, p(r) = p(v)
3:  List PossibleNodes
4:  if |K\{v}| == 1 then
5:      PossibleNodes = V
6:  else
7:      PossibleNodes = ComputePossibleNodes(G, K\{v})
8:  for all pn ∈ PossibleNodes do
9:      connect(r, pn)
10:     BDD bddMerged
11:     if v == t then
12:         bddMerged = bddResult ∨ Compose(Decompose(G, K\{v}, r))
13:     else
14:         bddMerged = merge(bddResult, Compose(PathConstruct(G, r, t))
15:     Reliability newRel = CalcRel(bddMerged)
16:     if newRel ≥ wantedRel then
17:         places.add(pn, newRel)
18:     disconnect(r, pn)
19: return places
```

Für das oben angeführte Beispiel würde der Algorithmus die in der Tabelle 5.3 angegebenen Daten ausgeben. Die Zuverlässigkeit für den Knoten s des Beispielnetzwerks beträgt 0.458793.

5.5 Annahmen und Anwendungsfälle

Nachdem in diesem Kapitel bisher die Funktionsweise der Berechnung der Zuverlässigkeit sowie des Vorschlagsystems beschrieben wurde, fokussiert sich dieser Abschnitt auf die Anwendungsmöglichkeiten sowie Einschränkungen und Annhamen, die vorausgesetzt werden.

Position	Zuverlässigkeit von s mit r
$v_3 = s$	0.5134282185718516
$v_9 = k_2$	0.5134282185718516
$v_{14} = k_3$	0.5134282185718516
v_{16}	0.5109357547440265
v_{10}	0.5107533932413183
v_7	0.5095444648965662
v_8	0.5094922376284332
v_6	0.5084087619560050
v_1	0.5083895412087551
v_{11}	0.5079068818263781
v_2	0.5040965304297409

Tabelle 5.3: Die jeweiligen Zuverlässigkeitswerte für den Knoten s, falls die Redundanz r bei den entsprechenden Positionen mit dem Graphen (siehe Abbildung 5.8) über eine Kante e verbunden wird.

5.5.1 Annhamen und Einschränkungen

Das Graph-Modell, bestehend aus Knoten und Kanten, ist ein sehr stark vereinfachtes, Modell mit statischen Eigenschaften, in das einige Informationen nicht aufgenommen werden. In diesem Punkt werden einige Annahmen, die für die Abstrahierung getroffen wurden, beschrieben.

Alle Berechnungen die durchgeführt werden, basieren auf einem Modell mit statischen Eigenschaften, d. h. es erfolgt keine Betrachtung zur Laufzeit. Dadurch erfolgt auch keine Verfügbarkeitsanalyse des Netzwerks nach x Zeiteinheiten oder hinsichtlich der Frage wie eine k-Komponente, die ausfällt, ersetzt werden kann (Reparaturzeiten).

Weiterhin besteht nicht die Möglichkeit einer Priorisierung der Netzentitäten, d. h. weder Knoten bzw. Komponenten noch Kanten bzw. Verbindungen können in irgendeiner Form gegenüber anderen Entitäten priorisiert werden. Dadurch kann bei der automatischen Generierungen von Redundanzen durch das Vorschlagsystem nicht angegeben werden, welche Entitäten eher Redundanzen erhalten sollten. Die Erstellung der Redundanzen wird dabei ausschließlich aufgrund der Erhöhung der Zuverlässigkeit durchgeführt, d. h. es erhalten diejenigen Entitäten Redundanzen, bei deren Ausfall sich die Zuverlässigkeit am meisten verringert.

Des Weiteren wird keine Betrachtung der Kosten von Entitäten durchgeführt. Es könnte z. B. den Fall geben, dass das Einbauen einer Kante für zwei bestehende Kanten für die Zuverlässigkeit noch ausreichend wäre, allerdings die Kosten verringern würde. Dies trifft auch für den Teil "Verringern der Zuverlässigkeit" des Vorschlagsystems zu. Der Algorithmus hierfür entfernt Entitäten ausschließlich aufgrund der Information, wie unkritisch Entitäten sind, und nicht auf Grundlage der Frage, ob diese aus Kostengründen

entfernt werden sollten.

Der komplette Netzplan wird als Graph-Modell abstrahiert und enthält dabei keine Informationen bezüglich der geografischen Gegebenheiten des Netzwerks, somit können keine entsprechenden Entscheidungen aufgrund dieser Informationen getroffen werden.

Bei der Frage, ob das Netzwerk funktionsfähig ist oder nicht, werden keine Zwischenstufen behandelt. Es sind nur die k-Knoten dafür ausschlaggebend, ob das Netzwerk funktioniert. Sollte einer nicht mehr erreichbar sein, wird das Netzwerk als nicht mehr funktionsfähig angesehen. Es gibt also keine Möglichkeit, einem der k-Knoten eine höhere Priorität zuzuteilen. Das betrifft beispielsweiße Fälle, in denen das Netzwerk noch betrieben werden kann, fall Dienst A ausfällt, aber nicht, falls Dienst B ausfällt.

Da das Graph-Modell ausschließlich die physische Topologie des Netzwerks beschreibt, ist die Konnektivität zwischen allen Komponenten uneingeschränkt. Dies bedeutet, dass z. B. Firewalls oder Router, die den Netzverkehr nach bestimmten Vorgaben regeln und dabei die Konnektivität zwischen Komponenten einschränken können, vom Modell nicht entsprechend berücksichtigt werden.

Bei der Generierung von Redundanzen durch das Vorschlagsystem werden für die Redundanzen immer dieselben Zuverlässigkeitswerte verwendet wie für die ursprünglichen Entitäten. Diese Werte können also nicht verändert werden. Des Weiteren wird die Zuverlässigkeit nur durch das Einfügen von Redundanzen und nicht durch das Verändern der Zuverlässigkeitswerte der einzelnen Entitäten erhöht. Dies bedeutet, dass der Nutzer Entitäten verwenden sollte, deren Zuverlässigkeitswerte in einem vernünftigen Verhältnis zur gewünschten Gesamtzuverlässigkeit stehen. Ist dies nicht der Fall, würde das Vorschlagsystem etliche Redundanzen einbauen, wodurch das Netzwerk nicht mehr rentabel wäre.

Um dies exemplarisch zu zeigen, betrachten wir den Netzwerk-Graphen $G = (V, E)$ aus Abbildung 5.2 mit 4 Knoten und 5 Kanten. Angenommen, alle Entitäten weisen dieselbe Ausfallsicherheit von $p(x_i) = 0.9$ mit $x_i \in V \cup E$ auf und der Nutzer gibt eine gewünschte Gesamtzuverlässigkeit von $R_{Gew} = 0.97$ an. Um dies mit Redundanzen zu erreichen, würde das System einen neuen Netzplan mit insgesamt 7 Knoten und 14 Kanten vorschlagen, wodurch sich das Netzwerk bezüglich der Knoten fast verdoppelt und bezüglich der Kanten fast verdreifachen würde.

5.5.2 Anwendungsfälle

Mit dem Algorithmus zur Bestimmung der Zuverlässigkeit eines Netzwerks mit k-Terminals kann für jede beliebige Komponente die Zuverlässigkeit unter Berücksichtigung der Erreichbarkeit der k-Komponenten errechnet und somit die Gesamtzuverlässigkeit eines Netzwerks berechnet werden. Der Algorithmus funktioniert für jegliche Topologie sowie für ungerichtete und gerichtete Netzwerke. Da allerdings die zu Grunde

liegende Problemstellung bei der Bestimmung aller möglichen Pfade von k-Terminals NP-schwer [5, 6] ist, kann der Algorithmus je nach Topologie in Verbindung mit der Größe des Netzwerks die Zuverlässigkeit nicht in vertretbarer Zeit berechnen. In Kapitel 7 werden zu diesem Thema verschiedene Evaluierungen durchgeführt.

Für das Vorschlagsystem gibt es mehrere Anwendungsmöglichkeiten. Durch den Algorithmus 5.10 "ImproveReliability" besteht die Möglichkeit, die Zuverlässigkeit eines gegebenen Netzwerks durch das Einfügen von Redundanzen auf einen gewünschten Wert zu erhöhen. Dabei muss, wie oben bereits erwähnt, darauf geachtet werden, dass die einzelnen Zuverlässigkeitswerte der Entitäten in einem "vernünftigen" Verhältnis zu der gewünschten Zuverlässigkeit stehen.

Des Weiteren kann für den Fall, dass der Nutzer eine Redundanz einfügen möchte, das Vorschlagsystem daraufhin abgefragt werden, wie sich die Zuverlässigkeit des Netzwerks aufgrund der Positionierung der Redundanz verändert. Hierfür liefert der Algorithmus 5.13 "FindPlacesForRedundancy" mehrere Vorschläge zur Positionierung der Redundanz im Verhältnis zur Zuverlässigkeit. Zusätzlich kann dem Algorithmus angegeben werden, dass nur Positionen für die Redundanz ausgegeben werden, mit denen eine gewünschte Mindestzuverlässigkeit erzielt wird.

Anschließend kann der Nutzer die für ihn am besten geeignete Positionierung wählen. Dabei spielt nicht nur die Maximierung der Zuverlässigkeit eine Rolle, sondern auch die Frage, an welcher Stelle im realen Netzwerk Platz für die Redundanz ist.

Mit dem Algorithmus 5.11 "ReduceReliability" kann die Zuverlässigkeit eines Knotens, der eine Menge K-Knoten besitzt, auf einen gewünschten Wert reduziert werden. Dies ist vor allem für sehr "überladene" sowie "zu sichere" Netzwerke sinnvoll. Weiterhin können damit Entitäten aus dem Netzwerk entfernt werden, die für die Zuverlässigkeit nicht relevant sind.

Kapitel 6

Implementierung

In diesem Kapitel wird die Implementierung des Programms beschrieben. Dabei wird die eingesetzte Entwicklungsumgebung mit dem Aufbau ihrer Datenstruktur und den verwendeten Bibliotheken angeführt. Anschließend wird auf die Berechnung der Zuverlässigkeit sowie des Vorschlagsystems eingegangen. Zum Abschluss wird noch erklärt, wie das Programm ausgeführt und verwendet werden kann.

6.1 Entwicklungsumgebung

Das Programm wurde in Java 1.8.0_45 implementiert. Als Entwicklungsumgebung kam Eclipse 4.4.2 Luna auf Ubuntu 14.04 64-Bit zum Einsatz. Als einzige zusätzliche Bibliothek wurde JavaBDD-1.0b2 [55] für die Repräsentation der BDD-Datenstruktur verwendet.

6.2 Datenstruktur

Es wurde ein Graph-Modell mit Knoten und Kanten, die beide ausfallen können, implementiert. Das Modell wird benötigt, um die entsprechenden Berechnungen bzw. Algorithmen ausführen zu können.

6.2.1 Netzwerk-Graph

Die Datenstruktur, die den Netzwerk-Graphen abbildet, besteht aus vier Klassen. Die Klasse "Node" repräsentiert die Knoten des Graphen und die Klasse "Link" entsprechend die Kanten. Dabei besitzt ein "Node" keine Informationen bezüglich seiner Verbindungen mit anderen Knoten. Allerdings kann ein Knoten in einem Array Referenzen zu seinen k-Knoten speichern, damit die Zuverlässigkeit für ihn unter der Berücksichtigung seiner k-Knoten berechnet werden kann.

Ein "Link" beinhaltet zwei Referenzen für Quell- und Zielknoten. Beide Klassen erben von der abstrakten Klasse "Component", um Knoten und Kanten zusammen in einem Container abbilden zu können. Dies wird z. B. für das Identifizieren kritischer Entitäten benötigt. Außerdem besitzen Knoten und Kanten einige gleiche Eigenschaften, wie z. B. die Ausfallsicherheit p, wodurch diese in der abstrakten Klasse "Component" abgebildet werden können.

Die komplette Struktur des Netzwerk-Graphen beinhaltet die Klasse "Graph". Die Implementierung der Datenstruktur ist sowohl für gerichtete als auch ungerichtete Graphen ausgelegt. Die Redundanzen der Knoten werden im Graphen als Map gespeichert. Dagegen werden die Redundanzen von Kanten durch einen Zähler in der Klasse "Link" gespeichert.

6.2.2 BDD

Wie bereits erwähnt, wurde für die BDD-Datenstruktur die bestehende Programmbibliothek JavaBDD-1.0b2 eingesetzt. Die JavaBDD-API basiert auf der weitverbreiteten BDD-Bibliothek BuDDy, die in C geschrieben wurde. JavaBDD wurde komplett in Java implementiert. Des Weiteren kann JavaBDD mit der JDD-Bibliothek über ein Interface kommunizieren oder mit den drei in C geschriebenen BDD-Bibliotheken BuDDy, CUDD und CAL über ein JNI-Interface verbunden werden. Da JavaBDD für alle diese Programmbibliotheken eine universelle Schnittstelle zur Verfügung stellt, kann zwischen ihnen gewechselt werden, ohne Änderungen am Programm vornehmen zu müssen [55].

6.3 Berechnung der Zuverlässigkeit

Für die Berechnung der Zuverlässigkeit wurde der Algorithmus 5.1 "KLYMethod" entsprechend programmiert. Dabei wird auf Basis der Knoten, die ein Array an K-Knoten besitzen, die Zuverlässigkeit für k-Terminals bestimmt. Daraus kann dann die Gesamtzuverlässigkeit des Netzwerks abgeleitet werden.

Die erstellten Methoden zur Bestimmung der Zuverlässigkeit von k-Terminals sind bezüglich ihrer Struktur gleich aufgebaut wie die beschriebenen Algorithmen im Abschnitt 5.3. D. h., es gibt für jeden dieser Algorithmen eine entsprechend implementierte Methode im Programm. Durch die modulare Programmierung kann für die Bestimmung der möglichen Pfade zwischen den k-Terminals auch eine andere BDD-basierte Methode, wie z. B. die Decomposition-Methode von Hardy et al. [31], verwendet werden. Sowohl die Funktion zum Hinfzufügen der Knoten als auch die Funktion zur Berechnung der Zuverlässigkeit eines BDD-Graphen sind unabhängig.

Das Caching der isomorphen Teilgraphen wurde sehr speichereffizient implementiert. Um dies zu erzielen, wurde eine Klasse "GraphHash" erstellt, die als Attribute den

Hash-Code des Quell- und Zielknotens sowie den des Graphen besitzt. Bei Überprüfung auf Gleichheit zweier "GraphHash"-Objekte werden diese drei Attribute miteinander verglichen. Die Objekte dieser Klasse werden als Schlüssel für die Hashtabelle des Algorithmus 5.2 "PathConstruct" genutzt. Somit bildet diese Hashtabelle "GraphHash"- auf BDD-Objekte ab und stellt damit das Caching dar.

6.4 Vorschlagsystem

In diesem Abschnitt wird das Vorschlagsystem beschrieben, in welchem zuerst die kritischen Komponenten/Verbindungen des Netzwerks identifiziert werden und anschließend die Zuverlässigkeit optimiert wird.

6.4.1 Identifizieren kritischer Entitäten

Es wurden zwei voneinander unabhängige Methoden implementiert, um die kritischen Entitäten identifizieren zu können. Dabei basieren beide auf der Idee von Yeh et al. [2].

Bei der ersten Methode wurde die Funktion "bdd.forAll()" der JavaBDD-Bibliothek verwendet, um eine Entität aus dem gegebenen BDD-Graphen, der alle möglichen Pfade zwischen den k-Terminals inkl. der Knoten abbildet, zu entfernen. Anschließend wird die Funktion zur Berechnung der Zuverlässigkeit eines BDDs – entspricht dem Algorithmus 5.8 "CalcRel" – aufgerufen, um jene ohne die entfernte Entität zu bestimmen. Dies wird für jede Entität des Netzwerks durchgeführt, um die kritischen Entitäten zu bestimmen. Falls der Ausfall mehrerer Entitäten gleichzeitig betrachtet wird, muss für jede dieser Entitäten der entsprechende BDD-Knoten, der diese Entität abbildet, aus dem BDD-Graphen durch Aufruf der Funktion "bdd.forAll()" entfernt werden.

In der zweiten Methode wird eine globale Liste verwendet, um zu definieren, welche Entitäten des Netzwerks nicht berücksichtigt werden sollen. Außerdem wird eine adaptierte Version der Funktion zur Berechnung der Zuverlässigkeit eines BDDs aufgerufen. In dieser rekursiven Funktion wird in jedem Durchgang überprüft, ob der gerade abzuarbeitende BDD-Knoten eine Entität darstellt, die in der Liste vorkommt. Ist dies der Fall, errechnet sich die Zuverlässigkeit dieses BDD-Knotens zu 100% aus dem BDD-Kinder-Knoten "low".

