Masterarbeit
Software Engineering
Sommersemester 2013

Masterarbeit

Berechnung einer optimalen Strategie in allgemeinen Risikoanalysen mit ARA

Marcel Michel

30. Dezember 2013

Techische Universität Dortmund
Fakultät Informatik
Lehrstuhl 14 Software Engineering – Prof. Dr. Jan Jürjens
betreut durch: Dipl.-Inform. Andreas Schmitz

Marcel Michel
marcel.michel@udo.edu.de
Matrikelnummer: 126953
Studiengang: Master Informatik

Seconomics - Socio-economics meets Security
Thema: Berechnung einer optimalen Strategie in allgemeinen Risikoanalysen mit ARA

Eingereicht: 30. Dezember 2013

Betreuer: Dipl.-Inform. Andreas Schmitz

Prof. Dr. Jan Jürjens Lehrstuhl 14 Software Engineering
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dortmund, den 30. Dezember 2013

Marcel Michel

Kurzbeschreibung

Die allgemeine Risikoanalyse ist im heutigen Zeitalter ein unerlässliches Werkzeug um Risiken verschiedenster Art zu identifizieren und zu bewerten, um anschließend einen Präventionsplan entwickeln zu können. Allerdings sieht die klassische Risikoanalyse konstante und nicht veränderbare Einflussfaktoren vor, welche im Zusammenhang mit intelligenten Gegenspielern an ihre Grenzen stößt. In Kontrast dazu steht die Spieltheorie, mit der Grundidee, gegen anpassbare, strategisch optimierende Gegenspieler beziehungsweise Agenten zu agieren. Leider setzten die verschiedenen Lösungsansätze innerhalb der Spieltheorie meist ein vollständiges Wissen über den Angreifer voraus. Diese Grundannahme über ein vollständiges Wissen ist allerdings realitätsfern. Um diesen gravierenden Nachteil auszugleichen, erlaubt die Adversarial Risk Analysis (ARA) Unsicherheiten über einzelne Agenten zu modellieren und diese bei der Berechnung von konkreten Entscheidungen einfließen zu lassen. Die bisher entwickelten ARA-Instanzen werden zurzeit manuell gelöst. Eine allgemeine Berechnungsvorschrift, um eine optimale Strategie innerhalb dieser Modelle zu finden, ist daher ein sehr erstrebenswertes Ziel.

Abstract

Nowadays risk analysis plays an important role in identifying and evaluating different types of vulnerabilities. The results of a risk analysis are used to develop adequate prevention strategies. However there is a serious weakness in classic risk analysis methodology. The classic version uses constant and unchangeable impact factors, which could result in aberrant model behaviour against intelligent opponents. On the other hand is game theory, wich takes into account the concept of intelligent, strategically interacting agents. Unfortunately, many concepts in this area require a common knowledge about the game and therefore about the other players. In most cases, this concept of the common knowledge is not applicable due to a lack of information about the other actors. To address this shortcoming, the Adversarial Risk Analysis (ARA) framework was designed to evaluate model with uncertainties that characterize the approximate knowledge of several actors. At the moment these models are solved manually. There are no general algorithms to compute an optimal strategy for the models. All in all, the construction of an algorithm that can successfully compute this strategy is desirable.

Vorwort und Danksagung

Diese Masterarbeit ist im Rahmen des Seconomics Projekt entstanden. Ziel dieses Projektes ist es, aktuellste Technologien im Bereich der Risikoanalyse weiterzuentwickeln, um diese im Anschluss auf verschiedene Fallstudien anwenden zu können. Dabei werden Sicherheitsprobleme von öffentlichen Transportmitteln, wie auch von kritischen Sicherheitsstrukturen, durchleuchtet. Das Endresultat einer solchen Risikoanalyse bildet stets eine optimale Verhaltensstrategie.

An dieser Stelle möchte ich mich recht herzlich bei allen Mitarbeitern des Lehrstuhles 14 für Informatik der TU Dortmund sowie dem Fraunhofer ISST, die mich während meiner Masterarbeit unterstützt haben, bedanken. Mein Dank gebührt auch meiner Familie, meinen Kommilitonen und Freunden, die mir während des Studiums stets Rückhalt gegeben haben.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird die Motivation und der Hintergrund (Abschnitt 1.1) dieser Masterarbeit vorgestellt. In Abschnitt 1.1.1 wird ein einführendes Beispiel betrachtet, welches eine konkrete Anwendung zeigt. Anschließend werden die Ziele in Abschnitt 1.2 beschrieben, wie auch eine Übersicht über den Verlauf dieser Arbeit gegeben (Abschnitt 1.3).

1.1 Motivation und Hintergrund

Viele Länder besitzen *kritische Infrastrukturen*, dessen Störungen enorme Auswirkungen auf das Gleichgewicht eines Landes haben können [Ref09]. Solche Infrastrukturen lassen sich in zwei Bereiche unterteilen. Die *technische Infrastruktur*, wie die allgemeine Energieversorgung oder die Telekommunikation, zum anderen die *soziale Infrastruktur*, welche als Beispiel das Gesundheitssystem, wie Krankenhäuser oder das Bildungssystem mit Schulen und Universitäten umfasst [PKP13].
Diese wichtigen Strukturen sind potentiell durch Terroristen oder andere intelligente Teilnehmer angreifbar. Die Entwicklung und Konzeption eines wirkungsvollen Schutzmechanismus ist daher unerlässlich. Die Erforschung eines solchen Ansatzes wurden in Anbetracht von Anschlägen, insbesondere durch den des 11. September 2001, immer mehr in den Fokus gerückt [BCA09].

Die klassische *Zuverlässigkeits-* und *Risikoanalyse* spielen dabei eine wichtige Rolle. Diese werden dazu eingesetzt, verschiedenste Risiken zu identifizieren und zu bewerten, um anschließend einen möglichst guten *Präventationsplan* zu entwickeln. Allerdings sehen diese beiden Ansätze in ihrer Urform konstante und nicht veränderbare Einflussfaktoren vor. In Kontrast dazu steht die *Spieltheorie*, welche die Grundidee beinhaltet, gegen anpassbare, strategisch optimierende Gegenspieler beziehungsweise *Agenten*, zu agieren. Das Vorgehen der einzelnen Agenten ist dadurch sehr *rational* und somit kann ein sehr realitätsnahes Verhalten simuliert werden. Leider setzten die verschiedenen Lösungsansätze, um solche *Strategieprofile* für die einzelnen Agenten innerhalb der Spieltheorie zu entwickeln, meist ein vollständiges Wissen über den Angreifer voraus. Ein vollständiges Wissen über den Angreifer ist allerdings realitätsfern [BCA09].

Die Kombination der klassischen Risikoanalyse und der Spieltheorie birgt viel Potential, da beide Bereiche schon eine geraume Zeit erforscht werden und daher viele verschiedene Lösungsansätze auf den unterschiedlichsten Gebieten existieren. Dieses Hintergrundwissen kann dazu genutzt werden, einen realitätsnäheren Ansatz

zu entwickeln, sodass Risiken in Gegenwart von intelligenten Gegenspielern leichter erkannt und gezielt durch *Gegenmaßnahmen* minimiert werden können. Die Modellierung des *Angreifers* spielt dabei eine entscheidende Rolle, da Risiken beziehungsweise *Verhaltensmuster* auf der Grundlage dieser Modellierung identifiziert werden. Dadurch ist es enorm wichtig, das Wissen beziehungsweise das Unwissen über den Angreifer geschickt einfließen zu lassen, um so eine sehr realitätsnahe Modellierung zu erhalten.

1.1.1 Einführendes Beispiel

Zunächst wird ein einführendes Beispiel betrachtet. Dabei kann die Problemsituation aus der vorherigen Motivation verwendet werden. Die Regierung besitzt kritische Infrastrukturen oder Systeme, welche es vor terroristischen Gruppen zu schützen gilt. Diese Situation ist vereinfacht in Abbildung 1.1 dargestellt. Das Modell zeigt zwei Spieler, deren Elemente farblich unterschiedlich hervorgehoben sind. Die Regierung fungiert als Verteidiger (`Defender`), wohingegen die Terroristen als Angreifer (`Attacker`) dargestellt werden. Beide versuchen die Infrastruktur (`System`) zu beeinflussen . Die Wertigkeiten der beiden Spieler (`Utility`) hängen von dem `System`, sowie weiteren Faktoren (`Cost`) ab.

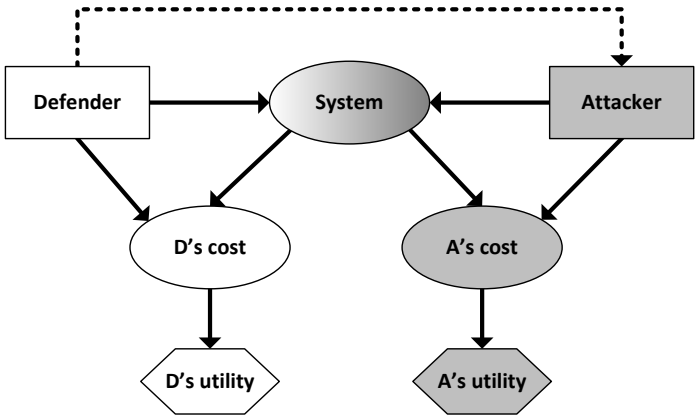


Abbildung 1.1: Einführendes Beispielszenario

Bei der gewählten Modellierung trifft zunächst der Verteidiger eine Entscheidung. Diese fließt im Anschluss in die Entscheidung des Angreifers ein. Es handelt sich folglich bei der Wahl des Verteidigers um eine Präventionsstrategie, um den Schaden des nachfolgenden Angriffs möglichst gering zu halten. Ziel dieser Modellierung ist es, eine optimale Verteidigungsstrategie unter den angegebenen Wertigkeiten des Verteidigers, sowie dem voraussichtlichen Verhalten des Angreifers zu berechnen. Da jedoch das Angreiferverhalten nicht bekannt ist, muss dieses möglichst exakt ermittelt werden. Dafür ist es nötig, die ungefähren Wertigkeiten des Angreifers einzuschätzen. Mit Hilfe dieser Daten kann das voraussichtliche Verhalten des Angreifers bestimmt werden, sodass es im Anschluss möglich ist, eine Verteidigungsstrategie zu berechnen.