Diese Methode wird auch von der Funktion zum Verringern der Zuverlässigkeit genutzt. Für beide Ansätze gilt, dass bei einer Kante, die Redundanzen besitzt, nicht der Ausfall der Kante inkl. ihrer Redundanzen betrachtet wird, sondern lediglich der Zähler für ihre Redundanzen um 1 reduziert wird.

6.4.2 Erhöhen der Zuverlässigkeit

Um die Zuverlässigkeit erhöhen zu können, wurde eine Methode entsprechend dem Algorithmus 5.10 "ImproveReliability" implementiert. Für das dabei angewandte Näherungsverfahren wird ein globales Array eingesetzt, in dem die Anzahl der Redundanzen der Entitäten gespeichert werden. Dies sind die Redundanzen, die zum Erhöhen der Zuverlässigkeit hinzugefügt werden.

Um nun die Zuverlässigkeit nach jedem Hinzufügen einer Redundanz zu bestimmen, wurde eine Funktion implementiert, die eine adaptierte Version des Algorithmus 5.8 "CalcRel" darstellt. In dieser Funktion wird in jedem Durchgang während des Abarbeitens der BDD-Knoten, die jeweils eine Entität abbilden, das zuvor erwähnte globale Array danach überprüft, ob von dieser Entität Redundanzen erzeugt wurden. Ist dies der Fall, wird die Ausfallsicherheit p der Entität durch die erzeugten Redundanzen entsprechend erhöht.

6.4.3 Verringern der Zuverlässigkeit

Für die Implementierung dieser Funktion wurde strikt der Algorithmus 5.11 "ReduceReliability" verwendet. Für die Berechnung der Zuverlässigkeit nach dem jeweiligen Entfernen der Entitäten wurde, wie bereits erwähnt, die zweite Methode zum Identifizieren der kritischen Entitäten verwendet.

6.4.4 Vorschläge zur Positionierung von Redundanzen

Diese Funktionalität wurde ebenfalls strikt nach dem Algorithmus 5.13 "FindPlacesForRedundancy" implementiert. Die Funktion *merge* aus Zeile 14 des Algorithmus wurde, wie in Abschnitt 5.4.4 beschrieben, implementiert.

Während der Berechnung der Zuverlässigkeit werden die jeweiligen BDDs aller $(k - 1)$ Terminal-Paare inklusive ihrer Redundanzen in einer globalen Liste gespeichert. Diese müssen anschließend nur noch entsprechend mit dem generierten BDD, das alle möglichen Pfade der Redundanz zum Zielknoten t darstellt, verknüpft werden.

6.5 Benutzbarkeit

Um das Programm ausführen zu können, wird nur die entsprechende Java-Version sowie die externe BDD-Bibliothek JavaBDD benötigt. Die ausführbare JAR-Datei funktioniert sowohl für Linux-Systeme als auch für Microsoft Windows.

Für die Benutzbarkeit wurde ein konsolenbasiertes UI entwickelt, in dem die entwickelten Algorithmen beliebig ausgeführt werden können. Sobald das Programm gestartet wird, muss eine Textdatei angegeben werden, die den Netzplan enthält. Dafür wurde die folgende Grammatik $G = (V, \Sigma, P, S)$ definiert, damit der Parser des Programms den Netzplan richtig verarbeiten kann.

$$V = \{NET, VL, RL, EL, KL, VT, RKT, ET, RK, E, VX, REL, D, DL, BR, SL, FL, ST, FT\}$$

$$\Sigma = \{\text{'V'}, \text{'R'}, \text{'E'}, \text{'K'}, \text{'='}, \text{'\{'}, \text{'\}'}, \text{'['}, \text{']'}, \text{'\textbackslash'}, \text{'n'}, \text{','}, \text{':'}, \text{'0'}, \text{'1'}, \text{'2'}, \text{'3'}, \text{'4'}, \text{'5'}, \text{'6'}, \text{'7'}, \text{'8'}, \text{'9'}\}$$

$$P = \left\{ \begin{array}{ll}
NET & \rightarrow VL\ BR\ (RL\ BR \mid \epsilon)\ EL\ BR\ KL \\
VL & \rightarrow \text{'V'}\ SL\ VT\ FL \\
RL & \rightarrow \text{'R'}\ SL\ RKT\ FL \\
EL & \rightarrow \text{'E'}\ SL\ ET\ FL \\
KL & \rightarrow \text{'K'}\ SL\ RKT\ FL \\
VT & \rightarrow ST\ VX\ DL\ REL\ FT\ (DL\ VT \mid \epsilon) \\
RKT & \rightarrow ST\ VX\ \text{'='}\ RK\ FT\ (DL\ RKT \mid \epsilon) \\
ET & \rightarrow ST\ E\ FT\ (DL\ ET \mid \epsilon) \\
RK & \rightarrow VX\ (DL\ RK \mid \epsilon) \\
E & \rightarrow VX\ DL\ VX\ DL\ REL \\
VX & \rightarrow D \\
REL & \rightarrow (\text{':'}\ D) \mid \text{'1'} \\
D & \rightarrow (\text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \text{'3'} \mid \text{'4'} \mid \text{'5'} \mid \text{'6'} \mid \text{'7'} \mid \text{'8'} \mid \text{'9'})\ (D \mid \epsilon) \\
DL & \rightarrow \text{','} \\
BR & \rightarrow \text{'\textbackslash'}\ \text{'n'} \\
SL & \rightarrow \text{'='}\ \text{'\{'} \\
FL & \rightarrow \text{'\}'} \\
ST & \rightarrow \text{'['} \\
FT & \rightarrow \text{']'}
\end{array} \right\}$$

$$S = NET$$

Falls die Datei dieser Grammatik nicht entspricht, bricht das Programm ab. Ist der Dateiinhalt korrekt, wird ein Graph aus dem Netzplan generiert.

Anschließend kann durch die Eingabe einer Ziffer von 0–9 eine der folgenden Funktionen durch das Programm ausgeführt werden:

[1] Gesamtzuverlässigkeit des Netzwerks berechnen

[2] Zuverlässigkeit einer Komponente berechnen

[3] Kritische Entitäten einer Komponente bestimmen

[4] Zuverlässigkeit einer Komponente erhöhen

[5] Vorschläge für die Positionierung einer Redundanz unterbreiten

[6] Zuverlässigkeit einer Komponente reduzieren

[7] Evaluierung speichern

[8] Graph als Graphviz-Datei abspeichern

[9] Graph abspeichern

[0] Programm beenden

Bei Punkten, bei denen eine Betrachtung auf Komponentenebene (2, 3, 4, 5, 6) erfolgt, muss anschließend die entsprechende Komponente bzw. der Knoten als Zahl angegeben werden. Zum Erhöhen sowie Verringern der Zuverlässigkeit muss zusätzlich ein gewünschter Zuverlässigkeitswert mit Intervall $[0,1]$ angegeben werden. Bei Punkt 5 – "Vorschläge für die Positionierung einer Redundanz unterbreiten" – muss neben der entsprechenden Komponente auch das k-Terminal, von dem eine Redundanz erstellt werden soll, als Zahl angegeben werden. Bei den Punkten 7, 8, 9 muss jeweils ein Dateipfad angegeben werden, um festzulegen, wo die Datei gespeichert werden soll.

Kapitel 7

Evaluierung

In diesem Kapitel wird die Evaluierung des entwickelten Programms beschrieben. Hierfür wird das Kapitel nach drei Kategorien unterteilt. Im ersten Abschnitt wird die qualitative Evaluierung beschrieben. Diese befasst sich mit den Anforderungen, die an das Programm gestellt wurden. Der zweite Punkt beschreibt die theoretische Komplexität für die im Entwurf-Kapitel beschriebenen Algorithmen. Im letzten Abschnitt wird die quantitative Evaluierung durchgeführt, um den Praxisbezug der entwickelten Algorithmen besser zu verdeutlichen.

7.1 Qualitativ

In diesem Abschnitt wird die qualitative Evaluierung durchgeführt. Dabei wird betrachtet, ob und wie die Anforderungen (siehe Kapitel 3) an das Programm erfüllt wurden.

A1 Unabhängigkeit von Topologie
Dies konnte durch das beschriebene Graph-Modell als Datenstruktur erfüllt werden. Durch den ungerichteten Graphen kann jegliche Netzwerk-Topologie entsprechend abgebildet werden. Die entwickelten Algorithmen benötigen dieses Graph-Modell sowie ein BDD als Datenstruktur.

A2 Abhängigkeiten abbilden
A2 wurde ebenfalls durch die Datenstruktur des Graph-Modell realisiert. Hierfür werden virtuelle Kanten e mit einer Ausfallsicherheit von $p(e) = 1$ zwischen zwei Komponenten eingesetzt, um die Hierarchie zwischen ihnen abzubilden. Ein Beispiel dafür ist eine VM, die auf einem Computer-System läuft.

A3 Zuverlässigkeit von k-Terminals bestimmen
Um diese Anforderung zu erfüllen, wurde der Algorithmus von Yeh et al. [2] implementiert, der die Zuverlässigkeit von k-Terminals unter der Verwendung von BDDs

berechnet. Damit der Algorithmus effizienter arbeitet, wurde das Identifizieren und Entfernen von Biconnected-Komponenten im Algorithmus 5.2 "PathConstruct" eingebaut. Dies entspricht dem Ansatz von Lê [1] sowie Lê et al. [36]. Mit dem entwickelten Algorithmus 5.1 "KLYMethod" kann die Zuverlässigkeit für einen beliebigen Graphen und mit einer, beliebigen Menge an K-Knoten bestimmt werden, wodurch die Anforderung erfüllt wurde.

A4 Berücksichtigung von Komponenten
A4 konnte ebenfalls realisiert werden. Dafür muss in der Datenstruktur der Komponente bzw. des Knotens die jeweiligen Ausfallsicherheit abgespeichert werden. Des Weiteren musste der Algorithmus 5.7 "Compose" entwickelt werden, der auf der Idee von Yeh et al. [2], wie der Ausfall von Komponenten abgebildet werden kann, basiert. Dabei ist der Algorithmus unabhängig von der Art der Berechnung der möglichen Pfade zwischen k-Terminals, solange dieser Algorithmus auf der Nutzung von BDDs basiert. Dies bedeutet, dass "Compose" auch für die Decomposition-Methode von Hardy et al. [31] funktioniert.

A5 Berücksichtigung der Verbindungen
Dies wurde ebenfalls durch die Datenstruktur erfüllt, in der die Ausfallsicherheiten der Verbindungen bzw. Kanten gespeichert werden. Der Algorithmus 5.2 "PathConstruct", in dem die Verbindungen verarbeitet und von den generierten BDD-Knoten abgebildet werden, basiert ebenfalls auf der Methode von Yeh et al. [2].

A6 Zuverlässigkeit pro Komponente bestimmen
Diese Anforderung konnte ebenso realisiert werden, weil die Datenstruktur für jede Komponente v, die keine Redundanz ist, die Möglichkeit bietet, eine Menge an K-Komponenten zu besitzen. Für die Komponente v inkl. ihrer K-Komponenten kann die Zuverlässigkeit durch den Algorithmus 5.1 "KLYMethod" bestimmt werden. Wie bereits zuvor erwähnt, basiert dieser auf der Idee von Yeh et al. [2].

A7 Berücksichtigung der Redundanzen von Komponenten, die für die Zuverlässigkeit relevant sind
Diese Anforderung wurde erfüllt durch die Datenstruktur des Graph-Modells, das die Redundanzen der Komponenten entsprechend abbildet. Zusätzlich mussten entsprechende Anpassungen (siehe Punkt 5.3.1) des Algorithmus zur Bestimmung der möglichen Pfade von k-Terminals der KLY-Methode vorgenommen werden, da Yeh et al. [2] keine Redundanzen berücksichtigen. Diese Anpassungen wurden im Algorithmus 5.3 "Decompose" implementiert.

A8 Identifizierung kritischer Entitäten
A8 konnte durch die Implementierung des Algorithmus 5.9 "CalcEV" erfüllt werden. Dabei basiert der Algorithmus auf der Idee der "Essential Variables" von Yeh et al. [2]. Der Algorithmus wurde für die Verwendung von Redundanzen der k-Terminals erweitert, indem der Ausfall von mehreren Entitäten gleichzeitig betrachtet wird und nicht nur der Ausfall einer Entität. Dabei entspricht die Anzahl der gleichzeitig zu betrachtenden

Ausfälle von Entitäten der maximalen Anzahl an Redundanzen eines k-Terminals (siehe Abschnitt 5.4.1).

A9 Erhöhen der Zuverlässigkeit einer Komponente durch Redundanzen
Diese Anforderung wurde durch die Entwicklung des Algorithmus 5.10 "Improve-Reliability" erfüllt. Um die Zuverlässigkeit zu erhöhen, werden Redundanzen eingefügt. Dabei werden als Metrik für die Bestimmung, von welchen Entitäten eine Redundanz erzeugt werden soll, die kritischen Entitäten des Algorithmus 5.9 "CalcEV" verwendet. Wie bereits erwähnt, basiert dieser Algorithmus auf der Idee der "Essential Variables" von Yeh et al. [2]. Zusätzlich wird der Algorithmus 5.1 "KLYMethod" benötigt, um die Zuverlässigkeit zu bestimmen, bevor sie erhöht werden kann. Während des schrittweisen Erhöhens der Zuverlässigkeit durch das Einfügen von Redundanzen wird die Zuverlässigkeit durch ein selbstentwickeltes Näherungsverfahren, das mindestens der exakten Zuverlässigkeit entspricht, berechnet. Sobald die gewünschte Zuverlässigkeit erreicht ist, wird sie durch die KLY-Methode exakt berechnet (siehe Punkt 5.4.2).

A10 Verringern der Zuverlässigkeit einer Komponente
A10 konnte durch die Implementierung des Algorithmus 5.11 "ReduceReliability" realisiert werden. Wie bei der Anforderung A9 verwendet auch dieser Algorithmus die "Essential Variables" von Yeh et al. [2]. Wobei "ReduceReliability" die unkritischsten Entitäten benötigt, um dadurch jene Entitäten zu entfernen, die die Zuverlässigkeit am geringsten reduzieren. Da die Zuverlässigkeit einer Komponente bestimmt werden muss, bevor sie verringert werden kann, wird die einmalige Ausführung des Algorithmus 5.1 "KLYMethod" benötigt.

A11 Vorschläge zur Positionierung einer Redundanz eines k-Terminals
Diese Anforderung wurde durch die Entwicklung des Algorithmus 5.13 "FindPlacesFor-Redundancy" erfüllt. Hierfür muss ebenfalls durch die KLY-Methode die Zuverlässigkeit zuvor berechnet werden. Anschließend werden mit dem Dijkstra-Algorithmus, falls $k > 2$ ist, mögliche Positionen bzw. Knoten für die Redundanz durch den Algorithmus 5.12 "ComputePossibleNodes" ermittelt. Mit diesen Knoten wird die Redundanz verbunden und die neue Zuverlässigkeit wieder durch die KLY-Methode berechnet.

Tabelle 7.1 vergleicht welche Anforderungen an das Programm von dieser Ausarbeitung sowie den verwandten Arbeiten erfüllt wurden.

7.2 Theoretische Komplexität

In diesem Abschnitt wird die Komplexitätsanalyse der Laufzeit für den Algorithmus zur Berechnung der Gesamtzuverlässigkeit eines Netzwerks sowie für die Algorithmen des Vorschlagsystems durchgeführt.

Quelle	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11
Yeh et al. [2, 3, 34]	+	−	+	+	+	−	−	+	−	−	−
Ayoub et al. [19]	+	−	+	−	+	−	−	−	−	−	−
Lê [1] und Lê et al. [20, 36]	+	−	+	+	+	−	−	−	−	−	−
Hardy et al. [31, 48]	+	−	+	−	+	−	−	+	+	−	−
Herrmann et al. [37, 38]	+	−	+	−	+	−	−	−	−	−	−
Sprotte [39]	+	−	+	−	+	−	−	−	−	−	−
Lin [44]	+	−	−	+	+	−	−	−	−	−	−
Menth et al. [45–47]	+	−	−	+	+	−	−	+	+	−	−
Abd-El-Barr et al. [50]	+	−	−	−	+	−	−	−	+	−	−
Elshqeirat et al. [51, 52]	+	−	−	−	+	−	−	−	+	−	−
Diese Arbeit	+	+	+	+	+	+	+	+	+	+	+

Tabelle 7.1: Welche Anforderungen von den verwandten Arbeiten sowie von der vorliegenden Ausarbeitung erfüllt wurden

7.2.1 Berechnung der Zuverlässigkeit von k-Terminals

Dieser Punkt beschreibt den Komplexitätsaufwand zur Berechnung der Zuverlässigkeit eines Netzwerks. Die folgenden drei Abschnitte beschreiben die Teilgebiete des Algorithmus 5.1 "KLYMethod". Damit wird die gleiche Unterteilung angewandt wie im Entwurf-Kaptiel in Punkt 5.3.

7.2.1.1 Berechnung der möglichen Pfade

Obwohl die KLY-Methode im Worst Case exponenzielle Komplexität in Bezug auf die Anzahl der Komponenten aufweist, kann durch Benchmarks gezeigt werden, dass für viele verschiedene Netzwerk-Topologien eine bessere Perfomanz möglich ist.

Laut [3] terminiert die Expansion des OBDDs im bestmöglichen Fall in h Schritten, wobei h äquivalent zur Länge des kürzesten Pfades ist. Dieser Fall tritt ein, falls die Topologie des Netzwerks dazu führt, dass in jeder Iteration der Expansion identische isomorphe Teilgraphen entstehen und es keine redundanten Knoten gibt.

Im Worst Case wird eine sehr große Expansion generiert, falls es keine isomorphen Teilgraphen gibt. Die generierte Anzahl von G_{node} ist für die exponenzielle Komplexität

verantwortlich, $O\left((d_m)^l\right)$, wobei d_m die maximale Anzahl der Kanten ist, die von einem Knoten wegführen (für gerichtete Netzwerke), und l der längste Pfad zwischen s und t [1]. Der Knoten, der den Wert für d_m bestimmt, muss sich auf einem der möglichen Pfade von s nach t befinden. Dieser Komplexitätsaufwand beschreibt das 2-Terminal-Problem (siehe Algorithmus 5.2 "PathConstruct").

Zur Berechnung der Zuverlässigkeit von k-Terminals werden bei der KLY-Methode $(k-1)$ Terminal-Paare inklusive ihrer Redundanzen miteinander verknüpft, wie im Algorithmus 5.3 "Decompose" beschrieben. Dies bedeutet, dass im Worst Case der Ausdruck $(d_m)^l$ mit $(k-1)$ inklusive der Anzahl der jeweiligen Redundanzen multipliziert werden muss, wobei die Variablen d_m und l durch die $(k-1)$ Terminal-Paare definiert werden. Angenommen, die Menge KTP beschreibt die $(k-1)$ Terminal-Paare inklusive ihrer Redundanzen, dann ergibt sich folgender Ausdruck

$$\sum_{i \in KTP} (d_{m_i})^{l_i}.$$

Der Komplexitätsaufwand wird durch das Maximum von

$$(d_{max})^{l_{max}} = \text{Maximum}\left((d_{m_i})^{l_i}\right), \quad i \in KTP$$

bestimmt. Somit ergibt sich für das k-Terminal-Problem der gleiche Aufwand wie für das 2-Terminal-Problem mit $O\left((d_{max})^{l_{max}}\right)$.

7.2.1.2 Hinzufügen der Knoten

Der Algorithmus 5.7 "Compose" ist ebenfalls rekursiv und wird abhängig von der Größe des BDD-Graphen BDD_G aufgerufen. Angenommen $n = |BDD_G|$ entspricht der Anzahl der BDD-Knoten des BDD-Graphen BDD_G, dann wird "Compose" durch die Verwendung einer Hashtabelle genau $(2n + 1)$ mal aufgerufen, da jeder BDD-Knoten zwei BDD-Kinder-Knoten besitzt und für alle abgearbeiteten BDD-Knoten das Ergebnis in der Hashtabelle gespeichert wird. Ohne eine Hashtabelle würde der Algorithmus im Worst Case n^2 mal für $n > 2$ aufgerufen werden.

Für jeden rekursiven Aufruf werden vier logische "und"-Operationen sowie eine logische "oder"-Operation durchgeführt. Diese Operationen weisen eine Zeitkomplexität von $O(|G_f| \cdot |G_g|)$ auf, wobei G_f und G_g die beiden BDD-Graphen sind. Für weitere Informationen siehe Abschnitt 2.2.2.

Zwei der vier "und"-Operationen sind für die Komplexität vernachlässigbar, da sie die Verknüpfung der jeweiligen Kante mir ihren beiden Endknoten darstellen und somit eine konstante Komplexität aufweisen. Bei den beiden anderen "und"-Operationen ist einer der beiden zu verknüpfenden BDD-Graphen, der der den BDD-Kinder-Knoten "high" oder den BDD-Kinder-Knoten "low" repräsentiert. Ihre Größe liegt zwischen 1 und

$(n - 1 + m)$, wobei m der Anzahl an BDD-Knoten entspricht, die durch das Hinzufügen der Graph-Knoten pro Kante in jedem rekursiven Aufruf entstehen. Dabei ist m linear abhängig von n, da in einem Graph die Anzahl der Knoten maximal der Anzahl an Kanten plus 1 entspricht. Dasselbe gilt für die "oder"-Operation, die eine Verknüpfung der beiden BDD-Kinder-Knoten darstellt.

Im bestmöglichen Fall gibt es nur einen Weg von s nach t, falls der Graph keine Zyklen und jeder Knoten maximal zwei Kanten besitzt. Somit zeigt in jedem rekursiven Aufruf der BDD-Kinder-Knoten "low" auf den Terminal-Knoten 0 (False), wodurch alle "und"- sowie "oder"-Operationen genau n mal ausgeführt werden. Dadurch ergibt sich der Ausdruck

$$3 \sum_{i=1}^{n-1} 3(n - i + m_i), \quad m_i = n - i + 1.$$

Wodurch sich für den Aufwand des bestmöglichen Falls $T_{best} = n^2$ ergibt.

Im Worst Case besitzen die BDD-Graphen für die Kinder-Knoten "low" und "high" jeweils die Hälfte der BDD-Knoten ihres Vaters, wodurch die "oder"-Operation zwischen beiden die Größenordnung n^2 aufweist. Da diese Operation bis zu $2n$ mal ausgeführt wird, ergibt sich eine kubische Komplexität von $O(n^3)$.

7.2.1.3 Berechnung der Zuverlässigkeit

Für den Algorithmus 5.8 "CalcRel" gilt Ähnliches wie für den Algorithmus 5.7 "Compose". "CalcRel" ist wie "Compose" rekursiv und wird proportional zu der Größe des BDD-Graphen BDD_G aufgerufen. Da dieser Algorithmus ebenfalls eine Hashtabelle zur Steigerung der Effizienz verwendet, wird er genau $(2n + 1)$ mal aufgerufen, wobei $n = |BDD_G|$ ist. Da sowohl die Verwendung der Hashtabelle als auch die arithmetischen Operationen eine konstante Komplexität besitzten, weist "CalcRel" eine lineare Komplexität von $O(n)$ auf.

7.2.2 Identifizieren kritischer Entitäten

Kritische Entitäten werden immer pro Knoten, der eine Menge an K-Knoten besitzt, bestimmt und nicht für das gesamte Netzwerk. Dabei werden mögliche kritische Entitäten durch die Menge der Variablen $VAR = V \backslash K \cup E$ definiert. Um kritische Entitäten bestimmen zu können, muss zuvor die Zuverlässigkeit des Knotens, der eine Menge K-Knoten besitzt, berechnet werden. Die Komplexitätsanalyse in diesem Abschnitt bezieht sich ausschließlich auf das Identifizieren kritischer Entitäten und nicht auf die Berechnung der Zuverlässigkeit.

Wie bereits in Punkt 5.4.1 beschrieben, ist der Komplexitätsaufwand abhängig von der Menge der Variablen VAR sowie der Größe des BDD-Graphen BDD_G, der alle möglichen Pfade (Knoten und Kanten) zwischen den k-Terminals abbildet. Bei der selbstentwickelten Methode zur Identifizierung der kritischen Komponenten wird der Algorithmus 5.8 "CalcRel" für jede Variable aufgerufen, d. h. es ergibt sich eine Komplexität von $O(|VAR| \cdot |BDD_G|)$. Dies betrifft den Fall, dass nur der Ausfall einer Variable gleichzeitig betrachtet wird.

Für den Fall, dass r Variablen gleichzeitig ausfallen, ergibt sich eine Komplexität von $O(|VAR|^r \cdot |BDD_G|)$.

7.2.3 Erhöhen der Zuverlässigkeit

Die Komplexität für den Algorithmus zur Erhöhung der Zuverlässigkeit ist exponenziell, da die Zuverlässigkeit mindestens einmal exakt berechnet werden muss und dies exponenziellen Aufwand bedeutet. Des Weiteren wird mindestens einmal der Algorithmus zum Identifizieren kritischer Entitäten durchgeführt, wobei der Fall betrachtet wird, dass mehrere Variablen gleichzeitig ausfallen können. Für das Näherungsverfahren wird eine adaptierte Version des "CalcRel"-Algorithmus mit gleicher Komplexität ausgeführt.

7.2.4 Verringern der Zuverlässigkeit

Dieser Algorithmus nutzt die bereits beschriebenen Algorithmen "CalcRel" zur Berechnung der Zuverlässigkeit eines BDD-Graphen sowie "CalcEV", um die unkritischsten Entitäten zu bestimmen. Dabei wird nur der Fall betrachtet, dass eine Variable und nicht mehrere gleichzeitig ausfallen können.

Der bestmögliche Fall für den Algorithmus 5.11 "ReduceReliability" tritt ein, wenn die Zuverlässigkeit bereits der gewünschten entspricht und der Algorithmus somit beendet werden kann. Im Worst Case ist die gewünschte Zuverlässigkeit, die erst erreicht wird, sobald alle Variablen entfernt wurden, gleich 0. Zusätzlich führt die Entfernung jeder Variablen zur Verringerung der Zuverlässigkeit, wodurch der Algorithmus "CalcEV" ausgeführt wird.

Angenommen $v = |V \backslash K \cup E|$ entspricht der Anzahl an Variablen, bevor eine entfernt wurde. Dann wird "CalcEV" genau v mal aufgerufen, wobei die Menge VAR sich jedesmal um ein Element verringert. Daraus ergibt sich eine quadratische Komplexität bezüglich der Anzahl der Variablen mit

$$O(|VAR|^2 \cdot |BDD_G| + |VAR| \cdot (|E| + |V|)),$$

wobei BDD_G den BDD-Graphen darstellt. Der Ausdruck $|VAR| \cdot (|E| + |V|)$ dient der Überprüfung, ob der Graph zusammenhängend ist.

7.2.5 Vorschläge zur Positionierung von Redundanzen

Da in diesem Algorithmus ebenfalls alle möglichen Wege von einem Ausgangspunkt s zu einem Ziel t berechnet werden, weist Algorithmus 5.13 "FindPlacesForRedundancy" exponenziellen Aufwand auf. Dies betrifft den Best Case wie auch den Worst Case. Die Methode zur Bestimmung der möglichen Positionen zum Einfügen einer Redundanz hat eine Komplexität von

$$O((|E| + |V| \cdot log|V|) \cdot (|K| + r)), \quad r = \sum_{k \in K} |R_k|,$$

wobei R_k die Menge der Redundanzen des Knoten k ist. Eine Beschreibung hierfür befindet sich in Punkt 5.4.4.

Angenommen, pn entspricht der Anzahl der möglichen Positionen, die berechnet wurden, dann ergibt sich ein Komplexitätsaufwand für den Algorithmus 5.13 "FindPlacesForRedundancy" von

$$O\left((|E| + |V| \cdot log|V|) \cdot (|K| + r) + pn \cdot \left((d_{max})^{l_{max}} + |BDD_G|^3 \cdot |BDD_F| + |BDD_{GF}|\right)\right).$$

Dabei ist BDD_G der BDD-Graph, der durch das Einfügen der Redundanz generiert wurde und BDD_F der gegebene BDD-Graph. Der Ausdruck $|BDD_G|^3$ ergibt sich durch den "Compose"-Algorithmus, und die Multiplikation mit $|BDD_F|$ erfolgt aufgrund der Verknüpfung beider BDD-Graphen durch die Funktion *merge*. Der daraus entstehende BDD-Graph wird durch BDD_{GF} dargestellt. Mit diesem wird der Algorithmus 5.8 "CalcRel" ausgeführt, wodurch der letzte Ausdruck $|BDD_{GF}|$ entsteht.

7.3 Quantitativ

In diesem Abschnitt werden mehrere Benchmarks für die entwickelten Algorithmen aufgezeigt, um den Praxisbezug besser zu verdeutlichen. Dabei werden verschiedene Netztopologien mit unterschiedlichen Größen ausgewertet. Diese Topologien werden anschließend miteinander verglichen, um zu zeigen, für welche Topologien der Algorithmus besser und für welche er schlechter geeignet ist. Des Weiteren zeigen die Benchmarks den Unterschied zwischen dem Worst Case und den tatsächlich berechneten Daten bezüglich des Zeitaufwands.

Alle Evaluierungen wurden auf dem gleichen PC durchgeführt. Der Prozessor des Rechners ist ein AMD Phenom II X4 955 mit 4x3.2GHz, 4x512KB L2-Cache und 6MB L3-Cache. Der Computer hat 12GB RAM. Als Betriebssystem wurde Ubuntu 14.04 LTS mit 64-Bit verwendet.

7.3.1 Berechnung der möglichen Pfade

In diesem Abschnitt werden Evaluierungen für den Algorithmus 5.2 "PathConstruct", der alle möglichen Pfade zwischen einem Quellknoten s und einem Zielknoten t berechnet, durchgeführt.

7.3.1.1 Grid-Topologie

Für alle Grid-Netzwerke, die in diesem Kapitel evaluiert werden, gilt, dass sie zwei k-Knoten besitzen, die sich in den gegenüberliegenden Ecken – links oben und rechts unten – befinden.

Abbildung 7.1 zeigt das Verhältnis der Anzahl der Knoten mehrerer Grids (2x2 bis 7x7) zur Laufzeit in ms.

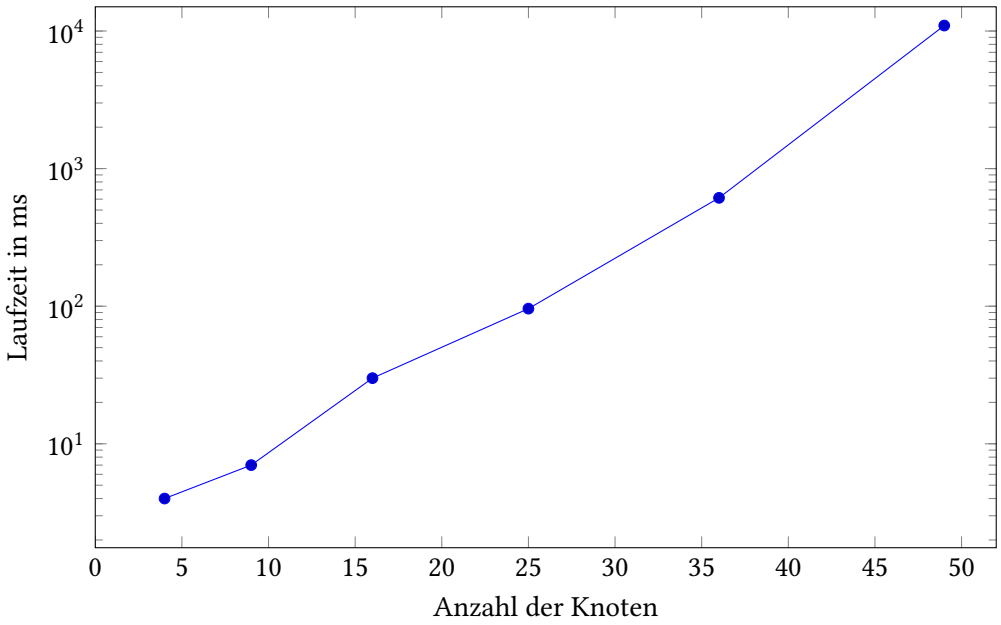


Abbildung 7.1: Verhältnis Anzahl der Knoten mehrerer Grids (2x2 bis 7x7) zur Laufzeit in ms

Abbildung 7.2 zeigt das Verhältnis der Anzahl der Pfade zur Laufzeit für Grid-Netzwerke von 3x10 bis 3x100, wobei die Schrittweite 10 beträgt, 4x4 bis 4x20, 5x5 bis 5x13, 6x6 bis 6x10 und 7x7 bis 7x8.