1.2 Ziele der Arbeit

Die *Adversarial Risk Analysis* (ARA) ist ein neu aufkommendes Forschungsgebiet, welches sich auf den Bereichen der Risikoanalyse und Spieltheorie bewegt. Derzeit erfolgt die Berechnung von ARA-Instanzen durch manuell aufgestellte Berechnungsvorschriften. Eine allgemeine Vorgehensweise, beziehungsweise ein Algorithmus zum Analysieren solcher Modelle ist derzeit nicht bekannt, wohingegen im Bereich der Spieltheorie verschiedene Algorithmen existieren, um beispielsweise *Nash-Gleichgewichte* zu berechnen [LH64; KM03; GW03; GW04]. Die einfache Übertragbarkeit dieser Algorithmen ist allerdings nicht gegeben, da die Grundansätze, zum Beispiel durch die Annahme der *Common Knowledge Assumption*, stark divergieren. Eine weitere Herausforderung bei der Konzeption eines solchen Algorithmus ist der exponentiell wachsende Lösungsraum.

1.3 Aufbau der Arbeit

In diesem Kapitel wird zunächst eine Einführung und eine damit verbundene Motivation zu diesem Forschungsgebiet vorgestellt. Ein konkretes Beispiel dient zur zusätzlichen Illustration. Mit Hilfe des vorgestellten Problems werden zudem die Ziele dieser Arbeit verdeutlicht. Im nachfolgenden Kapitel 2 werden die theoretischen Grundlagen zu dieser Arbeit erläutert. Es wird hierbei auf die *Bayes'schen Netze*, die *Multi-Agent Influence Diagrams* (MAIDs) und die *Adversarial Risk Analysis* (ARA) eingegangen. Eine Abgrenzung zwischen MAIDs und ARA findet im anschließend Kapitel 3 statt. Dabei werden gezielt die Unterschiede und Gemeinsamkeiten herausgestellt, um im Anschluss einen fundierten Ansatz zu entwickeln. In der Spieltheorie ist die Berechnung von Nash-Gleichgewichten eines der Schlüsselprobleme. Dazu werden in Kapitel 4 verschiedene Algorithmen vorgestellt. Der Algorithmus von Daphne Koller und Brian Milch bildet hierbei die Grundlage für den entwickelten Algorithmus und wird daher in diesem Kapitel besonders herausgestellt. Der darauf basierende Lösungsansatz zur Berechnung von Verteidigungsstratgien wird im Kapitel 5 vorgestellt. Zur Demonstration der Anwendbarkeit werden zwei Fallstudien aus dem Seconomics Projekt betrachtet und evaluiert. Mit den gewonnen theoretischen Kenntnissen wird im Anschluss, in Kapitel 6, das entwickelte Werkzeug vorgestellt. Hierbei wird zur besseren Nachvollziehbarkeit ein Überblick über die eingesetzten Technologien gegeben. Die Vorstellung der entwickelten Komponente erfolgt durch Betrachtung einzelner Schlüsselfunktionalitäten. Da das entwickelte Konzept, sowie die Komponente, eine Grundlage zur Weiterentwicklung geben, ist eine besondere Qualitätssicherung des Programmes erforderlich. Die dafür eingesetzten Konzepte und Technologien werden ebenfalls in diesem Kapitel betrachtet. Die kritische Evaluation des Konzeptes und des entwickelten Werkzeugs erfolgt in Kapitel 7. Das abschließende Kapitel 8 gibt eine kurze und prägnante Zusammenfassung der Arbeit wieder. Es werden im Ausblick Eckpunkte vorgestellt, an denen noch Potential für die weitere Entwicklung oder Arbeiten, stattfinden kann.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vermittelt, um das nötige Hintergrundwissen für diese Arbeit zu erhalten. Dabei wird in Abschnitt 2.1 zunächst auf allgemeine *Bayes'sche Netze* eingegangen, welche anschließend zu *Multi-Agent Influence Diagramns* in Abschnitt 2.2 erweitert werden. Der Ansatz der *Adversarial Risk Analysis* wird in Abschnitt 2.3 vorgestellt.

Sofern nicht anders gekennzeichnet, sind die Beispiele und Definitionen von Abschnitt 2.1 und 2.2 aus der grundlegenden Arbeit zum Thema MAIDs von Daphne Koller und Brian Milch entnommen [KM03]. Die Grundlage zum Abschnitt 2.3, sowie das darin aufgeführte Beispiel, sind an der Ausarbeitung von David Insua, Jesus Rios und David Banks angelehnt [RRB09].

2.1 Bayes'sche Netze

Bayes'sche Netze (BN) (auch bekannt unter dem Namen *Probabilistic Networks* oder *Belief Networks*) bilden eine grafische Repräsentation für den Zusammenhang von *Zufallsvariablen (Random Variable, RV)* und derer *bedingten Wahrscheinlichkeiten (CPDs)* mit Hilfe eines *gerichteten azyklischen Graphen (DAG)*.

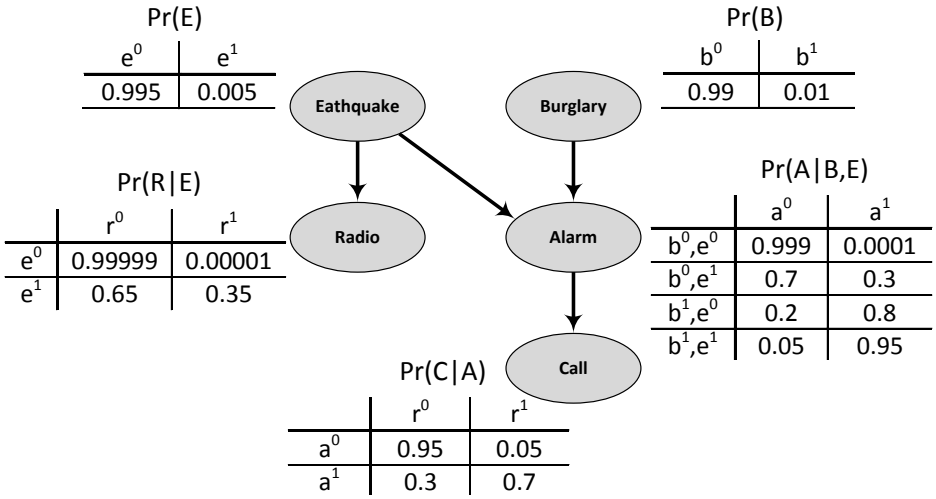


Abbildung 2.1: Beispiel eines Bayes'schen Netzes

Definition 2.1

Ein *Bayes'sches Netz* $\mathcal{B} = (G, Pr)$ wird durch einen gerichteten azyklischen Graphen G und einer Zuordnungsfunktion Pr beschrieben. Der Graph G besteht aus n Knoten, welche die gleichnamigen Zufallsvariablen $X_1, \ldots, X_n$ repräsentieren. Mit den Kanten von G wird jeweils die Abhängigkeit zwischen den Knoten dargestellt. Dabei liefert die Funktion $Pa(X)$ die *Eltern* des Knotens X in G. Die Funktion $Pr(X|Pa(X))$ ordnet jedem Knoten X, unter Einbeziehung der *Elternknoten*, eine bedingte Wahrscheinlichkeitsverteilung $Pr(X|\mathbf{pa})$ hinzu. Wobei **pa** eine Instanziierung von $Pa(X)$ ist.

Die Semantik hinter einem BN bildet die *multivariate Verteilung* (oder auch mehrdimensionale Verteilung) über $dom(\mathcal{X})$. Sei $U = \{X_1, \ldots X_n\}$ eine Menge von Zufallsvariablen. Wobei m_i die Anzahl der Zuständen von X_i beschreibt. Dann kann die Kombination der Zustände in einer Tabelle $P(U) = P(X_1, \ldots, X_n)$ gespeichert werden. Die Größe dieser Tabelle ist exponentiell beziehungsweise durch $m_1 \cdot \ldots \cdot m_n$ bestimmt.

Bestehen folglich Abhängigkeiten unter einzelnen Zufallsvariablen, lassen sich die selben Informationen in Form eines Bayes'schen Netzes ausdrücken. Dies hat den Vorteil, dass die Abhängigkeiten zwischen den einzelnen Variablen leichter erkennbar werden und die Anzahl der zu speichernden Wahrscheinlichkeiten erheblich sinken kann. Mit Hilfe der *Kettenregel* für BN kann jeder Wert der ursprünglichen multivariaten Verteilung über $dom(\mathcal{X})$ wiederhergestellt werden.

Definition 2.2

Die *Kettenregel* für Bayes'sche Netze kann zur Bestimmung der multivariaten Verteilung eines BN $\mathcal{B} = (G, Pr)$ über die RVs $X_1, \ldots, X_n$ verwendet werden:

$$P(X_1, \ldots X_n) = \prod_{i=1}^{n} Pr\Big(X_i | Pa(X_i)\Big)$$

Innerhalb eines BN kann es zu *Abhängigkeits-* beziehungsweise *Unabhängigkeitsstrukturen* kommen, welche auch innerhalb der durch die Kettenregel definierten multivariaten Verteilung auftreten. Dies gilt für jede mögliche Parametrisierung des Graphen.

Definition 2.3

Sei P eine Wahrscheinlichkeitsverteilung über $\mathcal{X}$ und seien $\mathbf{X}$, $\mathbf{Y}$ und $\mathbf{Z}$ drei paarweise disjunkte Untermengen von $\mathcal{X}$. Dann ist $\mathbf{X}$ *bedingt unabhängig* von $\mathbf{Y}$ gegeben $\mathbf{Z}$ in P, falls für jedes $\mathbf{z} \in dom(\mathbf{Z})$ mit $P(\mathbf{z}) > 0$ und für jedes $\mathbf{x} \in dom(\mathbf{X}), \mathbf{y} \in dom(\mathbf{Y})$ eine der folgenden äquivalenten Bedingungen erfüllt wird:

(i) $P(x|y,z) = P(x|z)$
(ii) $P(y|x,z) = P(y|z)$
(iii) $P(x,y|z) = P(x|z) \cdot P(y|z)$

Diese Art der Definition von *bedingter Unabhängigkeit* ist sehr stark, da diese die bedingte Unabhängigkeit für jede mögliche Belegung der Variablen fordert. Es sind außerdem die Unterschiede zur *marginalen Unabhängigkeit* zu beachten. Obwohl zwei Ereignisse nicht marginal unabhängig sind, können diese gegeben eines weiteren Ereignisses die Eigenschaften der bedingten Unabhängigkeit erfüllen.
Die Unabhängigkeit von Variablen kann allerdings auch mit der Struktur des Graphen effizient berechnet werden. Dabei werden Pfade als *aktiv* bezeichnet, falls diese Informationen *propagieren* beziehungsweise weiterleiten. Andererseits können Pfade auch durch Informationen *blockiert* werden. Diese Grundidee der aktiven Pfade hilft dabei die Unabhängigkeit von einem Knoten X zu einem Knoten Y gegeben einem dritten Knoten Z zu analysieren.

Definition 2.4

Gegeben sei ein BN $\mathcal{B} = (G, Pr)$ und ein ungerichteter Pfad $X_1 - \cdots - X_n$ in G. Ferner sei **E** eine Menge von Knoten aus G, welche das gegenwärtige Wissen darstellen. Dann ist der Pfad $X_1 - \cdots - X_n$ *aktiv*, gegeben der Menge **E**, falls eine der nachfolgenden Bedingungen zutrifft:

(i) eine Konfiguration $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ existiert, mit X_i oder ein Nachfolger ist in **E**
(ii) kein anderer Knoten des Pfades ist in **E**

Ein Pfad der nicht aktiv ist, wird als *blockiert* bezeichnet.

Mit der Eigenschaft der aktiven Pfade ist es möglich, eine zentrale Unabhängigkeitseigenschaft auf BN zu formulieren - die *d-separation*:

Definition 2.5

Sei G ein Graph einer BN-Struktur. Dann sind zwei Knoten X und Y gegeben einer Menge **E** von Knoten innerhalb von G *d-separiert*, falls jeder mögliche Pfad zwischen X und Y blockiert ist.

Zusammenfassend stellen Bayes'sche Netze ein gut erforschtes, mathematisches Hilfsmittel dar, um Unabhängigkeitsbedingungen in einer multivariaten Verteilung darzustellen. Die Repräsentation ist zudem kompakt und erlaubt es, effiziente Algorithmen anzuwenden.

2.2 Multi-Agent Influence Diagrams

Eine Erweiterung von BN bilden die *Influence Diagrams* (IDs). Diese führen die Möglichkeit ein, innerhalb eines solchen Netzwerkes Entscheidungen zu treffen, welche zu einer Veränderung eines Ausgangswertes (*Utility*) führen. Den daraus resultierenden *Erwartungswert* gilt es in der Regel zu maximieren. IDs betrachtet nur einen *Spieler*, beziehungsweise *Agenten*. Das Konzept der *Multi-Agent Influence Diagrams* erweitern diesen Ansatz zusätzlich um eine *Mehrspielerlösung*. Zunächst eine formalere Beschreibung.

Definition 2.6

Ein MAID beinhaltet eine feste Anzahl von Agenten, welche durch die Menge N repräsentiert werden. Die *Welt*, in der die Agenten agieren, wird durch die Menge der *Chance*-Variablen $\mathcal{X}$ und einer Menge der *Decision*-Variablen $\mathcal{D}_a$ für jedes $a \in \mathcal{A}$ beschrieben. Die *Utility*-Variablen eines Agenten a werden durch $\mathcal{U}_a$ notiert, wobei der Wertebereich stets durch eine Menge von eindeutig bestimmten, reellen Zahlen gegeben ist. Zur einfacheren Notation können die Gesamtheiten der Decision-Variablen durch $\mathcal{D} = \bigcup_{a \in \mathcal{A}} \mathcal{D}_a$ und der Utility-Variablen durch $\mathcal{U} = \bigcup_{a \in \mathcal{A}} \mathcal{U}_a$ beschrieben werden.

MAIDs definieren, analog zu BN, einen gerichteten azyklischen Graphen, mit den oben definierten Variablen als Knoten. Dabei sind die Eltern eines Knotens X durch die Menge $Pa(X) \subset \mathcal{X} \cup \mathcal{D}$ definiert. Analog zu BN werden die Chance-Variablen durch eine CPD $Pr(X|Pa(X))$ in Abhängigkeit zu den Elternknoten definiert. Die Eltern einer Decision-Variable $Pa(D)$ für ein $D \in D_a$ sind die Beobachtungen des Agenten a, welche ihm zum Zeitpunkt dieser Entscheidung zur Verfügung stehen. Um formal korrekt zu bleiben, werden auch CPDs über Utility-Variablen definiert $Pr(U|\mathbf{pa})$, wobei $\mathbf{pa}$ eine konkrete *Instanziierung* von $Pa(U)$ ist. Dabei wird zusätzlich gefordert, dass es sich um eine deterministische Zuweisung handelt. Die Zuweisungswerte sind also nur Wahrscheinlichkeitswerte mit $0 \dot{\vee} 1$, wobei die Wahrscheinlichkeit 1 nur einmal vorkommt. Abkürzend werden die Werte, welche durch eine Instanziierung $\mathbf{pa}$ bestimmt werden, durch $U(\mathbf{pa})$ angegeben.

Die grafische Notation ist ebenfalls eine erweiterte Fassung der BN. Chance-Variablen werden analog zu den BN als Ellipse dargestellt. Decision-Variablen als Rechteck und Utility-Variablen als Raute. Eine Kante zwischen zwei Decision-Variablen wird zur besseren Illustration durch eine gestrichelte Linie hervorgehoben.

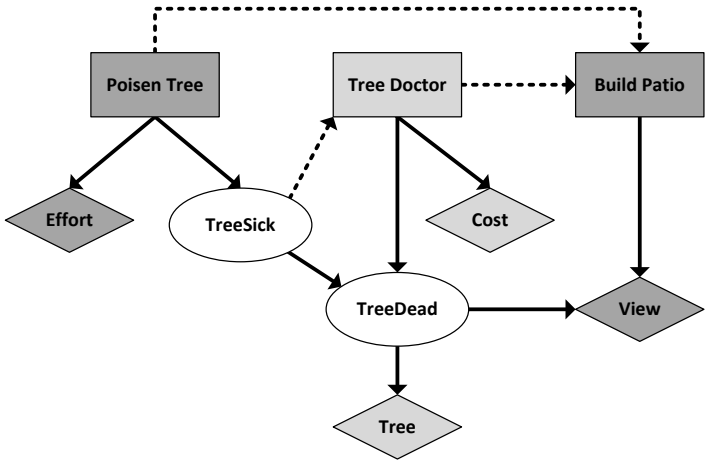


Abbildung 2.2: Einfaches 2-Spieler MAID (Knoten sind farblich dem Spieler zugeordnet)

Innerhalb eines solchen MAIDs treffen jeweils die Agenten a bei dem Knoten $D \in \mathcal{D}_a$ eine Entscheidung.

Definition 2.7

Eine *Entscheidungsregel* für eine Decision-Variable D ist eine Wahrscheinlichkeitsverteilung über $dom(D)$, wobei die vorherigen Entscheidungen **pa** aus $P(D)$ berücksichtigt werden. Eine Menge von Entscheidungsregeln, für alle Entscheidungen $D \in \mathcal{D}_a$ die einem Agenten a zur Verfügung stehen, wird als *Strategie* bezeichnet.

Eine Zuweisungsmenge σ für jede Entscheidung $D \in \mathcal{D}$ wird als *Strategieprofil* bezeichnet. Ein *partielles* Strategieprofil $\sigma_{\mathcal{E}}$ ist eine Zuweisungsmenge von Entscheidungsregeln, wobei $\mathcal{E} \subset \mathcal{D}$. Die Negation $\sigma_{-\mathcal{E}}$ bezieht sich dabei auf die Decision-Variablen, welche nicht in der Menge $\mathcal{E}$ enthalten sind.

Gegeben sei nun ein konkretes Strategieprofil σ für ein MAID $\mathcal{M}$. Die Anwendung der einzelnen Entscheidungsregeln aus σ induzieren ein neues MAID $\mathcal{M}[\sigma]$, in welchem die Elemente von σ in Chance-Knoten umgewandelt worden sind, mit $\sigma(D)$ als Wahrscheinlichkeitstabelle. Werden alle Entscheidungsknoten aus $\mathcal{M}$ abgedeckt, bildet das induzierte MAID $\mathcal{M}[\sigma]$ mit der multivariaten Verteilung $P_M[\sigma]$ ein BN. Mit Hilfe dieser Definitionen kann nun auch leicht der *Erwartungswert* der gesamten Utility-Variablen eines Agenten a bestimmt werden.

$$E[U_a(\sigma)] = \sum_{U \in \mathcal{U}_a} \sum_{u \in dom(U)} P_{\mathcal{M}[\sigma]}(U = u) \cdot u$$

Da das Verhalten eines Agenten von den Wertigkeiten abhängt kann mit dem definierten Erwartungswert der Optimierungsprozess des jeweiligen Agenten beschrieben werden.

Definition 2.8

Sei $\mathcal{E} \subset \mathcal{D}_a$ und σ ein Strategieprofil. Dann ist $\sigma^*_{\mathcal{E}}$ *optimal bezüglich des Strategieprofils* σ wenn, im induzierten MAID $\mathcal{M}[\sigma_{-\mathcal{E}}]$, die Strategie optimal bezüglich aller anderen Strategien $\sigma'_{\mathcal{E}}$ ist:

$$E[U_a]((\sigma_{-\mathcal{E}}, \sigma^*_{\mathcal{E}})) \geq E[U_a]((\sigma_{-\mathcal{E}}, \sigma'_{\mathcal{E}}))$$

In der Spieltheorie ist besonders das *Nash-Gleichgewicht* (auch *Nach-Equilibrium*), ein zentraler Begriff um die Rationalität bei einem nicht-kooperativen Spiel auszudrücken. Anschaulich erklärt ist ein Strategieprofil in einem Nash-Gleichgewicht, falls kein Agent, ohne Abnahme des Utility-Wertes, seine Strategie verändern kann. Formal lässt sich dies wie folgt formulieren:

Definition 2.9

Ein Strategieprofil σ befindet sich im Nash-Gleichgewicht für ein MAID $\mathcal{M}$, falls alle Agenten $a \in A$ mit ihren Entscheidungsregeln $\sigma_{\mathcal{D}_a}$ optimal bezüglich des Strategieprofils σ sind.