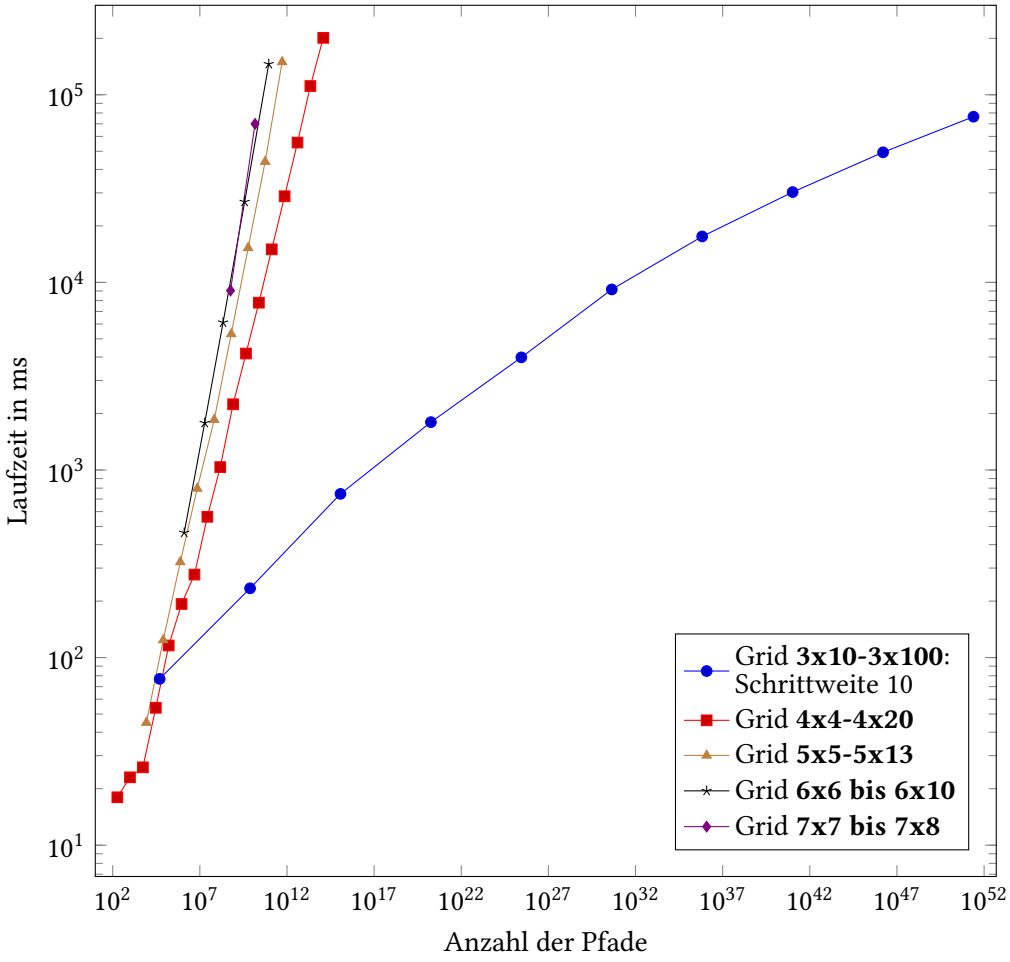


Abbildung 7.2: Verhältnis Anzahl der Pfade zur Laufzeit für Grid-Netzwerke von 3x10 bis 3x100, wobei die Schrittweite 10 beträgt, 4x4 bis 4x20, 5x5 bis 5x13, 6x6 bis 6x10 und 7x7 bis 7x8

Abbildung 7.3 zeigt das Verhältnis der Anzahl der Graph-Knoten zur Anzahl generierter BDD-Knoten bei verschiedenen Grid-Netzwerken.

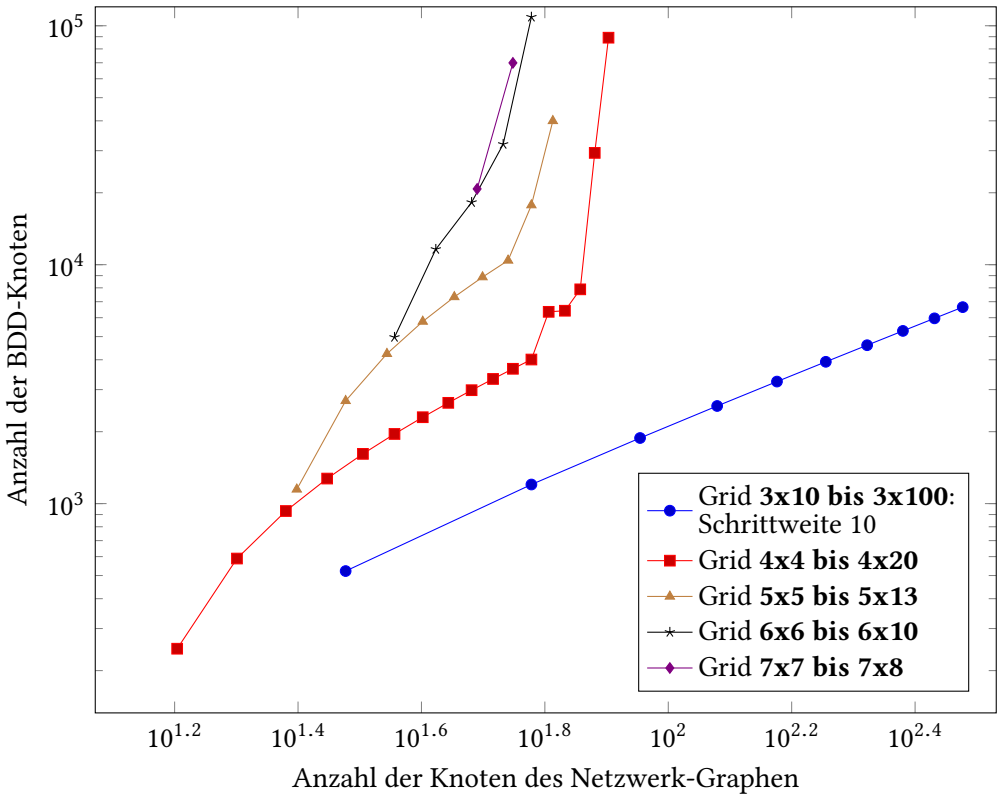


Abbildung 7.3: Verhältnis Anzahl der Graph-Knoten des Grids zu generierten BDD-Knoten

Die Abbildungen 7.4 und 7.5 zeigen die Auswirkung des Cachings auf die Laufzeit. Dabei zeigt Abbildung 7.4 das Verhältnis der Anzahl der Pfade zur Laufzeit in ms bei Grid-Netzwerken von 2x2 bis 2x50 (mit Caching) bzw. 2x2 bis 2x25 (ohne Caching). Bei Grid-Netzwerken von 2xn gibt es genau 2^{n-1} Pfade. Da die Anzahl der Pfade exponenziell steigt, steigt auch die Laufzeit exponenziell ohne die Nutzung des Cachings. Bereits für ein Grid von 2x25 benötigte das Programm ohne Caching eine Laufzeit von 130s. Mit der Verwendung des Cachings erzielte das Programm für ein 2x5000 Grid eine Laufzeit von 86s.

Abbildung 7.5 zeigt das Verhältnis der Anzahl der Pfade zur Anzahl der generierten G_{node} bei Grids von 2x2 bis 2x25 mit und ohne Caching.

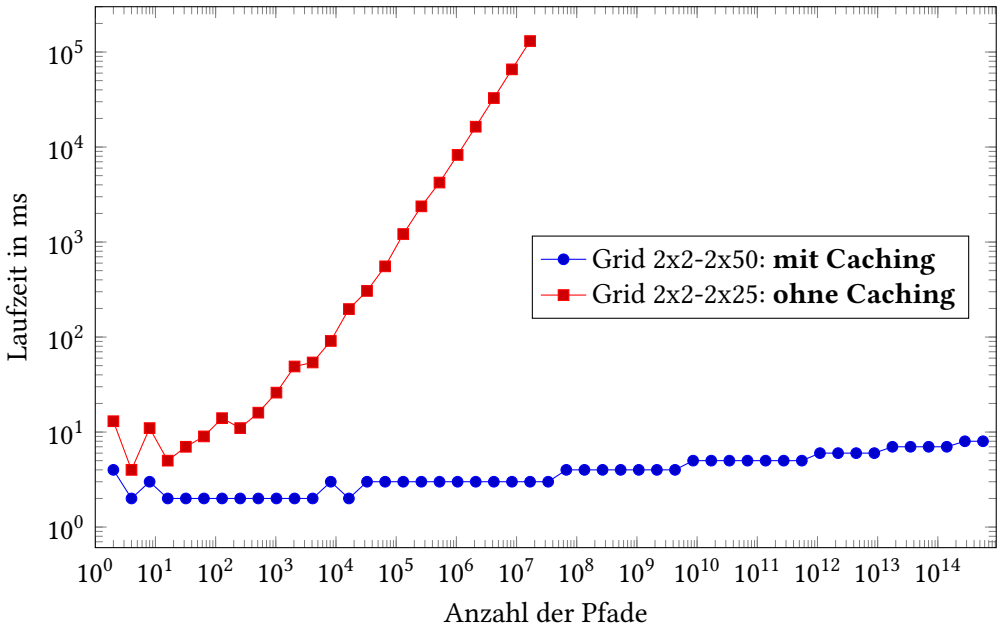


Abbildung 7.4: Verhältnis Anzahl der Pfade zur Laufzeit bei Grid-Netzwerken von 2x2 bis 2x50 mit Caching und 2x2 bis 2x25 ohne Caching

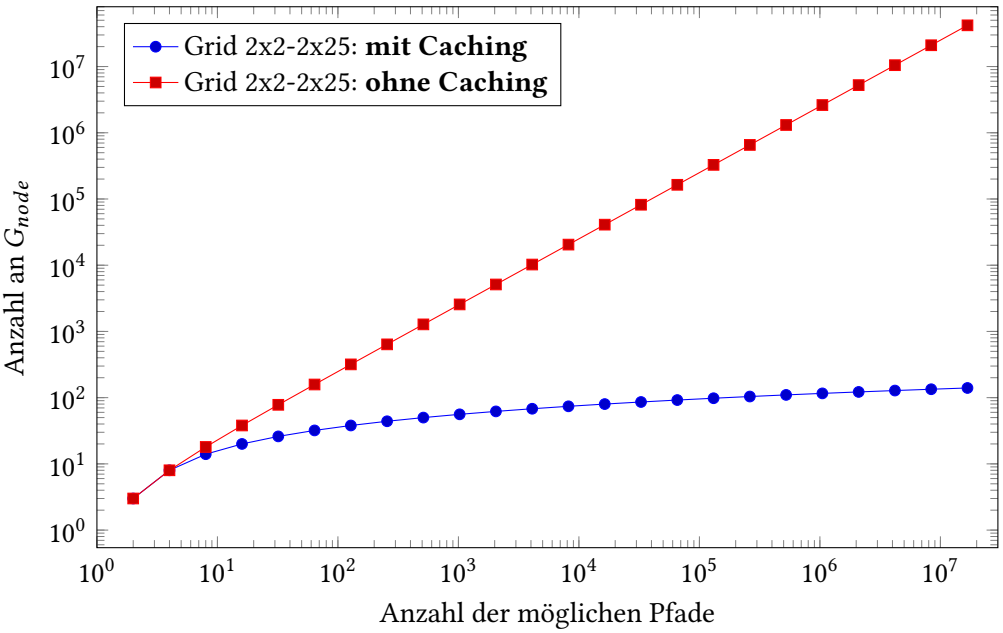


Abbildung 7.5: Verhältnis Anzahl der Pfade zur Anzahl generierter G_{node}, d. h. Aufrufe des Algorithmus 5.2 "PathConstruct" bei Grid-Netzwerken von 2x2 bis 2x25

Abbildung 7.6 zeigt den Vergleich zwischen der originalen KLY-Methode (vgl. [2, 3, 34]) mit der Entfernung redundanter Knoten und der von Lê [1] verbesserten KLY-Methode mit der Entfernung redundanter Biconnected-Komponenten bei einem 5x5 bis 5x11 bzw. 5x13 Grid-Netzwerk.

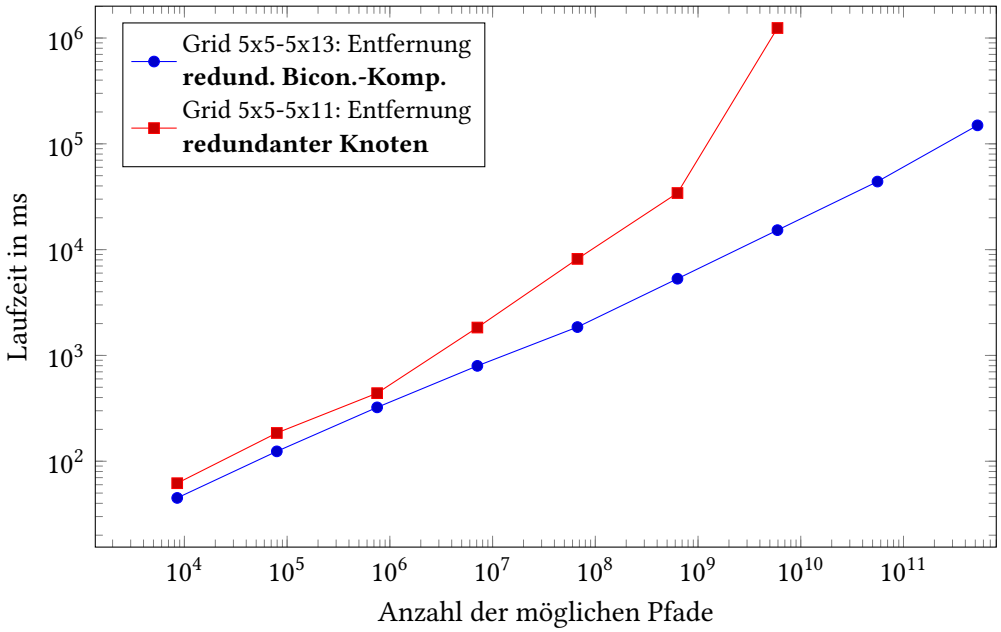


Abbildung 7.6: Vergleich zwischen dem Entfernen redundanter Knoten (originale KLY-Methode [3]) und dem Entfernen redundanter Biconnected-Komponenten (verbesserte KLY-Methode nach Lê [1])

7.3.1.2 Mesh-Topologie

Abbildung 7.7 zeigt das Verhältnis der Anzahl an Pfaden zur Laufzeit in ms bei Full-Mesh Graphen mit 2 bis 12 Knoten, wobei jeder Graph zwei k-Knoten besitzt. Dabei wird die Auswirkung sowohl des Cachings als auch des Entfernens redundanter Biconnected-Komponenten auf die Laufzeit gezeigt.

Obwohl es bei Full-Mesh-Graphen keine redundanten Biconnected-Komponenten gibt, zeigt die Abbildung, dass die Auswirkung auf die Laufzeit minimal ist, weil der Algorithmus zum Identifizieren der Biconnected-Komponenten linearen Aufwand besitzt. Des Weiteren zeigt die Abbildung, dass das Caching auch bei Full-Mesh-Graphen die Laufzeit verringert, auch wenn der Unterschied wesentlich geringer ist als bei den gezeigten Beispielen mit Grid-Graphen.

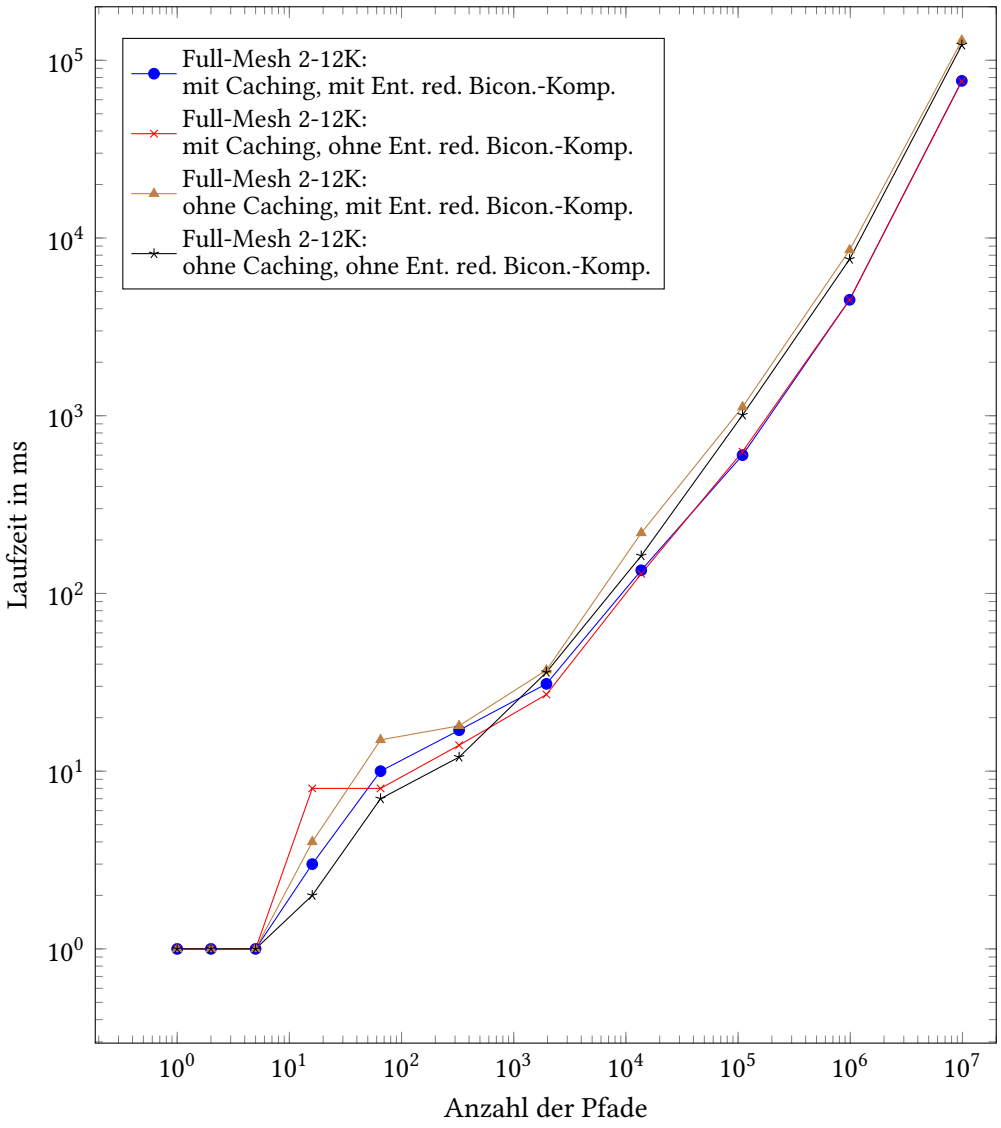


Abbildung 7.7: Verhältnis der Anzahl der Pfade zu Laufzeiten in ms bei Full-Mesh-Graphen mit 2 bis 12 Knoten

7.3.1.3 Ring-Topologie

Abbildung 7.8 zeigt das Verhältnis der Anzahl der Kanten des Ring-Graphen zur Laufzeit in ms, beginnend mit 1000 bis 10000 Knoten. Dabei werden in jedem Schritt 1000 Knoten hinzugefügt. Die Graphen haben jeweils zwei k-Knoten, die einander gegenüberliegen, d. h. einer der k-Knoten ist der erste Knoten des Graphen und der zweite ist der $n/2$ Knoten des Graphen, wobei n der Anzahl der Knoten des Graphen entspricht.