Eine der Hauptaufgabe bei der Analyse von MAIDs liegt darin, ein solches Strategieprofil zu finden, welches die Eigenschaften des Nash-Gleichgewichtes erfüllt.

2.3 Adversarial Risk Analysis

Die *Adversarial Risk Analysis (ARA)* definiert ein Konzept um eine Risikoanalyse mit mehreren intelligenten Gegenspielern durchzuführen. Dabei ist das wesentliche Vorgehen von ARA durch zwei Grundüberlegungen geprägt. Zum einen soll die klassische Risikoanalyse durch Modellierung von intelligenten Gegenspielern erweitert werden. Dies ermöglicht, ähnlich zu dem spieltheoretischem Ansatz, eine Auswertung von komplexen Entscheidungssituationen. Der zweite Gedanke bezieht sich gleichzeitig auf einen oft diskutierten Kritikpunkt der Spieltheorie - des gemeinsamen und vollständigen Wissens über die Mitspieler *(Common Knowledge Assumption)*. Diese Grundannahme, dass jeder Spieler über die einzelnen Aktionen und Wertigkeiten aller anderen Spieler informiert ist, entzieht sich insbesondere im Rahmen der Risikoanalyse einem realitätsnahen Ansatz. Daher werden innerhalb von ARA-Analysen einzelne Spieler, deren Wertigkeiten nicht genau bekannt sind, durch eine passende Wahrscheinlichkeitsverteilung, die zum Beispiel auf Grundlage historischer Daten ermittelt wurde, modelliert. Ein so behandelter Spieler wird also *verrauscht*, beziehungsweise mit einer *Unsicherheit* dargestellt.

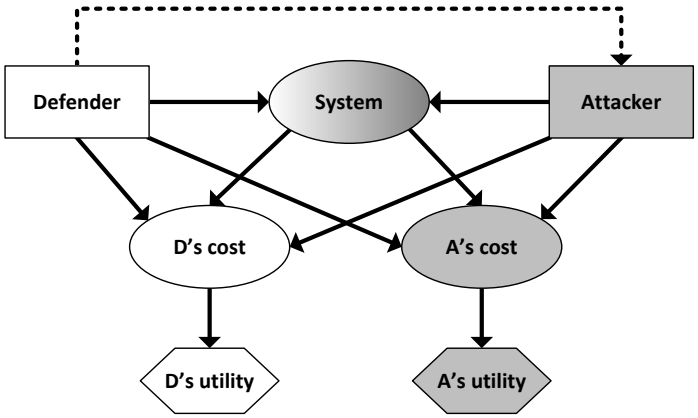


Abbildung 2.3: Ein Verteidigungs-Angriffsmodell in ARA

Abbildung 2.3 stellt ein Beispielmodell für einen ARA-Prozess dar. In dem Modell gibt es zwei Parteien. Einen *Angreifer* (`Attacker`) und einen *Verteidiger* (`Defender`). Beide Parteien beeinflussen ein `System`, welches der *Verteidiger* schützen und der *Angreifer* infiltrieren möchte. Bei dieser Modellierung handelt es sich um eine *sequentielles* Verteidigungs-Angriffs-Muster. Zunächst trifft der *Verteidiger* eine Entscheidung bezüglich seiner *Verteidugungsstrategie*. Anschließend, auf der vorherigen Entscheidung vom *Verteidiger* basierend, wählt der *Angreifer* seine dementsprechende *Angriffsstrategie*.

3 Spielrepräsentationen

Dieses Kapitel gibt einen Überblick über die gängigsten Spielrepräsentationen der Spieltheorie und untersucht die Darstellbarkeit von ARA-Instanzen mit Hilfe von MAIDs.

3.1 Normalform

Die *Normalform* bildet eine Beschreibung für Spiele im Rahmen der Spieltheorie. Dabei wird die Normalform als Matrix dargestellt, welche die Strategien und die Wertigkeiten der einzelnen Spieler beinhaltet. Die Tabellen 3.1, 3.2 und 3.3 zeigen unterschiedliche Beispiele für die Darstellungsform. Dabei sind Nash-Gleichgewichte fett hervorgehoben. Es ist also möglich, dass Spiele keine, mehrere oder nur ein solches Nash-Gleichgewicht beinhalten.

P1/P2	σ_3	σ_4
σ_1	(1,0)	(0,1)
σ_2	(0,1)	(1,0)

Tabelle 3.1: Matching Pennies

P1/P2	σ_3	σ_4
σ_1	**(3,1)**	(0,0)
σ_2	(0,0)	**(1,3)**

Tabelle 3.2: Battle of Sexes

P1/P2	σ_3	σ_4
σ_1	**(-5,-5)**	(-1,-10)
σ_2	(-10,-1)	(-2,-2)

Tabelle 3.3: Prisoner's Dilemma

Mit Hilfe der Normalform werden in der Regel nur simultane Spiele, beziehungsweise Spiele mit nicht perfekter Informationslage beschrieben. Es ist allerdings auch möglich eine sequentielle Spieldarstellung zu formulieren. Dafür müssen die unterschiedlichen Ausgänge zu einer neuen Strategie zusammengefasst werden [FT91].

3.2 Extensivform

Im Gegensatz zur Normalform (siehe Abschnitt 3.1) bildet die Extensivform eine grafische Art der Spieldarstellung. Mit dieser ist es leicht möglich sequentielle Spielabläufe zu modellieren und die Entscheidungsphasen explizit darzustellen. Dabei kann auch eine nicht perfekte Informationslage modelliert werden. Unvollständige Informationen werden mit Chance Knoten dargestellt. Abbildung 3.1 zeigt ein Bei-

spiel für ein einfaches 2-Spieler Spiel in der Extensivform. Die Extensivform wird in der Fachliteratur auch als Spielbaum ($GameTree$, GT) bezeichnet [FT91].

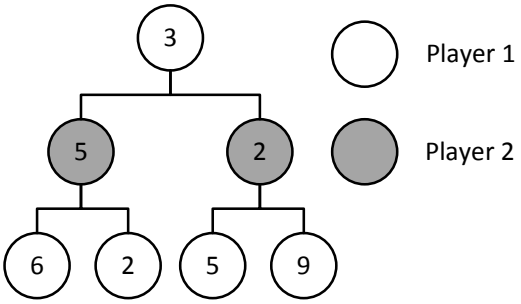


Abbildung 3.1: Extensivform eines 2-Spieler Spiels

3.3 MAIDs

Die Multi-Agent Influence Diagrams werden detailliert in den Grundlagen vorgestellt (vergleiche Abschnitt 2.2). Diese stellen eine Erweiterung der Bayes'schen Netze und der Influence Diagramms dar. Ähnlich zu der Normalform und der Extensivform sind diese für Mehrspielerprobleme ausgelegt. In der einführenden Literatur von Daphne Koller und Brian Milch [KM03] wird gezeigt, dass diese semantisch äquivalent zu der Extensivform, beziehungsweise den Spielbäumen sind. Im Gegensatz zu Spielbäumen können MAIDs im besten Fall exponentiell kompakter sein. Dies hängt mit dem Aufspalten an den Entscheidungsknoten im Spielbaum zusammen, welche durch schlechte Anordnung viele Symmetrien enthalten können (vergleiche Abbildung 3.2).

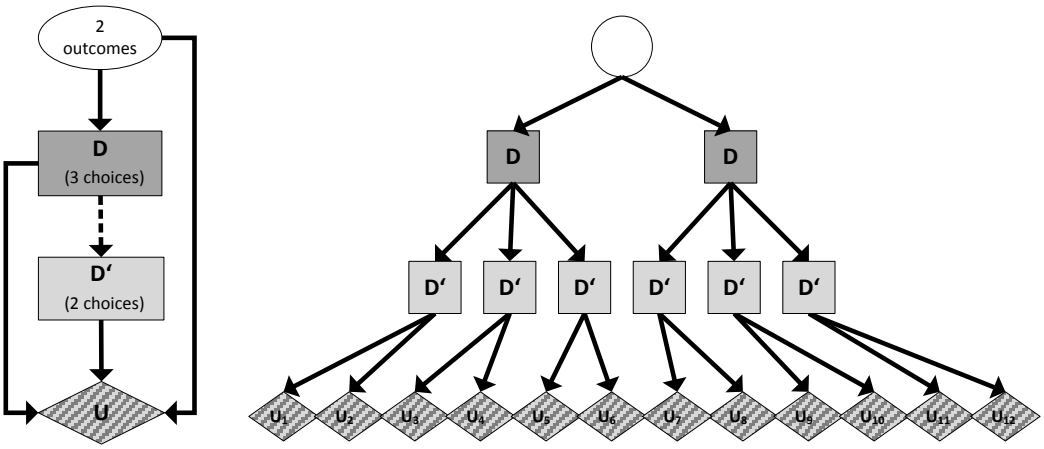


Abbildung 3.2: MAID und zugehöriger Spielbaum

3.4 MAIDs und ARA

Eine Vorstellung der Adversarial Risk Analysis erfolgte bereits in den Grundlagen (vergleiche Abschnitt 2.3). Bei dieser handelt es sich um keine Spieldarstellung sondern um ein Konzept, welches zur Risikoanalyse verwendet werden kann. Im Folgenden wird nun diskutiert, ob ARA-Analysen mit Hilfe von MAIDs durchgeführt werden können.
In der bisherigen ARA-Literatur (vergleiche [RRB09; Río+13a; Río+13b]) werden die Modelle über Influence Diagramms und Spielbäume beschrieben. Weitere Modellierungen ähneln der Syntax von MAIDs. Im Allgemeinen sieht ARA eine einseitige Unterstützung eines Agenten vor, dem Verteidiger. Dieser versucht mit einem subjektiv entwickelten Modell das Verhalten der anderen Spieler zu ermitteln, welche durch das fehlen von Informationen mit Hilfe von Wahrscheinlichkeitsverteilungen dargestellt werden. Dies stellt auch den wesentliche Unterschied zu der Struktur der MAIDs dar, da diese eine symmetrische Informationslage modellieren. Jeder Spieler besitzt die gleichen Kenntnisse über die Chance- und Utility-Knoten. Diese Annahme entspricht allerdings nicht dem ARA-Konzept. Die Modellierung von zusätzlichen Wahrscheinlichkeitstabellen, welche jeweils der Perspektive des Spielers angepasst sind, kann an dieser Stelle jeweils als eine subjektive Sichtweise auf das Spiel genutzt werden. Mit dieser einfachen Erweiterung der MAID Definition um Perspektiven der jeweiligen Agenten, ist es möglich ARA-Modelle zu formulieren. Beispiele für solche Modellierungen bilden die Fallstudien in Abschnitt 5.2.