In der Abbildung wird die Auswirkung auf die Laufzeit gezeigt, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob keines der beiden Verfahren angewandt wird. Bei allen drei Varianten wird dieselbe Anzahl an G_{node} erzeugt, d. h. der Algorithmus 5.2 "PathConstruct" wird gleich oft aufgerufen. Bei Ring-Graphen mit zwei k-Knoten hat das Caching keinen Nutzen. Da das Caching allerdings sehr effizient implementiert wurde (siehe Abschnitt 6.3), ist der zusätzliche Rechenaufwand minimal und wirkt sich nicht auf die Laufzeit aus.

Beim Entfernen redundanter Biconnected-Komponenten wird zwar im ersten Schritt die Hälfte der Knoten entfernt (auf dieses Beispiel bezogen), der Rechenaufwand zum Identifizieren der Komponenten wirkt sich jedoch negativ auf die Laufzeit aus, wie die Abbildung zeigt. Grund dafür ist, dass für jeden weiteren Knoten auf dem Pfad zum Zielknoten keine redundanten Biconnected-Komponenten mehr entfernt werden können, der zusätzliche Rechenaufwand allerdings bestehen bleibt.

Im Gegensatz dazu ist der Rechenaufwand beim Entfernen redundanter Knoten geringer. Bei diesem Verfahren (bezogen auf dieses Beispiel) wird in jedem Schritt genau ein redundanter Knoten entfernt. Dies ist jener Knoten, der nur eine Kante besitzt. Somit werden in jedem Schritt auf dem Weg zum Zielknoten zwei Knoten aus dem Graphen entfernt. Dies sind: der redundante Knoten sowie der Knoten, der gerade besucht wurde. Aus diesem Grund ist die Laufzeit für größere Ring-Graphen (siehe Abbildung 7.8) durch das Entfernen der redundanten Knoten geringer, als wenn diese nicht entfernt werden.

7.3.1.4 Stern-Topologie

Abbildung 7.9 zeigt das Verhältnis der Anzahl der Kanten des Stern-Graphen zur Laufzeit in ms, beginnend mit 10001 bis 100001 Knoten. Dabei werden in jedem Schritt 10000 Knoten sowie Kanten hinzugefügt. Die Graphen haben jeweils zwei k-Knoten, die einander gegenüberliegen, d. h. einer der k-Knoten ist der erste Randknoten und der zweite ist der $n/2$ Randknoten, wobei n der Anzahl der Knoten des Graphen entspricht.

In der Abbildung wird die Auswirkung auf die Laufzeit gezeigt, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob keines der beiden Verfahren angewandt wird. Da bei einer Stern-Topologie alle redundanten Biconnected-Komponenten aus nur einem Knoten bestehen, werden bei beiden

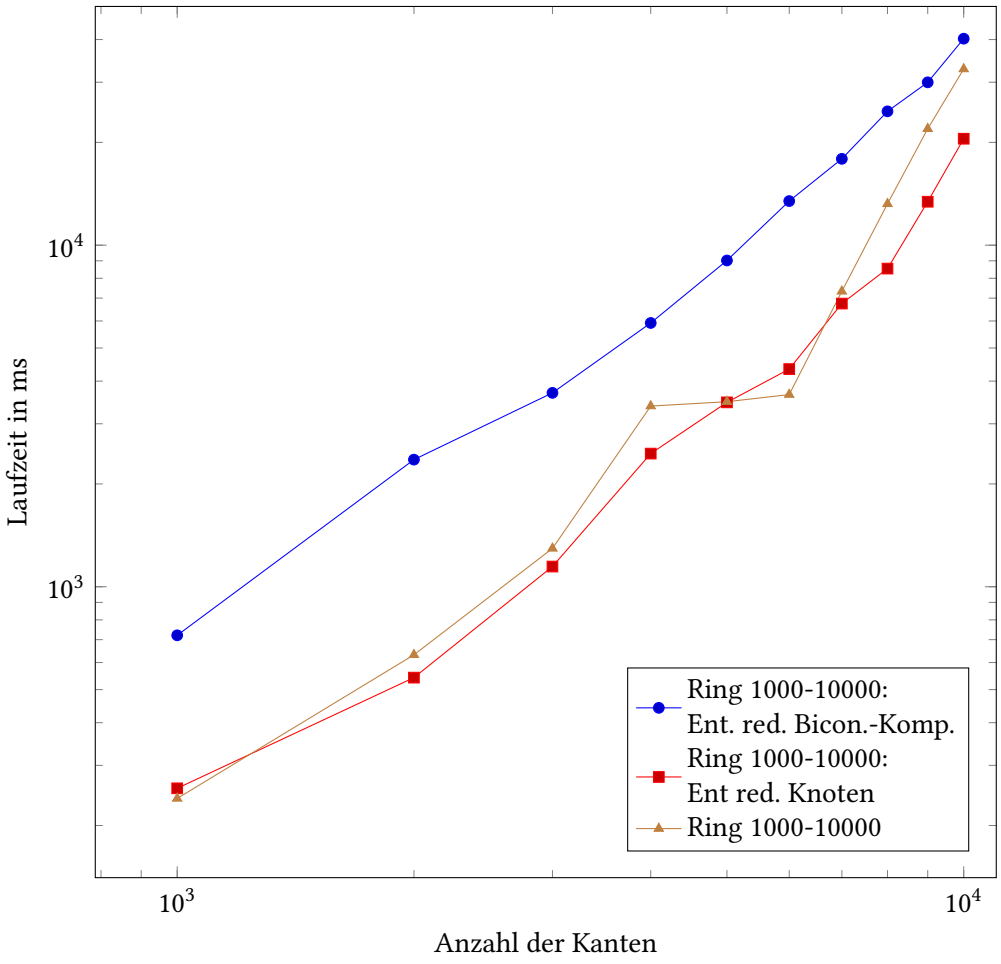


Abbildung 7.8: Auswirkung auf die Laufzeit bei Ring-Graphen mit 1000 bis 10000 Knoten, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob keines der beiden Verfahren angewandt wird. In jedem Schritt werden 1000 Knoten hinzugefügt.

Verfahren im ersten Schritt alle Knoten außer den beiden k-Knoten und den zentralen Knoten entfernt. Das Identifizieren von redundanten Biconnected-Komponenten benötigt jedoch mehr Rechenaufwand als das Identifizieren von redundanten Knoten. Somit ist für eine reine Stern-Topologie das Entfernen redundanter Knoten besser geeignet.

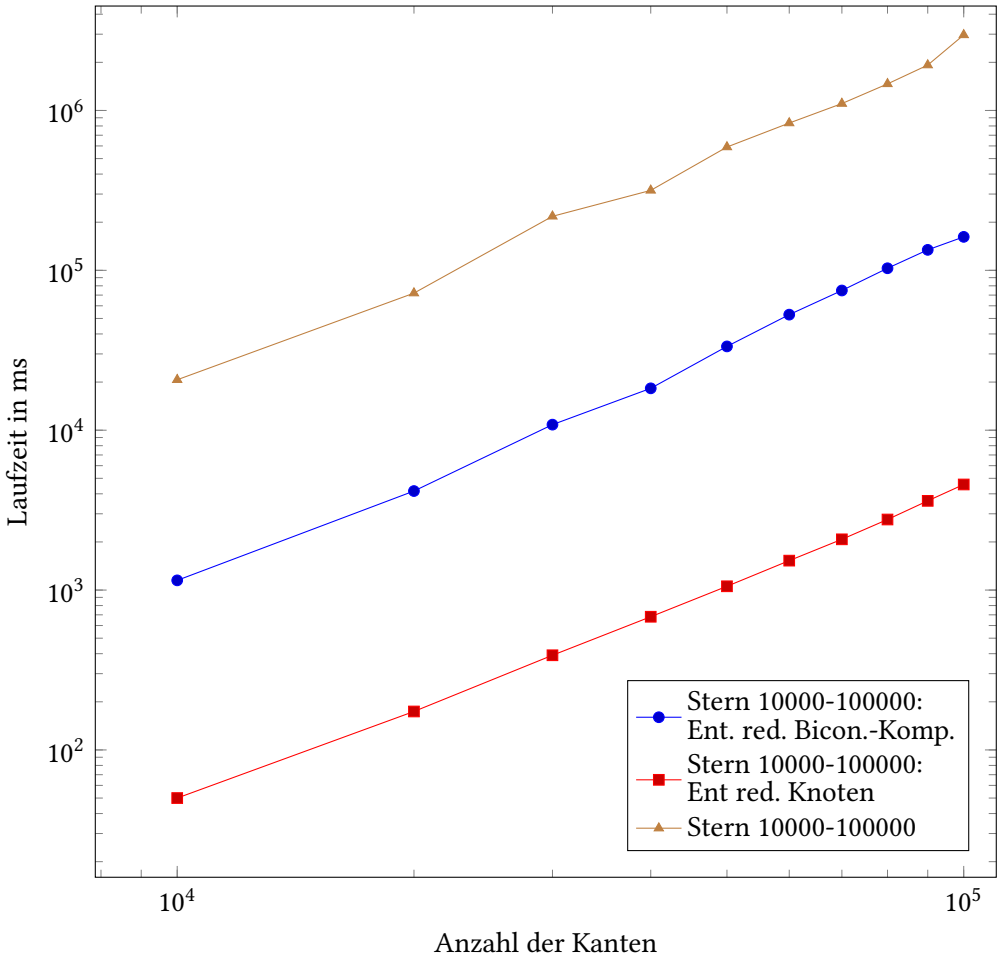


Abbildung 7.9: Auswirkung auf die Laufzeit bei Stern-Graphen mit 10001 bis 100001 Knoten, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob keines der beiden Verfahren angewandt wird. In jedem Schritt werden 10000 Knoten sowie Kanten hinzugefügt.

Abbildung 7.10 zeigt einen Vergleich zwischen der Ring- und der Stern-Topologie. Dabei wird ebenfalls das Verhältnis der Anzahl der Kanten, beginnend bei 1000 bis inkl. 10000, zur Laufzeit in ms dargestellt.

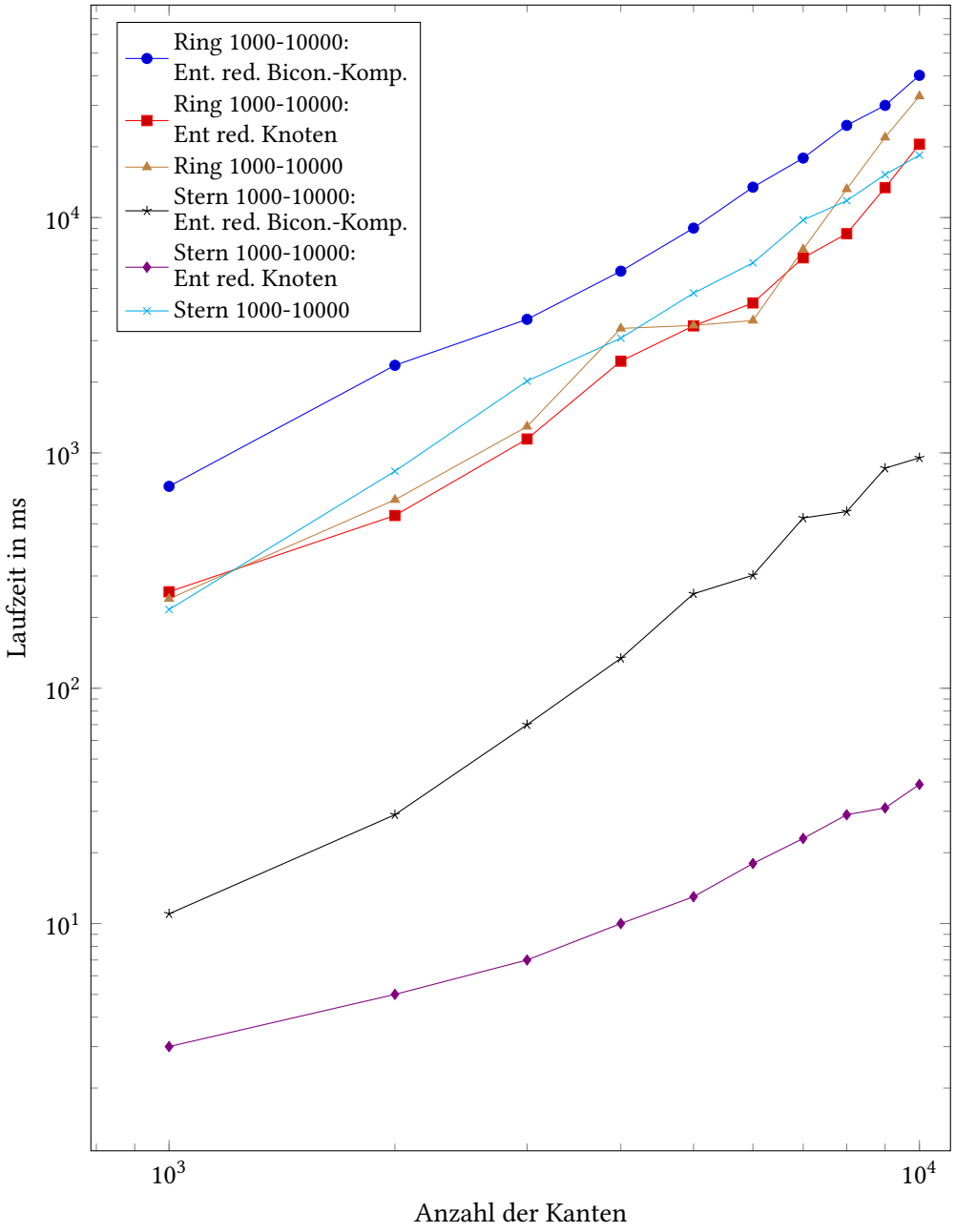


Abbildung 7.10: Auswirkung auf die Laufzeit bei Ring- und Stern-Graphen mit 1000 bis 10000 Kanten, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob keines der beiden Verfahren angewandt wird. In jedem Schritt werden 1000 Kanten hinzugefügt.

7.3.1.5 Netzwerk mit verschiedenen Topologien

In diesem Abschnitt wird das Netzwerk in Abbildung 7.11 evaluiert. Wie die Abbildung zeigt, besteht der Graph aus verschiedenen Einheiten. Die Einheiten "Redundanz", "Verteiler" und "Device" sind hinsichtlich ihrer Anzahl variabel. Wobei die Symmetrie gegeben sein muss, d. h. jeder Knoten des Kerns hat die gleiche Anzahl an Redundanz-Einheiten, jede Redundanz hat die gleiche Anzahl an Verteiler-Einheiten und jeder Verteiler hat die gleiche Anzahl an Devices.

Zusätzlich besitzt der Graph drei k-Knoten, wobei jeder davon ein Device darstellt. Diese drei k-Knoten sind dabei gleichmäßig auf die n Devices verteilt, d. h., dass das erste Device, das $n/3$ Device und das $2n/3$ Device ein k-Knoten darstellt. Der Beispielgraph der Abbildung 7.11 beinhaltet eine Redundanz- und eine Verteiler-Einheit sowie drei Devices.