4 Algorithmen zur Berechnung von Nash-Gleichgewichten

In diesem Kapitel werden in Abschnitt 4.1 ausgewählte Algorithmen aus der Spieltheorie vorgestellt. Die Auswahl eines Algorithmus, welcher die Grundlage für den entwickelten Ansatz bildet, wird in Abschnitt 4.2 getroffen. Der ausgewählte Algorithmus von Daphne Koller und Brian Milch wird anschließend in Abschnitt 4.3 detailliert betrachtet.

4.1 Vorstellung verschiedener Algorithmen

Dieser Abschnitt widmet sich der Vorstellung verschiedener Algorithmen zur Berechnung von Nash-Gleichgewichten.

4.1.1 Strategische Relevanz: Koller und Milch

Daphne Koller und Brian Milch zeigten in ihrer Ausarbeitung *Multi-Agent-Influence-Diagrams for representing and solving games* [KM03] eine Möglichkeit, Abhängigkeiten zwischen *Entscheidungsknoten* zu definieren. Diese genannte *Strategische-Relevanz* zwischen zwei Entscheidungen, kann auf die Optimierungsreihenfolge übertragen werden. So ist eine Entscheidung D' strategisch relevant für eine weitere Entscheidung D, wenn bei der Optimierung der Entscheidung D die Entscheidung von D' in diese mit einbezogen werden muss. Diese Eigenschaft kann mit Hilfe einer rein graphbasierten Definition in polynomieller Zeit überprüft werden. Die Abhängigkeitsbeziehung zwischen zwei Entscheidungen kann auf die Gesamtheit aller Entscheidungen ausgeweitet werden. Die dadurch entstehende Abhängigkeitsstruktur wird als *Relevanz-Graph* bezeichnet. Dieser direkte Graph zeigt sämtliche Abhängigkeiten innerhalb eines MAIDs und wird zur Bestimmung der Optimierungsreihenfolge verwendet.

4.1.2 Global Newton Method: Govindan und Wilson

Srihari Govindan und Robert Wilson haben in ihrer Arbeit *A Global Newton Method to compute Nash-Equilibria* [GW03] einen Algorithmus für endliche Spiele vorgestellt, welcher auf der globalen Newton Methode von Smale [Sma76] und der Homotopie Methode von Eaves basiert [Eav72; Eav84]. Mit Hilfe eines Theorems von Kohlberg und Mertens [KM86] kann gezeigt werden, dass der Graph der Nash-Equilibria homöomorph zu dem gesamten Zustände des Spiels ist. Durch den Homomorphismus kann eine Homotopie definiert werden, welche die gleiche Dimensionsgröße aufweist wie die Anzahl an reinen Strategien im Spiel. Durch Verfolgung des durch die Homotopie beschriebenen Pfades, mit der globalen Newton Methode,

ist es im Anschluss möglich, die Inverse des Homeomporphismus zu berechnen und damit die gemischten Strategien der Spieler zu bestimmen, welche die Eigenschaft eines Nash-Gleichgewichts erfüllen.

4.1.3 Interated Polymatrix Approximation: Govindan und Wilson

In einer weiteren Ausarbeitung von Srihari Govindan und Robert Wilson mit dem Titel *Computing Nash-Equilibria by iterated polymatrix approximation* [GW04] wird ein Algorithmus beschrieben, welcher Spiele durch eine Abfolge von *Polymatrizen* approximiert. Ein approximiertes Nash-Gleichgewicht kann im Anschluss mit dem Lemke-Howson Algorithmus [LH64] berechnet werden. Da der Algorithmus nicht zwangsläufig das gewünschte Ergebnis liefert, wird dieser als Vorberechnung für die globale Newton Methode (siehe Abschnitt 4.1.2) verwendet, um einen besseren Startwert zu erhalten.

4.1.4 Continuation Method: Blum, Shelton und Koller

Ben Blum, Christian R. Shelton und Daphne Koller haben in ihrer Arbeit mit dem Titel *A Continutation Method for Nash Equilibria in Structured Games* [BSK06] einen Algorithmus zur Berechnung von Nash-Gleichgewichten entwickelt, welcher unter anderem auf den zuvor entwickelten Konzepten von Srihari Govindan und Robert Wilson (Abschnitte 4.1.2 und 4.1.3) aufbaut. Es handelt sich bei dem entwickelten Algorithmus um eine exakte Berechnung eines Nash-Gleichgewichtes. Wie bei den Vorgängern wird hierzu die Lösung eines abgewandelten Spieles bestimmt, welches leichter zu berechnen ist. Anschließend wird das Spiel sukzessiv zurück ins Ursprungsspiel gewandelt. Die berechnete Lösung wird dementsprechend aktualisiert. Ist die Umwandlung abgeschlossen, liegt ebenfalls eine Lösung für das Ursprungsspiel vor, welches die Eigenschaft eines Nash-Gleichgewichtes erfüllt. Der Algorithmus garantiert mindestens ein gefundenes Gleichgewicht.

4.2 Auswahl eines Algorithmus

Nudelman und weitere [Nud+04] haben in ihrer Ausarbeitung *Run the GAMUT: A Comprehensive Approach To Evaluating Game-Theoretic Algorithms* verschiedene Algorithmen aus der Spieltheorie miteinander verglichen. Die Algorithmen von Govindan und Wilson [GW03; GW04] waren in einigen Fällen die Effizientesten. Aus diesem Grund basiert die Methode von Blum, Shelton und Koller auf dem selben Prinzip und bildet vermutlich den derzeit effizientesten Algorithmus [BSK06]. Allerdings kann es in manchen Fällen vorkommen, dass der Algorithmus einen Zyklus aufweist und nicht terminiert. Dieses Phänomen ist auch bei dem Algorithmus von Govindan und Wilson zu beobachten. Derzeit ist nicht bekannt, wann diese Zyklen entstehen. Einen sehr interessanten Aspekt bildet allerdings auch die Struktur des Relevanz-Graphen. Mit diesem lassen sich leicht Abhängigkeiten zwischen Entscheidungen berechnen. Die Datenstruktur könnte somit auch zum Auffinden von Mustern genutzt werden, um im Anschluss gezielt ARA-Templates anwenden zu können (vergleiche [RRB09]). Da der Algorithmus von Koller und Milch nur grundlegende Operationen benötigt, welche auch später von anderen Prozeduren leicht verwendet

werden können und auf der Struktur des Relevanz-Graphen arbeitet, welche später zur Mustererkennung verwendet werden kann, eignet sich dieser wesentlich besser für den Aufbau eines ersten Analyse-Werkzeugs.

4.3 Algorithmus von Daphne Koller und Brian Milch

Dieser Abschnitt widmet sich dem Algorithmus von Daphne Koller und Brian Milch [KM03]. Ziel ist es, die wichtigsten Erkenntnisse der Arbeit zusammenzufassen und diese im späteren Verlauf der Ausarbeitung nutzbar zu machen.

4.3.1 Strategische Relevanz

Der mögliche Entscheidungsraum eines MAIDs wächst exponentiell mit jeder zukommenden Entscheidung. Daher ist es insbesondere bei großen Modellen wichtig, eine effiziente Berechnungsvorschrift anzuwenden, welche nicht den vollständigen Lösungsraum traversiert. Eine Möglichkeit die Struktur der MAIDs auszunutzen, bildet das Ermitteln von Abhängigkeiten zwischen Entscheidungen. Mit diesem Wissen kann die Berechnung eines globales Gleichgewicht auf die lokale Berechnungen einzelner Entscheidungsknoten reduziert werden, sodass die Gesamtheit des Lösungsraumes nicht mehr betrachtet werden muss.
Um folglich für einen Spieler a bei einem Strategieprofil σ eine optimale Entscheidungsregel δ für ein $D \in D_a$ zu finden, ist es wichtig zu wissen, welche anderen Entscheidungsregeln aus σ die Optimalität der Entscheidungsregel beeinflussen können.

Definition 4.1

Seien D und D' zwei Entscheidungsknoten innerhalb eines MAID $\mathcal{M}$. Dann ist D' strategisch relevant für D, wenn zwei Strategieprofile σ und σ' und eine Entscheidungsregel δ für D existieren, sodass δ optimal für σ ist und σ' nur von σ in D' abweicht.
Aber keine Entscheidungsregel δ^* existiert, welche mit allen Instanziierungen der Elternknoten $\mathbf{pa} \in dom(Pa(D))$ übereinstimmt, wobei $P_{\mathcal{M}[\sigma]}(\mathbf{pa}) > 0$ erfüllt wird und optimal für σ' ist.

Der zweite Abschnitt der Definition behandelt den Sonderfall der Null-Zug Wahrscheinlichkeiten. Im Gegensatz zu der kanonischen Forderung der nicht Optimalität von σ'. Wird beispielsweise ein Strategieprofil σ betrachtet, welches einer Entscheidung D eine reine Entscheidungsregel zuweist und somit einer zweiten folgenden, observierenden Entscheidung D' ebenfalls eine reine optimale Entscheidungsregel δ für σ ermöglicht, dann ist δ möglicherweise nicht mehr optimal, falls in σ die Entscheidung von D abgeändert wird. Somit würde D' von D mittels der kanonischen Definition strategisch abhängen. Dies ist jedoch nicht sinnvoll, da die Entscheidung von D dem Knoten D' zur Verfügung steht. Die zusätzlichen Forderungen erlauben bei dieser Situation eine Konstruktion von δ^*, welche die Änderungen von σ berücksichtigt.
Da es jedoch nicht einfach möglich ist alle verschiedenen Kombinationen von Strategieprofilen σ und σ' zu testen um die strategische Relevanz zweier Entscheidungskno-

ten zu ermitteln, wird ein qualitativ hochwertigeres Kriterium gesucht. Das folgende Lemma bildet dafür den ersten Schritt. Es zeigt auf, welches Maximierungsproblem gelöst werden muss, um eine optimale Entscheidungsregel δ unter einem gegeben Strategieprofil σ für eine Entscheidung D zu berechnen.

Lemma 4.1

Sei δ eine Entscheidungsregel für eine Entscheidungsvariable $D \in \mathcal{D}_a$ eines MAID $\mathcal{M}$, und σ ein Strategieprofil für $\mathcal{M}$. Dann ist δ optimal für σ, wenn für jede Instanziierung $\mathbf{pa}_D$ von $Pa(D)$ mit $P_{M[\sigma]}(\mathbf{pa}_D) > 0$, die Entscheidungsfunktion $\delta(D|\mathbf{pa}_D)$ eine Lösung des folgenden Maximierungsproblems ist:

$$\max_{P^*} \sum_{d \in dom(D)} P^*(d) \sum_{U \in \mathcal{U}_D} \sum_{u \in dom(U)} P_{\mathcal{M}[\sigma]}(u|d, \mathbf{pa}_D) \cdot u \tag{4.1}$$

Um die Optimalität einer Entscheidungsregel δ bezüglich eines Strategieprofils σ zu gewährleisten, muss δ also nur dem Maximierungsproblem aus (4.1) genügen.

Da MAIDs auf Bayes'schen Netzen definiert sind und diese bereits eine solide Forschungsgrundlage liefern, kann auf die Definition der *notwendigen Wahrscheinlichkeitsknoten* (*requisite probability node*) zurückgegriffen werden, um herauszufinden welche Knoten die Lösung aus (4.1) beeinflussen können.

Definition 4.2

Sei G ein Bayesches Netzwerk und X und Y zwei Mengen von Variablen des Bayeschen Netzwerkes. Dann ist ein Knoten Z ein notwendiger Wahrscheinlichkeitsknoten für eine Anfrage $P(X|Y)$, falls zwei Bayesche Netzwerke $\mathcal{B}_1$ und $\mathcal{B}_2$ über G existieren, welche identisch bis auf die Zuweisung von Z sind, aber $P_{\mathcal{B}_1}(X|Y) \neq P_{\mathcal{B}_2}(X|Y)$ gilt.

Geiger und Pearl [GP90] haben ein grafisches Kriterium entwickelt um herauszufinden, ob ein Knoten Z innerhalb eines Bayes'schen Netzwerkes für eine Anfrage $P(X|Y)$ gebraucht wird.

Lemma 4.2

Seien $\mathcal{B}_1$ und $\mathcal{B}_2$ zwei Bayesche Netzwerke über der selben Variablenmenge, welche identisch bis auf die Zuweisung der Wahrscheinlichkeitstabellen der Knotenmenge $\mathcal{Z}$ sind. Seien weiter $\mathcal{X}$ und $\mathcal{Y}$ zwei beliebige Knotenmengen. Falls kein $Z \in \mathcal{Z}$ existiert, mit einem zusätzlichen Knoten $\hat{Z}$, welcher den Eltern von Z hinzugefügt wird, sodass ein aktiver Pfad von $\hat{Z}$ zu $\mathcal{X}$, gegeben der Menge $\mathcal{Y}$, führt. So gilt $P_{\mathcal{B}_1}(X|Y) = P_{\mathcal{B}_2}(X|Y)$

Mit Hilfe des Lemmas 4.2 und der Eigenschaft des Maximierungsproblemns aus (4.1) kann die *strategische Erreichbarkeit* (*s-reachability*) definiert werden. Ähnlich zu der *d-Separation* von den Bayes'schen Netzwerken, ist diese ein rein grafisches Kriterium. Dieses Kriterium erfüllt allerdings nicht die Eigenschaft der Symmetrie.

Definition 4.3

Ein Knoten D' ist *strategisch erreichbar* (*s-reachable*) von einem anderen Knoten D in einem MAID $\mathcal{M}$, falls ein *Utility*-Knoten $U \in \mathcal{U}_D$ existiert, mit einem neuen Elternknoten $\widehat{D'}$, welcher D hinzugefügt wird, sodass ein aktiver Pfad in $\mathcal{M}$ von $\widehat{D'}$ zu U gegeben $Pa(D) \cup \{D\}$ existiert.

Die Übereinstimmung der strategischen Relevanz zu der Definition der strategischen Erreichbarkeit kann ähnlich zu der d-seperation und der Unabhängigkeit innerhalb der Bayes'schen Netzwerken mit Hilfe der von zwei Theoremen bewiesen werden.

Theorem 4.1

Seien D und D' zwei Entscheidungsknoten eines MAIDs $\mathcal{M}$ und D' ist nicht strategisch erreichbar von D in $\mathcal{M}$, dann ist D' auch nicht strategisch relevant für D.

Theorem 4.2

Seien D und D' zwei Entscheidungsknoten eines MAID $\mathcal{M}_1$ und D' ist strategisch erreichbar von D in $\mathcal{M}_1$, dann existiert ein MAID $\mathcal{M}_2$ mit der gleichen Graphstruktur, in welchem D' in $\mathcal{M}_2$ auch strategisch relevant für D ist.

Die Eigenschaft der strategischen Erreichbarkeit und somit der strategischen Relevanz kann mit Hilfe eines Graphen visualisiert werden. Dieser ist durch Definition stets ein gerichteter Graph und wird wie folgt definiert:

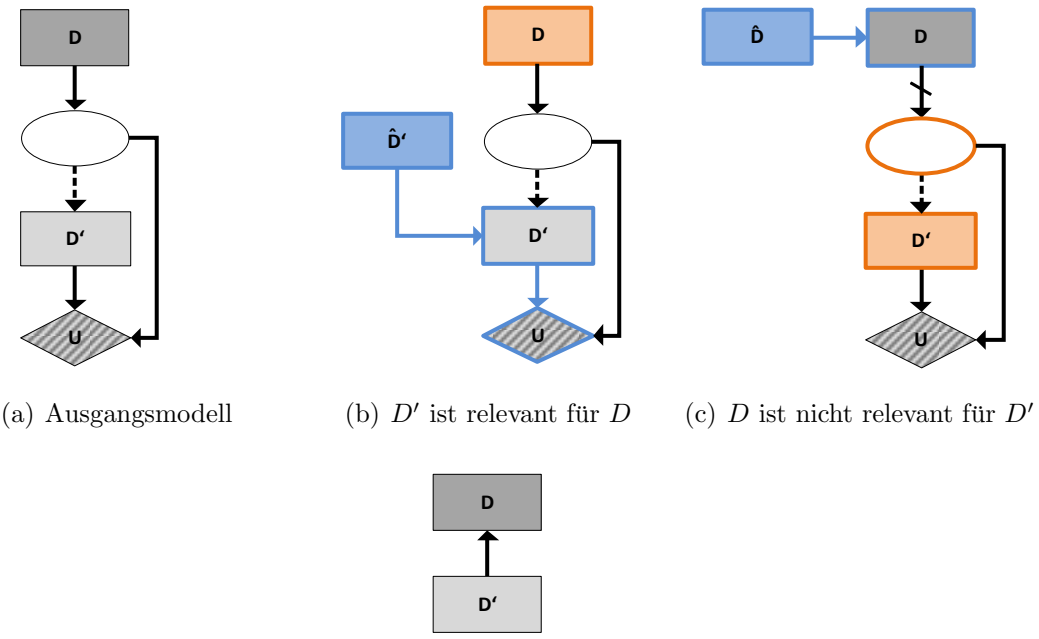
Definition 4.4

Der Relevanz-Graph $\mathcal{R}_\mathcal{M}$ eines MAID $\mathcal{M}$ ist ein gerichteter Graph, mit der Knotenmenge $\mathcal{D}$ aus $\mathcal{M}$ und einer Kante $D' \to D$, falls D' strategisch von D erreicht werden kann.

Um für ein beliebiges MAID $\mathcal{M}$ den zugehörigen Relevanz-Graphen zu berechnen, muss also folglich für jedes Entscheidungspaar die strategische Relevanz ausgewertet werden. Dies sind $|\mathcal{D}|^2$ viele Auswertungen. Die Bestimmung selbst kann beispielsweise mit Shachters Bayesball [Sha98] berechnet werden, welcher in linearer Anzahl zu den Knoten die Prüfung eines aktiven Pfades und somit auch der strategischen Relevanz vornehmen kann.

Beispieldurchführung einer Strategischen Relevanz-Analyse

Im Folgenden wird eine Bestimmung des Relevanz-Graphen zu einem vorgegeben MAID durchgeführt (siehe Abbildung 4.1), um die vorgestellten Ergebnisse zu illustrieren. Abbildung 4.1(a) zeigt ein einfaches 2-Spieler MAID. Der Utility-Knoten ist hierbei vereinfacht dargestellt und bildet für jeden Spieler einen eigenen Knoten, jeweils mit den selben Elternknoten.