Abbildung 7.12 stellt eine Evaluierung für diesen Graphen dar, wobei er drei Redundanz-Einheiten pro Kern-Knoten, 5 bis 15 Verteiler-Einheiten pro Redundanz und 50 Devices pro Verteiler besitzt. Dabei wird die Auswirkung auf die Laufzeit mit 5 bis 15 Verteilern gezeigt, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob beide Verfahren gemeinsam angewandt werden oder keines von beiden. Wenn beide Verfahren gemeinsam zur Anwendung kommen, wird das Verfahren zur Entfernung der redundanten Knoten vor der Ausführung des Algorithmus 5.2 "PathConstruct" angewandt, um alle Randknoten zu entfernen, die für die Berechnung der möglichen Pfade nicht benötigt werden. Während der Ausführung von "PathConstruct" wird nur das Verfahren zur Entfernung redundanter Biconnected-Komponenten angewandt.

7.3.2 Vorschlagsystem

In diesem Abschnitt werden verschiedene Teile des Vorschlagsystems evaluiert. Dies betrifft das Identifizieren kritischer Entitäten sowie das Erhöhen und Verringern der Zuverlässigkeit.

7.3.2.1 Identifizieren kritischer Entitäten

Abbildung 7.13 zeigt in einem Diagramm den Unterschied der Laufzeiten des von Yeh et al. [2] entwickelten Algorithmus und dem selbstentwickelten Algorithmus zur Bestimmung der kritischen Entitäten.

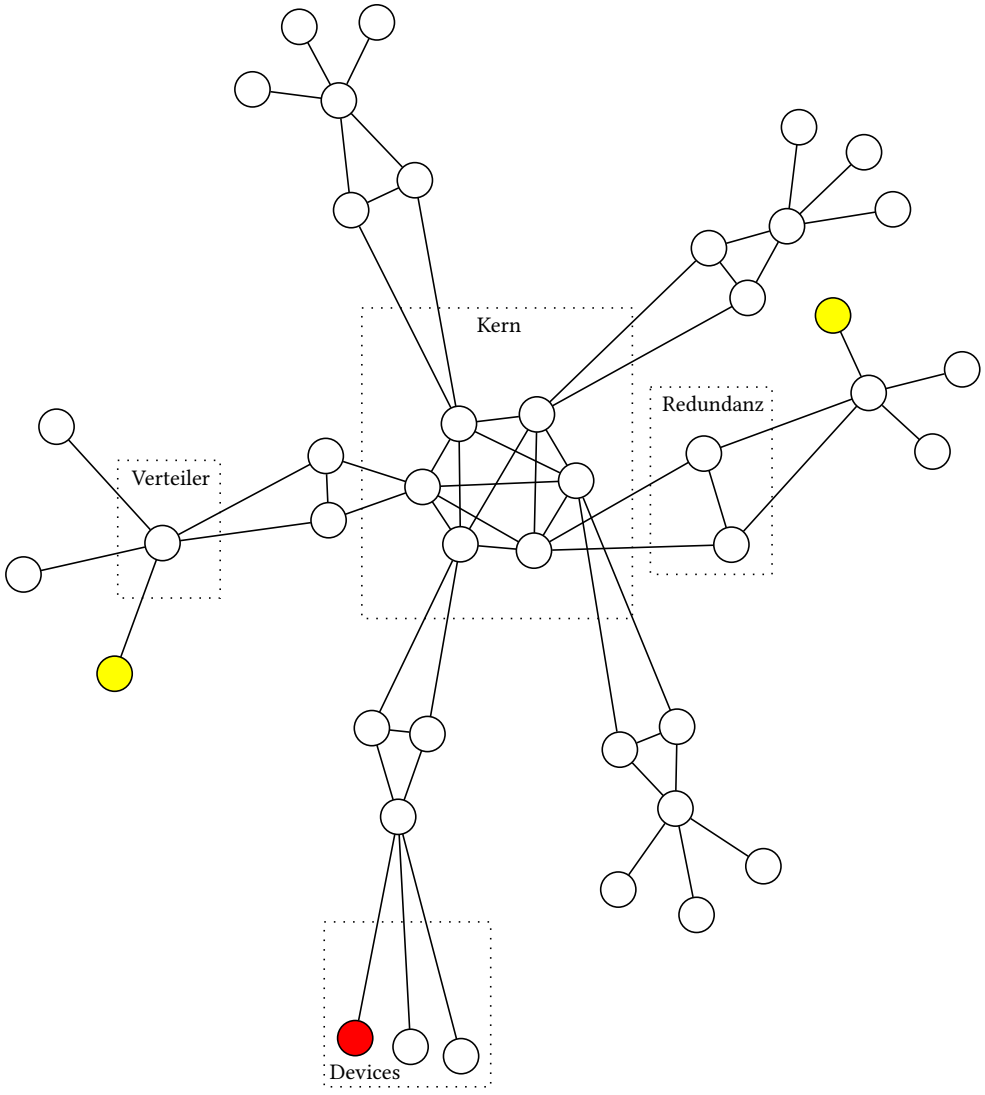


Abbildung 7.11: Netzwerk-Graph, bestehend aus einer Kern-Einheit mit 6 Knoten sowie einer Redundanz-Einheit pro Kern-Knoten. Des Weiteren besitzt der Graph eine Vermittler-Einheit pro Redundanz sowie 3 Devices pro Vermittler. Der Graph beinhaltet drei k-Knoten, davon sind zwei gelb und einer rot markiert. Der rot markierte Knoten ist der, der eine Menge an K-Knoten besitzt.

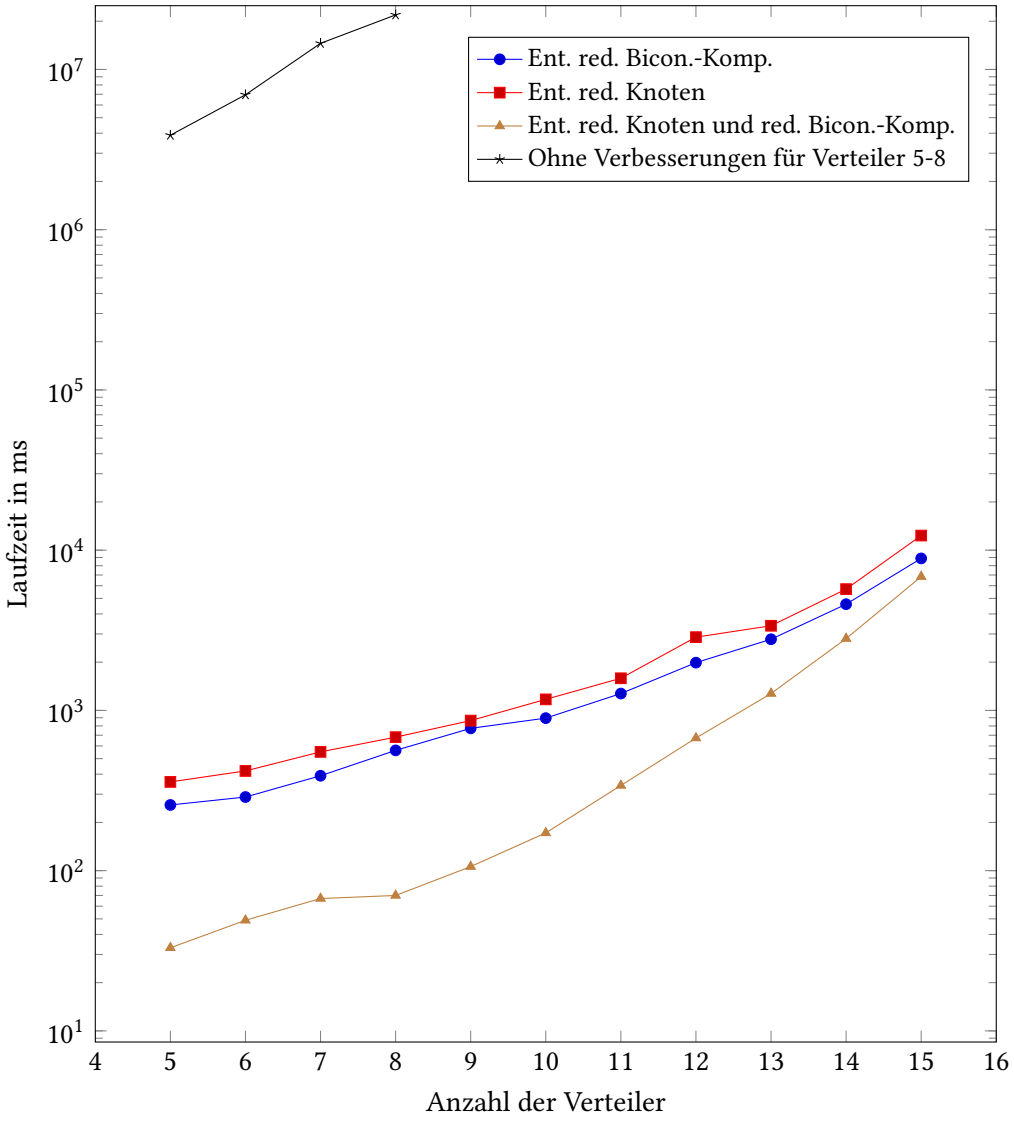


Abbildung 7.12: Auswirkung auf die Laufzeit des Graphen der Abbildung 7.11 mit 5 bis 15 Verteilern, abhängig davon, ob redundante Knoten oder redundante Biconnected-Komponenten entfernt werden, oder ob beide Verfahren gemeinsam angewandt werden oder keines von beiden. Der Graph besitzt drei Redundanz-Einheiten pro Kern-Knoten und 50 Devices pro Verteiler.

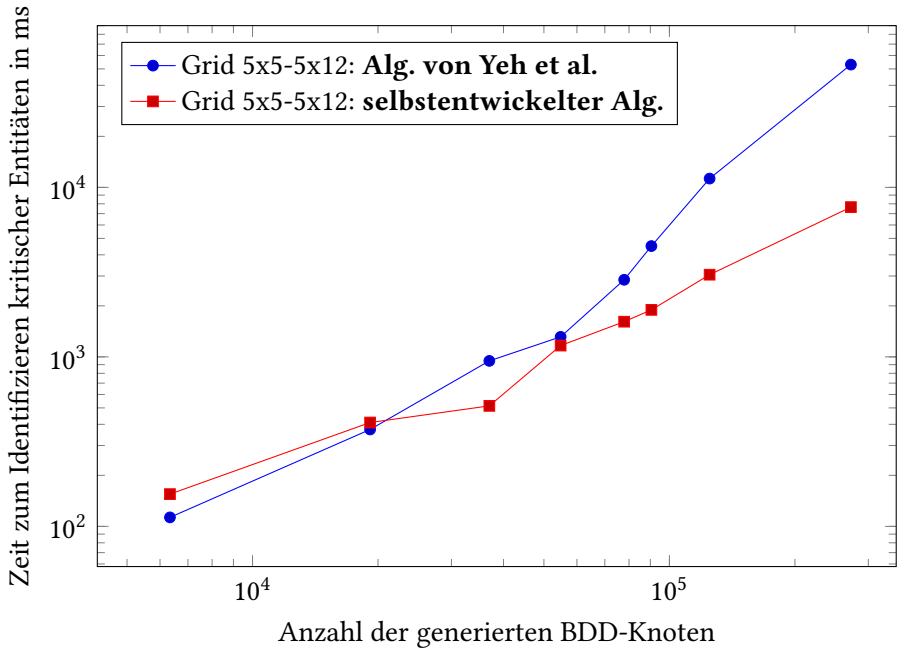


Abbildung 7.13: Vergleich Laufzeiten des Algorithmus zur Bestimmung der kritischen Entitäten von Yeh et al. [2] mit selbstentwickelter Variante

7.3.2.2 Erhöhen und Verringern der Zuverlässigkeit

Die folgenden drei Abbildungen zeigen wie sich ein Netzwerk-Graph durch das Erhöhen sowie anschließende Verringern der Zuverlässigkeit verändert. Abbildung 7.14 stellt dabei den ursprünglichen Graphen dar. Er beinhaltet einen Knoten v_3, der eine Menge an K-Knoten mit v_5, v_9 und v_{14} besitzt. Durch diese vier Knoten wird die Zuverlässigkeit des Netzwerks bestimmt. Angenommen, alle Entitäten außer dem Knoten v_{16} ($p(v_{16}) = 0.99$) haben eine Ausfallsicherheit von $p = 0.9$. Dadurch ergibt sich eine Zuverlässigkeit von 0.458793.

Abbildung 7.15 zeigt den Netzwerk-Graphen, nachdem die Zuverlässigkeit durch Anwendung des Algorithmus 5.10 "ImproveReliability" auf mindestens 0.6 erhöht wurde. Dabei wurden insgesamt 4 Knoten (v_{17}/v'_2, v_{18}/v'_{10}, v_{19}/v'_{16} und v_{20}/v'_7) sowie 16 Kanten hinzugefügt, wobei der Teil der Knotenbezeichnung nach dem "/" (Slash) den Knoten angibt, von dem eine Redundanz erzeugt wurde. Dadurch ergibt sich eine Zuverlässigkeit von 0.607029.

Abbidlung 7.16 zeigt den Netzwerk-Graphen, nachdem die Zuverlässigkeit durch Anwendung des Algorithmus 5.11 "ReduceReliability" auf maximal 0.6 verringert wurde. Dabei wurden insgesamt 3 Knoten (v_0, v_{13} und v_{15}) sowie 8 Kanten entfernt. Dadurch ergibt sich eine Zuverlässigkeit von 0.600823.

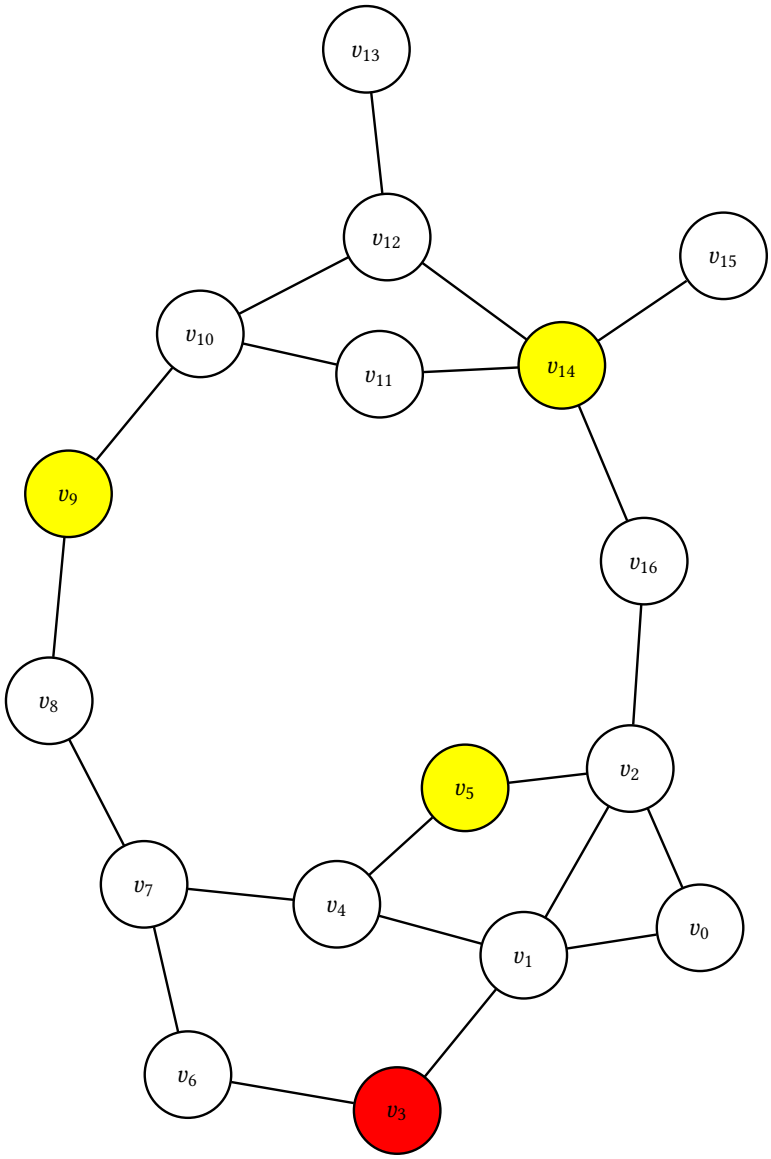


Abbildung 7.14: Netzwerk-Graph mit einem Knoten v_3 (rot markiert), der eine Menge an K-Knoten mit v_5, v_9 und v_{14} (alle gelb markiert) besitzt. Alle Entitäten außer dem Knoten v_{16} ($p(v_{16}) = 0.99$) haben eine Ausfallsicherheit von $p = 0.9$, somit ergibt sich für die Zuverlässigkeit der Wert 0.458793.