(a) Ausgangsmodell (b) D' ist relevant für D (c) D ist nicht relevant für D'

(d) Resultierender Relevanz-Graph

Abbildung 4.1: Untersuchung der Strategischen Relevanz

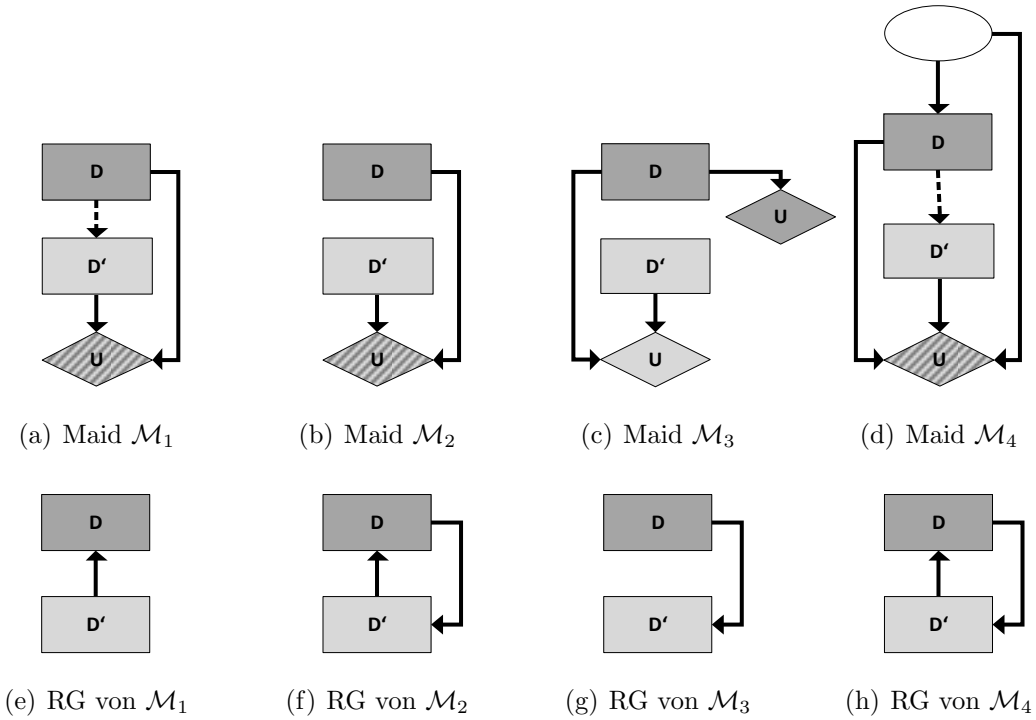
Um den Relevanz-Graphen aufzubauen, muss die strategische Relevanz der beiden Entscheidungsknoten überprüft werden. Zunächst wird in Abbildung 4.1(b) getestet, ob der Entscheidungsknoten D' strategisch relevant für den Knoten D ist. Hierfür wird ein *Dummy*-Knoten $\hat{D}'$ der Menge der Elternknoten von D' hinzugefügt. Die Menge E wird durch D bestimmt. Nun wird nachvollzogen, ob ein aktiver Pfad von $\hat{D}'$ zu U gegeben der Beweismenge E existiert. Da ein Pfad gefunden werden kann, der nicht blockiert ist, existiert ein aktiver Pfad und somit ist D' strategisch relevant für D. Umgekehrt gilt dies aber nicht und kann in Abbildung 4.1(c) nachvollzogen werden. Hierfür wird erneut ein Dummy-Knoten der Menge der Elternknoten von D hinzugefügt. In der Beweismenge E befinden sich nun D' und $Pa(D')$, also auch der Chance-Knoten. Da der einzige Pfad zu dem Utility-Knoten U über einen Knoten der Beweismenge führt, ist dieser blockiert und demnach existiert kein aktiver Pfad von D nach U. Daher ist D auch nicht strategisch relevant für D'. Der resultierende Relevanz-Graph ist in Abbildung 4.1(d) dargestellt.

Weitere Beispiele für Relevanz-Graphen

Die Abbildung 4.2 zeigt vier unterschiedliche MAIDs und deren berechnete Relevanz-Graphen.

MAID $\mathcal{M}_1$ zeigt ein Spiel mit perfekten Informationen - es existiert kein Chance Knoten, welcher die Ergebnisse verrauscht. Da D' die Aktion von D observieren kann, ist dessen Entscheidung nicht von D abhängig. Umgekehrt ist dies nicht der Fall. Daher ist D' strategisch relevant für D.

MAID $\mathcal{M}_2$ und $\mathcal{M}_3$ bilden Beispiele für simultane Spiele. Beide Spieler haben keine Kenntnisse über die Entscheidung des Gegenspielers. Bei $\mathcal{M}_2$ ist es jedem Spieler möglich, den Utility-Knoten des Gegenspielers zu beeinflussen. Daher ist der zugehörige Relevanz-Graph zyklisch. Dies ist bei $\mathcal{M}_3$ nicht der Fall, da D' keinen Einfluss auf den Utility-Knoten von D hat. Dementsprechend ist der zugehörige Relevanz-Graph azyklisch.



(a) Maid $\mathcal{M}_1$ (b) Maid $\mathcal{M}_2$ (c) Maid $\mathcal{M}_3$ (d) Maid $\mathcal{M}_4$

(e) RG von $\mathcal{M}_1$ (f) RG von $\mathcal{M}_2$ (g) RG von $\mathcal{M}_3$ (h) RG von $\mathcal{M}_4$

Abbildung 4.2: Weitere Beispiele für Relevanz-Graphen

MAID $\mathcal{M}_4$ bildet ein Gegenbeispiel für die Annahme, dass eine Entscheidung D', nicht von einer Entscheidung D abhängig sein kann, obwohl die Entscheidung von D seitens D' überwacht wird. Dies geschieht mittels eines Chance-Knotens, dessen Ausgabe nur D bekannt ist, aber nicht D'.

4.3.2 Berechnung von Nash-Gleichgewichten

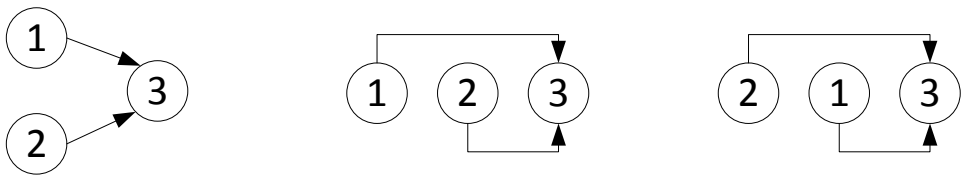
Die gewonnen Kenntnisse aus dem Bereich der strategischen Abhängigkeiten von Entscheidungen beziehungsweise dem Relevanz-Graphen, sind für die Berechnung eines Nash-Gleichgewichtes von Vorteil. Mit der erhaltenen Abhängigkeitsstruktur kann die Berechnung eines Gleichgewichtes auf einzelne Teilprobleme reduziert werden. Jedes Teilproblem kleiner als der Vorgänger. Im Folgenden werden zwei Algorithmen vorgestellt, welche beide auf dem Relevanz-Graph basieren. Zunächst wird mit Algorithmus 4.1 eine Version für den azyklischen Relevanz-Graphen vorgestellt, diese wird im späteren Verlauf auf den zyklischen Fall (Algorithmus 4.2) erweitert.

Azyklischer Fall

Ist der gewonnene Relevanz-Graph $\mathcal{R}_{\mathcal{M}}$ eines MAID $\mathcal{M}$ azyklisch, führt die topologische Sortierung des Graphen $\mathcal{R}_{\mathcal{M}}$ zu einer Optimierungsreihenfolge. Im azyklischen Fall existiert stets eine solche Sortierung. Diese muss allerdings nicht eindeutig sein (siehe Abbildung 4.3).

Definition 4.5

Eine Folge aller Knoten $v_1, v_2, \ldots, v_n$ eines gerichteten Graphen G heißt topologisch sortiert, falls für einen Knoten v_i eine Kante zu v_j existiert, mit $i < j$.



(a) Ausgangsgraph G_1 (b) Topologische Sortierung T_1 (c) Topologische Sortierung T_2

Abbildung 4.3: Unterschiedliche topologische Sortierungen

Der azyklische Fall ist eine Verallgemeinerung von bereits existierenden *Backward-Induction* Algorithmen [Sha90; JJD94]. Durch die zugrundeliegende strategische Abhängigkeit ist es möglich, jede einzelne Entscheidung sukzessiv zu optimieren.

Algorithmus 4.1: Berechnung des Nash-Gleichgewichtes (Azyklisch)

Input : Given a MAID $\mathcal{M}$ with an acyclic relevance graph, a topological ordering $D_1, \ldots, D_n$ of the relevance graph for $\mathcal{M}$

1 Let σ^0 be an arbitrary fully mixed strategy profile for $\mathcal{M}$
2 **for** $i = 1$ *to* n **do**
3 Let δ be a decision rule for D_i that is optimal for σ^{i-1}
4 Let $\sigma^i = (\sigma^{i-1}_{-D_i}, \delta)$
5 Output σ^n as an equilibrium of $\mathcal{M}$

Eingabe des Algorithmus 4.1 bildet der azyklische Relevanz-Graph des zugehörigen MAID und die daraus resultierende topologische Sortierung der Entscheidungsknoten $D_1, \ldots, D_n$. Zunächst wird in Zeile 1 ein beliebiges gemischtes Strategieprofil σ^0 erzeugt. Anschließend erfolgt die Traversierung der topologischen Sortierung in Schritt 2. Für jede betrachtete Entscheidung wird in Schritt 3 eine Entscheidungsregel δ ermittelt, welche unter dem akutellen Strategieprofil σ^{i-1} optimal ist. In Schritt 4 wird das Strategieprofil mit der neuen Entscheidungsregel aktualisiert. Dieser Vorgang wird für jede Entscheidung D durchgeführt. Bei Terminierung erfüllt das aktuelle Strategieprofil σ^n die Eigenschaften eines Nash-Gleichgewichtes auf $\mathcal{M}$.

Um zu zeigen, dass Algorithmus 4.1 immer als Ausgabe ein Nash-Gleichgewicht liefert, muss unter anderem bewiesen werden, dass die berechneten Entscheidungsregeln δ in jedem Schritt nie ihre Optimalität bezüglich des neu generierten Strategieprofiles σ^i verlieren. Dies kann mit Hilfe des folgenden Lemmas gezeigt werden.