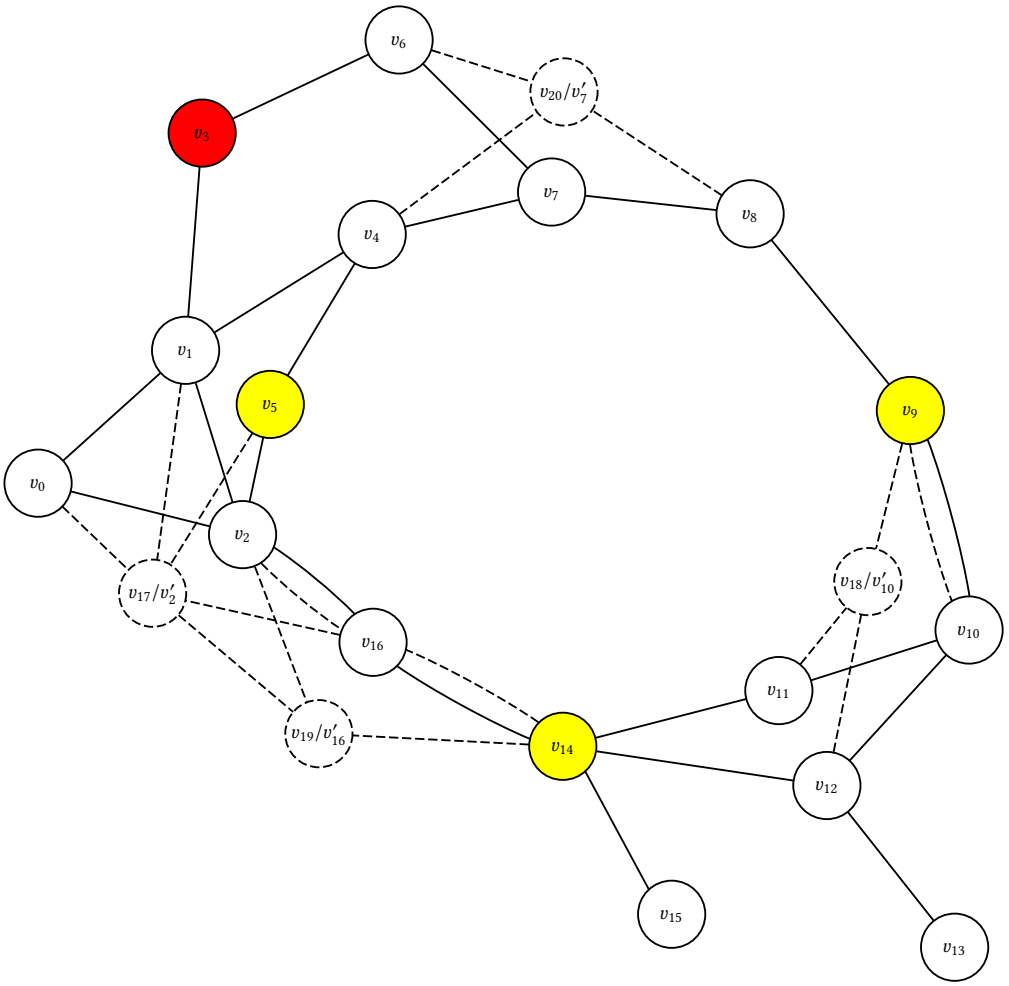


Abbildung 7.15: Netzwerk-Graph der Abbildung 7.14, nachdem die Zuverlässigkeit auf mindestens 0.6 erhöht wurde. Es wurden insgesamt 4 redundante Knoten (v_{17}/v'_2, v_{18}/v'_{10}, v_{19}/v'_{16} und v_{20}/v'_7) sowie 16 Kanten hinzugefügt, wobei der Teil der Knotenbezeichnung nach dem "/" (Slash) den Knoten angibt, von dem eine Redundanz erzeugt wurde. Alle hinzugefügten Entitäten sind strichliert dargestellt. Der neue Wert der Zuverlässigkeit ist 0.607029.

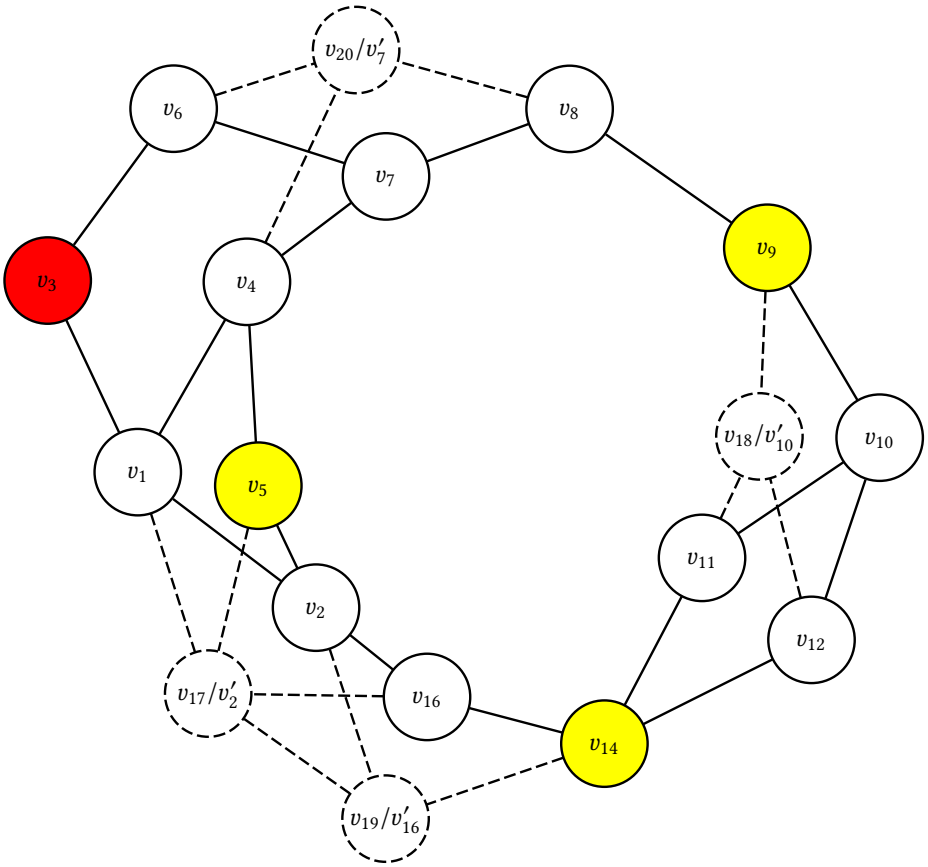


Abbildung 7.16: Netzwerk-Graph der Abbildung 7.15, nachdem die Zuverlässigkeit auf maximal 0.6 reduziert wurde. Dabei wurden insgesamt 3 Knoten (v_0, v_{13} und v_{15}) sowie 8 Kanten entfernt. Die Entitäten, die durch die vorherige Erhöhung der Zuverlässigkeit hinzugefügt wurden, sind strichliert dargestellt. Der neue Wert der Zuverlässigkeit ist 0.600823.

Abbildung 7.17 zeigt die Evaluierung des Näherungsverfahren beim Erhöhen der Zuverlässigkeit. Sie zeigt die Auswirkung auf die Laufzeit, abhängig davon, ob das Näherungsverfahren eingesetzt wird oder nicht. Das dabei verwendete Netzwerk ist ein 3x12 Grid mit zwei k-Knoten (links oben und rechts unten). Die Ausfallsicherheit der einzelnen Entitäten ist $p(x) = 0.9$, somit beträgt die Zuverlässigkeit des Netzwerks 0.579399.

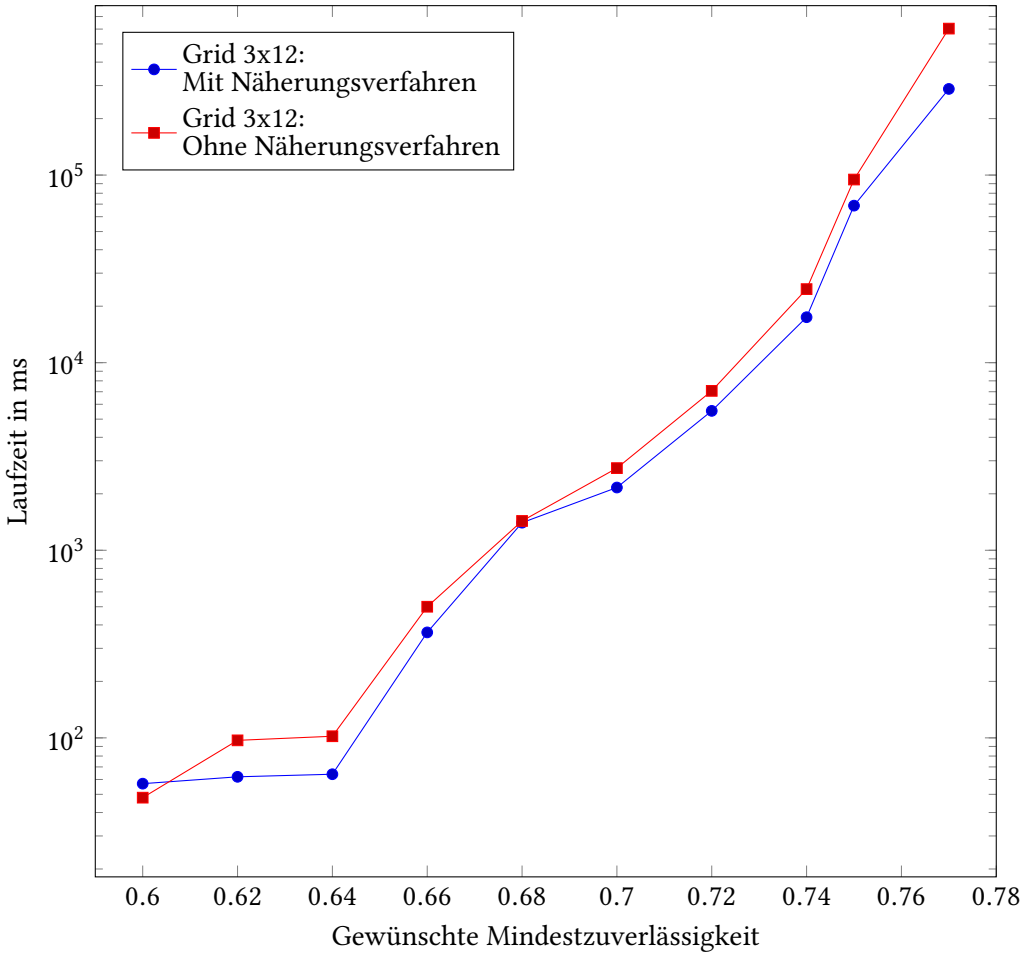


Abbildung 7.17: Verhältnis der gewünschten Mindestzuverlässigkeit zur Laufzeit in ms für ein 3x12 Grid-Netzwerk. Dabei wurde die Zuverlässigkeit einmal mit und einmal ohne Anwendung des Näherungsverfahren auf den gewünschten Wert erhöht. Alle Entitäten haben eine Ausfallsicherheit von $p(x) = 0.9$, wodurch sich eine Zuverlässigkeit von 0.579399 für das Netzwerk ergibt. Die zwei gegenüberliegenden Eckpunkte – links oben und rechts unten – sind die k-Knoten.

Die folgende Evaluierung betrifft den Fehler, der durch das Näherungsverfahren bezüglich der exakten Berechnung der Zuverlässigkeit auftritt. Dieser Fehler ergibt sich aufgrund der errechneten Zuverlässigkeit durch das Näherungsverfahren abzüglich der exakten Zuverlässigkeit. Wie in Punkt 5.4.2 erwähnt, wird dieser Fehler geringer, je höher die Dichte eines Graphen $G = (V, E)$ ist. Die Dichte des Graphen G wird durch

folgende Formel angegeben:

$$d(G) = \frac{|E|}{\binom{|V|}{2}}$$

Dies stellt das Verhältnis der tatsächlichen Kanten zu den maximal möglichen Kanten dar, wobei Mehrfachkanten und Schleifen (verbinden Knoten mit sich selbst) ausgenommen sind.

Der Graph G ist ein zusammenhängender Graph, bestehend aus 7 Knoten und 6 Kanten. Die beiden Randknoten bilden die zwei k-Knoten des Graphen. Nun soll in jedem Schritt eine Redundanz eines Knotens erstellt und der Fehler des Näherungsverfahrens bestimmt werden. Weiterhin wird in jedem Schritt eine Kante zwischen zwei zufällig ausgewählten Knoten (ohne Mehrfachknoten und ohne Schleifen) hinzugefügt, um die Dichte des Graphen zu erhöhen, bis diese den Wert 1 erreicht. Im konkreten Beispiel werden dem Graphen G so lange Kanten hinzugefügt, bis dieser 21 Kanten besitzt und somit eine Full-Mesh-Topologie aufweist.

Abbildung 7.18 zeigt das Verhältnis der Dichte des Graphen G mit 7 Knoten und 6 bis 21 Kanten zum Fehler des Näherungsverfahren. In der Abbildung werden drei Funktionen der erstellen Redundanz unterschieden. In der ersten Funktion (Kreis-Symbol) wird von jenem Knoten eine Redundanz erstellt, der die meisten Kanten besitzt. In der zweiten Funktion (Quadrat-Symbol) wird von dem Knoten eine Redundanz erstellt, der die wenigsten Kanten besitzt. In der dritten Funktion (Dreieck-Symbol) wird jeweils von dem Knoten eine Redundanz erstellt, der bezüglich Zuverlässigkeit am kritischsten ist. Wie Abbildung 7.18 zeigt, nimmt der Fehler des Näherungsverfahrens mit zunehmender Dichte des Graphen ab.

Da die zwei Knoten, die durch eine Kante eine Verbindung erhalten, in jedem Schritt zufällig ausgewählt werden, wurde die Evaluierung für jede der drei Funktionen 100-mal durchgeführt und der jeweilige Mittelwert errechnet. Diese Mittelwerte sind die Datengrundlage für die Abbildung.

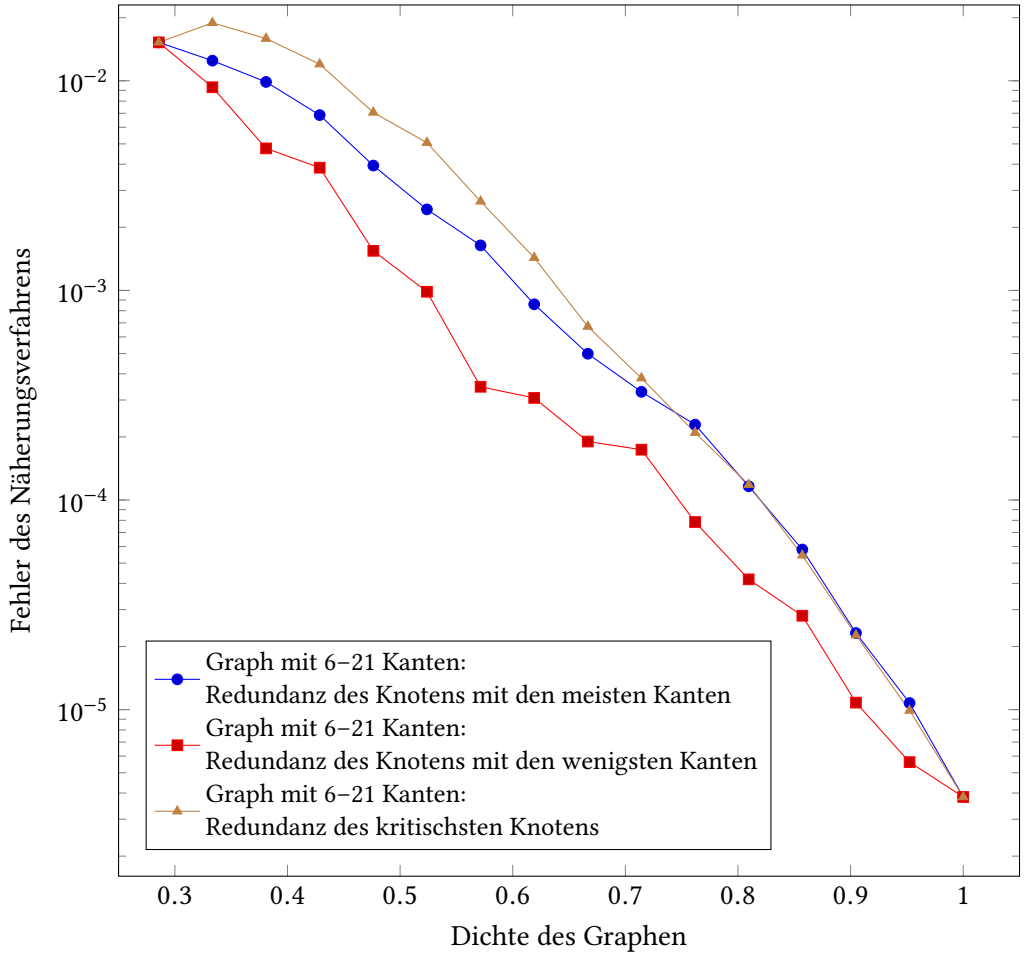


Abbildung 7.18: Verhältnis der Dichte eines Graphen G mit 7 Knoten und 6–21 Kanten zum Fehler des Näherungsverfahrens. In jedem Schritt wird eine Kante zwischen zwei zufällig ausgewählten Knoten hinzugefügt. Aus diesem Grund entsprechen die Daten der Funktionen dem Mittelwert aus 100 Durchläufen.

Kapitel 8

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Algorithmus entwickelt, um die Zuverlässigkeit von k-Terminals eines Netzwerks bestimmen zu können. Da das zugrundeliegende Problem NP-schwer [5, 6] ist, wurde ein effizienter Lösungsansatz [2] gewählt. Dabei wurde die Lösung entsprechend adaptiert, damit mögliche Redundanzen der k-Terminals für die Berechnung der Zuverlässigkeit berücksichtigt werden.

Das Netzwerk wird durch ein Graph-Modell mit statischen Eigenschaften abstrahiert, in dem sowohl redundante Netzentitäten als auch Hierarchien zwischen Komponenten abgebildet werden können. Zusätzlich ermöglicht das Modell eine auf Komponenten basierte Berechnung der Zuverlässigkeit, wodurch für jede Komponente separat k-Terminals angegeben werden können, um deren Zuverlässigkeit zu bestimmen.

Des Weiteren wurde ein Vorschlagsystem vorgestellt, das aufbauend auf der berechneten Zuverlässigkeit von k-Terminals verschiedene Algorithmen bereitstellt. Einer dieser Algorithmen dient dem Identifizieren kritischer Netzentitäten. Dadurch können Entitäten aufgrund ihrer Relevanz für die Zuverlässigkeit der k-Terminals bewertet werden. Dabei wird die kritischste Entität als jene definiert, bei deren Ausfall sich die Zuverlässigkeit um den höchsten Wert verringert.

Auf dieser Bewertungsmaßnahme der Netzentitäten basieren zwei weitere Algorithmen des Vorschlagsystems. Durch diese kann die Zuverlässigkeit von k-Terminals sowohl durch das Einfügen redundanter Entitäten erhöht als auch durch das Entfernen von Entitäten verringert werden. Für das Erhöhen der Zuverlässigkeit wurde ein Näherungsverfahren entwickelt, um einen Teil der aufwendigen Berechnung zu umgehen.

Ein weiterer Algorithmus, der vorgestellt wurde, unterbreitet dem Nutzer mehrere Vorschläge für die Positionierung einer Redundanz für eines der k-Terminals im Netzwerk. Dabei wird für jede mögliche Positionierung der Redundanz die Zuverlässigkeit bestimmt und in einer Liste zurückgegeben. Dadurch kann die Zuverlässigkeit beim Einfügen einer Redundanz maximiert werden.

Die Algorithmen des Vorschlagsystems können sinnvoll miteinander kombiniert werden. Wenn z. B. die Zuverlässigkeit eines Netzwerks für k-Terminals erhöht werden soll, kann folgende Kombination ausgeführt werden:

1. Berechnung der Zuverlässigkeit, um herauszufinden, ob diese überhaupt erhöht werden muss. Falls nicht, kann abgebrochen werden.

2. Zu Beginn des Algorithmus 5.10 "ImproveReliability" wird überprüft, ob die k-Terminals mindestens die gewünschte Zuverlässigkeit aufweisen. Ist dies nicht der Fall, müssen so viele Redundanzen der jeweiligen Komponente erzeugt werden, bis dies zutrifft. Für die jeweilige Positionierung der Redundanz kann der Algorithmus 5.13 "FindPlacesForRedundancy" ausgeführt werden, um den Anstieg der Zuverlässigkeit zu maximieren. Durch zusätzliche Angaben könnte vermieden werden, dass Positionen ausgewählt werden, an denen die Terminals nicht angeschlossen werden dürfen.

3. Anschließend werden durch den Algorithmus 5.9 "CalcEV" die kritischen Entitäten bestimmt und die Zuverlässigkeit durch Hinzufügen von Redundanzen erhöht.

4. Zum Abschluss kann der Algorithmus 5.11 "ReduceReliability" aufgerufen werden, um Entitäten zu entfernen, die für die Zuverlässigkeit keine Relevanz aufweisen.

Die Implementierung und Evaluierung der vorgestellten Algorithmen wurde in der Arbeit ebenfalls beschrieben. Dabei wurde eine qualitative (bezogen auf die Anforderungen an das System) und quantitative Evaluierung sowie eine theoretische Komplexitätsanalyse für die entwickelten Algorithmen durchgeführt.

Die Betrachtung möglicher zukünftiger Arbeiten wird in zwei Kategorien unterteilt. Die erste bezieht sich auf Laufzeitverbesserung und -vergleiche, die zweite auf Erweiterungsmöglichkeiten für das aufgestellte Graph-Modell.

Laufzeit
Zukünftige Arbeiten sollten sich mit dem Entwickeln einer besser geeigneten Heuristik für die Variablen-Ordnung der BDDs beschäftigen, da die Größe eines BDDs sehr stark von dessen Variablen-Ordnung abhängt. Dadurch könnte die Laufzeit des Algorithmus zur Bestimmung der Zuverlässigkeit reduziert werden.

Es könnte unter Berücksichtigung von Redundanzen ein Laufzeit-Vergleich vorgenommen werden zwischen der in dieser Thesis entwickelten Methode zur Bestimmung der Zuverlässigkeit von k-Terminals und der auf Hardy et al. [31] basierenden Decomposition-Methode von Lê [1]. Unter diesem Gesichtspunkt könnte ein Algorithmus zur Bestimmung der Zuverlässigkeit von k-Terminals inkl. der Redundanzen entwickelt werden, der auf dem Algorithmus "SIMPATH" von Knuth [43, exercise 225, ch. 7.1.4] basiert und anstelle der Datenstruktur BDD die Datenstruktur ZBDD verwendet.

Graph-Modell
In zukünftigen Arbeiten könnte eine Betrachtung der Kosten der Netzentitäten vorgenommen werden. Dies würde das Identifizieren kritischer Entitäten, das Einfügen redundanter Entitäten, um die Zuverlässigkeit zu erhöhen, und das Entfernen von Netzentitäten, um die Zuverlässigkeit zu verringern, betreffen.

Eine mögliche Erweiterung für das Modell wäre die Abbildung geografischer Gegebenheiten des Netzwerks oder die Berücksichtigung von Beschränkungen der Konnektivität von Komponenten aufgrund von Routern oder Firewalls.

Des Weiteren wäre eine dynamische Betrachtung des Systems interessant. Dazu könnte eine Verfügbarkeitsanalyse durchgeführt und die Reparaturzeiten der einzelnen Komponenten berücksichtigt werden.

Literaturverzeichnis

[1] M. Lê, "Quantitative evaluation of network reliability," Ph.D. dissertation, Technische Universität München, Dec. 2013.

[2] S.-Y. Kuo, F.-M. Yeh, and H. Lin, "Efficient and Exact Reliability Evaluation for Networks With Imperfect Vertices," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 288–300, Jun. 2007.

[3] F.-M. Yeh, S. Lu, and S.-Y. Kuo, "Determining Terminal-Pair Reliability Based On Edge Expansion Diagrams Using OBDD," *IEEE Transactions on Reliability*, vol. 48, no. 3, pp. 234–246, Sep. 1999.

[4] Institute of Electrical & Electronics Engineers, *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries: 610.* IEEE Publications,U.S., Jan. 1991.

[5] L. G. Valiant, "The Complexity of Enumeration and Reliability Problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, Aug. 1979.

[6] M. O. Ball, "Computational Complexity of Network Reliability Analysis: An Overview," *IEEE Transactions on Reliability*, vol. 35, no. 3, pp. 230–239, Aug. 1986.

[7] Z. Ma, "Towards a Unified Definition for Reliability, Survivability and Resilience (I): the Conceptual Framework Inspired by the Handicap Principle and Ecological Stability," *Aerospace Conference, IEEE*, pp. 1–12, Mar. 2010.

[8] M. L. Shooman, *Reliability of Computer Systems and Networks*, 1st ed. John Wiley & Sons, Feb. 2002.

[9] A. Mili, L. Wu, F. Sheldon, M. Shereshevsky, and J. Desharnais, "Modeling Redundancy: Quantitative and Qualitative Models," *IEEE International Conference on Computer Systems and Applications*, pp. 1–8, Mar. 2006.

[10] V. Ebrahimipour, M. Sheikhalishahi, B. Maleki Shoja, and M. Goldansaz, "A Universal Generating Function Approach for Redundancy Optimization for Hot-Standby Multi-State Series-Parallel k-out-of-n Systems," *Fourth UKSim European Symposium on Computer Modeling and Simulation (EMS)*, pp. 235–239, Nov. 2010.

[11] H. R. Andersen, "An Introduction to Binary Decision Diagrams," Lecture notes for Efficient Algorithms and Programs at the IT University of Copenhagen, 1997.

[12] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[13] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, 1st ed. Addison-Wesley, Jan. 1974.

[14] D. Sieling and I. Wegener, "Reduction of OBDDs in linear time," *Information Processing Letters*, vol. 48, no. 3, pp. 139–144, Nov. 1993.

[15] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," *IEEE Transactions on Computers*, vol. 39, no. 5, pp. 710–713, May 1990.

[16] R. E. Barlow and F. Proschan, *Mathematical Theory of Reliability*. SIAM, 1996, reprint of Et: John Wiley & Sons, 1965.

[17] E. S. Elmallah, "Algorithms for K-Terminal Reliability Problems with Node Failures," *Networks*, vol. 22, no. 4, pp. 369–384, Jul. 1992.

[18] J. S. Provan and M. O. Ball, "The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 777–788, 1983.

[19] J. N. Ayoub, W. H. Saafin, and B. Z. Kahhaleh, "k-Terminal Reliability of Communication Networks," *The 7th IEEE International Conference on Electronics, Circuits and Systems*, vol. 1, pp. 374–377, Dec. 2000.

[20] M. Lê, J. Weidendorfer, and M. Walter, "A Novel Variable Ordering Heuristic for BDD-based k-terminal Reliability," *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 527–537, Jun. 2014.

[21] J. M. Wilson, "An Improved Minimizing Algorithm for Sum of Disjoint Products," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 42–45, Apr. 1990.

[22] S. Soh and S. Rai, "Experimental Results on Preprocessing of Path/Cut Terms in Sum of Disjoint Products Technique," *IEEE Transactions on Reliability*, vol. 42, no. 1, pp. 24–33, Mar. 1993.

[23] P. A. Jensen and M. Bellmore, "An Algorithm to Determine the Reliability of a Complex System," *IEEE Transactions on Reliability*, vol. R-18, no. 4, pp. 169–174, Nov. 1969.

[24] K. K. Aggarwal, Y. C. Chopra, and J. S. Bajwa, "Modification of cutsets for reliability evaluation of communication systems," *Microelectronics Reliability*, vol. 22, no. 3, pp. 337–340, 1982.

[25] Y. G. Chen and M. C. Yuang, "A Cut-Based Method for Terminal-Pair Reliability," *IEEE Transactions on Reliability*, vol. 45, no. 3, pp. 413–416, Sep. 1996.

[26] F. Moskowitz, "The Analysis of Redundancy Networks," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 77, no. 5, pp. 627–632, Nov. 1958.

[27] A. Satyanarayana and M. K. Chang, "Network Reliability and the Factoring Theorem," *John Wiley & Sons*, vol. 13, no. 1, pp. 107–120, 1983.

[28] R. K. Wood, "A factoring algorithm using polygon-to-chain reductions for computing K-terminal network reliability," *Networks*, vol. 15, no. 2, pp. 173–190, Nov. 1985.

[29] A. Satyanarayana and R. K. Wood, "A Linear-Time Algorithm for Computing K-Terminal Reliability in Series-Parallel Networks," *SIAM Journal on Computing*, vol. 14, no. 4, pp. 818–832, 1985.

[30] O. R. Theologou and J. G. Carlier, "Factoring & Reductions for Networks with Imperfect Vertices," *IEEE Transactions on Reliability*, vol. 40, no. 2, pp. 210–217, Jun. 1991.

[31] G. Hardy, C. Lucet, and N. Limnios, "K-Terminal Network Reliability Measures With Binary Decision Diagrams," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 506–515, Sep. 2007.

[32] S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, Jun. 1978.

[33] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 293–318, Sep. 1992.

[34] F.-M. Yeh, S. Lu, and S.-Y. Kuo, "OBDD-Based Evaluation of k-Terminal Network Reliability," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 443–451, Dec. 2002.

[35] J. Carlier and C. Lucet, "A decomposition algorithm for network reliability evaluation," *Discrete Applied Mathematics*, vol. 65, no. 1, pp. 141–156, 1996.

[36] M. Lê, M. Walter, and J. Weidendorfer, "Improving the Kuo-Lu-Yeh algorithm for assessing Two-Terminal Reliability," *Tenth European Dependable Computing Conference (EDCC)*, pp. 13–22, May 2014.

[37] J. U. Herrmann and S. Soh, "A Memory Efficient Algorithm for Network Reliability," *15th Asia-Pacific Conference on Communications (APCC)*, pp. 703–707, Oct. 2009.

[38] J. U. Herrmann and S. Soh, "Improving Reliability Calculation with Augmented Binary Decision Diagrams," *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 328–333, Apr. 2010.

[39] M. Sprotte, "Vergleichende Untersuchung der Zuverlässigkeit und Resilienz verschiedener Netztopologien," Master's thesis, Universität Hamburg, FB Informatik, 2014.

[40] J. Abraham, "An Improved Algorithm for Network Reliability," *IEEE Transactions on Reliability*, vol. R-28, no. 1, pp. 58–61, Apr. 1970.

[41] K. Heidtmann, "Statistical Comparison of Two Sum-of-Disjoint-Product Algorithms for Reliability and Safety Evaluation," *Computer Safety, Reliability and Security Lecture Notes in Computer Science*, vol. 2434, pp. 70–81, Sep. 2002.

[42] H. Iwashita, J. Kawahara, and S.-I. Minato, "ZDD-Based Computation of the Number of Paths in a Graph," Hokkaido University – Division of Computer Science, Tech. Rep., Sep. 2012.

[43] D. E. Knuth, *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1*, 1st ed. Addison-Wesley, 2011.

[44] Y.-K. Lin, "Reliability Evaluation for an Information Network With Node Failure Under Cost Constraint," *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 37, no. 2, pp. 180–188, Mar. 2007.

[45] D. Hock, M. Hartmann, C. Schwartz, and M. Menth, "ResiLyzer: Ein Werkzeug zur analyse der Ausfallsicherheit in paketvermittelten Kommunikationsnetzen," *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, vol. 34, no. 3, 2011.

[46] D. Hock, M. Menth, M. Hartmann, C. Schwartz, and D. Stezenbach, "ResiLyzer: A Tool for Resilience Analysis in Packet-Switched Communication Networks," *15th International GI/ITG Conference, MMB&DFT*, vol. 5987, pp. 302–306, Mar. 2010.

[47] M. Menth, M. Duelli, R. Martin, and J. Milbrandt, "Resilience Analysis of Packet-Switched Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 6, pp. 1950–1963, Dec. 2009.

[48] G. Hardy, C. Lucet, and N. Limnios, "A BDD-Based Heuristic Algorithm for Design of Reliable Networks with Minimal Cost," *Mobile Ad-hoc and Sensor Networks*, vol. 4325, pp. 244–255, 2006.

[49] M.-L. Rebaiaia and D. Ait-Kadi, "Network Reliability Evaluation and Optimization: Method, Algorithms and Software Tools," *CIRRELT*, Dec. 2013.

[50] M. Abd-El-Barr, A. Zakir, S. M. Sait, and A. Almulhem, "Reliability and Fault Tolerance based Topological Optimization of Computer Networks - Part I: Enumerative Techniques," *IEEE Pacific Rim Conference on Communications, Computers and signal Processing (PACRIM)*, vol. 2, pp. 732–735, Aug. 2003.

[51] B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "Topology Design with Minimal Cost Subject to Network Reliability Constraint," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 118–131, Mar. 2015.

[52] B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "Dynamic Programming Algorithm for Reliable Network Design," *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 443–454, Jun. 2014.

[53] J. Hopcroft and R. Tarjan, "Efficient Algorithms for Graph Manipulation," *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973.

[54] D. Jungnickel, *Graphs, Networks and Algorithms*, 4th ed. Springer, Nov. 2012.

[55] J. Whaley, "JavaBDD," Oct. 2007, aufgerufen am 14. Juli 2015. [Online]. Available: http://javabdd.sourceforge.net/