Lemma 4.3

Sei D ein Entscheidungsknoten innerhalb eines MAID $\mathcal{M}$, δ eine Entscheidungsregel für D, und σ das zugehörige Strategieprofil mit der Optimalität von δ. Sei σ' ein weiteres Strategieprofil, welches nur für Entscheidungsknoten D' abweicht, welche nicht strategisch erreichbar von D sind. Wenn δ ein gemischt Belegung für alle Entscheidungsknoten D' beinhaltet, dann ist δ auch optimal für σ'.

Mit Lemma 4.3 kann einfach gezeigt werden, dass für jeden Spieler die gewählte Entscheidungsfolge für das das resultierende Strategieprofil σ^n optimal ist. Daher führt zwangsläufig jede einzelne Entscheidungsänderung eines Spielers zu einem schlechteren Erwartungswert. Die Definition des Nash-Gleichgewichtes ist jedoch noch umfassender. Diese erlaubt es auch einem Spieler simultan mehrere Entscheidungen zu ändern (vergleiche Abschnitt 2.2). Es kann jedoch mit Hilfe eines Induktionsbeweises auf den möglich änderbaren Entscheidungen gezeigt werden, dass die Eigenschaft der Optimalität eines Spielers nicht verletzt wird und das dadurch das finale Strategieprofil in jedem Fall die Eigenschaft eines Nash-Gleichgewichtes erfüllt.

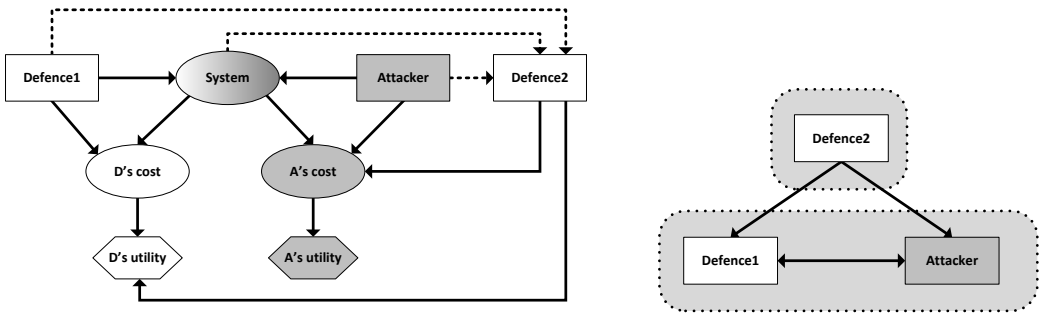
Zyklischer Fall

Allerdings ist es auch möglich, dass der Relevanz-Graph einen Zyklus aufweist und daher Algorithmus 4.1 nicht mehr anwendbar ist. Dies ist zum Beispiel bei jedem simultanen 2-Spieler Spiel der Fall, welches aus zwei Entscheidungen D_1 und D_2 besteht. Da weder D_1 noch D_2 von der anderen Entscheidung in Kenntnis gesetzt werden, sind diese beide voneinander abhängig und beeinflussen somit gegenseitig den möglichen Ausgangswert. Der Relevanz-Graph enthält also einen Zyklus. Dennoch kann in solchen Fällen die Relevanz-Struktur hilfreich sein. Es ist möglich die Zyklen als einzelne Komponenten zu betrachten, welche möglicherweise wiederum von weiteren Entscheidungen beziehungsweise Komponenten abhängen. Solche Komponenten lassen sich wie folgt definieren.

Definition 4.6

Die Mengen $\mathcal{S}_i$ von Knoten in einem gerichteten Graphen G, bilden eine Menge von *starken Zusammenhangskomponenten* (*strongly connected components*, *SCCs*), falls für jedes Paar von Knoten $v_i \neq v_j \in \mathcal{S}_i$ ein gerichteter Pfad von v_i zu v_j existiert, wobei $v_i \neq v_j$ und $S_i \nsubseteq S_j \forall j$ mit $j \neq i$.

Abbildung 4.4(a) zeigt ein Beispiel für ein simultanes ARA-Szenario, bei dem der Verteidiger und der Angreifer beim ersten Zug keine Kenntnis über die Maßnahmen des Gegenspielers haben. Anschließend kann der Verteidiger mit dem Wissen über den Angriff und der angewendeten Verteidigung noch weitere Gegenmaßnahmen einleiten. Der zugehörige Relevanz-Graph ist mit den starken Zusammenhangskomponenten in Abbildung 4.4(b) dargestellt. Durch diesen ist deutlich erkennbar, dass nur die erste Verteidigung und der Angriff voneinander abhängen. Die zweite Verteidigung kann unabhängig dazu berechnet werden, da zu diesem Zeitpunkt die Ausgänge der beiden anderen Entscheidungen feststehen.



(a) ARA-Modell

(b) Zugehöriger Relevanz-Graph mit den starken Zusammenhangskomponenten

Abbildung 4.4: Beispiel für Starke Zusammenhangskomponenten

Die Bestimmung von starken Zusammenhangskomponenten kann in linearer Zeit geschehen, zum Beispiel mit Hilfe der Tiefensuche [KN05]. Aus den gewonnen Zusammenhangskomponenten kann im Anschluss ein *Komponentengraph* erzeugt werden, welcher eine Kante zwischen Komponenten enthält, wenn im zugehörigen Relevanz-Graphen zwei Knoten aus den jeweiligen Komponenten existieren, welche jeweils miteinander verbunden sind. Der Relevanz-Graph ist somit laut Konstruktion azyklisch. Daher kann auch wieder eine topologische Sortierung $\mathcal{C}_1, \ldots, \mathcal{C}_m$ des Komponentengraphes erzeugt werden, welche die strategische Relevanz der einzelnen Komponenten widerspiegelt. Mit dieser neuen Konstruktion kann die Grundidee von Algorithmus 4.1 wieder angewandt werden.

Die Eingabe von Algorithmus 4.2 bildet ein azyklischer Komponenten-Graph eines zugehörigen MAID und die daraus resultierende topologische Sortierung der Komponenten $\mathcal{C}_1, \ldots, \mathcal{C}_m$. Zunächst wird in Zeile 1 ein beliebiges gemischtes Strategieprofil σ^0 erzeugt. Im Anschluss erfolgt die Traversierung der topologischen Sortierung in Schritt 2. Für jede einzelne Komponente $\mathcal{C}_i$ wird in Schritt 3 ein Nash-Gleichgewicht berechnet. In Schritt 4 wird das Strategieprofil mit der neuen Entscheidungsregel aktualisiert. Dieser Vorgang wird für jede Komponente $\mathcal{C}$ durchgeführt. Bei Terminierung enthält das aktuelle Strategieprofil σ^m ein Nash-Gleichgewicht für $\mathcal{M}$.

Algorithmus 4.2: Berechnung des Nash-Gleichgewichtes (Zyklisch)

Input : Given a MAID $\mathcal{M}$, a topological ordering $\mathcal{C}_1, \ldots, \mathcal{C}_m$ of the component graph derived from the relevance graph for $\mathcal{M}$

1 Let σ^0 be an arbitrary fully mixed strategy profile for $\mathcal{M}$
2 **for** $i = 1$ *to* m **do**
3 Let τ be a partial strategy profile for $\mathcal{C}_i$ that is a Nash equilibrium in $\mathcal{M}[\sigma^{i-1}_{-\mathcal{C}}]$
4 Let $\sigma^i = (\sigma^{i-1}_{-\mathcal{C}_i}, \tau)$
5 Output σ^m as an equilibrium of $\mathcal{M}$

Die Berechnung in Schritt 3 des Algorithmus 4.2 ist nicht trivial und verwendet eine Unterprozedur bei der das MAID in einen *Spielbaum* konvertiert wird. Im Anschluss werden bereits bekannte Lösungsalgorithmen aus der Spieltheorie angewendet (vergleiche [MM96]). Dieses Vorgehen ist aber dennoch eine Steigerung der Effizienz zu bisherigen Ansätzen, da die Spielbäume aufgrund der zu betrachtenden Komponenten in den meisten Fällen kleiner sind als das komplette MAID.

5 Lösungsansatz für ARA

In diesem Kapitel wird der entwickelte Lösungsansatz für ARA-Modelle vorgestellt (Abschnitt 5.1). Es werden prinzipiell zwei Fälle unterschieden. Der zyklische und azyklische Fall. Die Anwendbarkeit des Algorithmus wird mit Hilfe zweier Fallstudien demonstriert, welche in Abschnitt 5.2 vorgestellt werden.

5.1 Entwickelter Algorithmus

Der entwickelte Algorithmus basiert auf dem in Kapitel 4 vorgestellten Algorithmus von Daphne Koller und Brian Milch. Auch dieser arbeitet auf der Grundlage der strategischen Relevanz von Entscheidungen. Im Gegensatz zu der vorgestellten Lösungsstrategie im Bereich der MAIDs verfügen die einzelnen Spieler über eine divergierende Sicht auf die Instanz (vergleiche Abschnitt 3.4).

Definition 5.1

Ein AID $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ besitzt zuzüglich der MAID-Definition für n Spieler n mögliche Perspektiven, welche sich durch die Instanziierung der Wahrscheinlichkeitstabellen unterscheiden können.

Im Folgenden werden ARA-Instanzen für zwei Spieler betrachtet. Einen Angreifer $\mathcal{A}_A$ und einen Verteidiger $\mathcal{A}_D$. Dem Verteidiger ist es stets möglich, sein Modell konkret zu füllen, wobei der Angreifer auch durch Wahrscheinlichkeitsverteilungen modelliert werden kann. Der entwickelte Algorithmus arbeitet in zwei Phasen. Zunächst wird das Angreiferverhalten approximiert. Dies geschieht durch Einnahme der jeweiligen Angreiferperspektive $\mathcal{A}_A$. Ist das ungefähre Verhalten des Angreifers bekannt kann im nächsten Schritt die optimale Verteidigungsstrategie unter dem approximierten Angreiferverhalten berechnet werden. Beide entwickelten Algorithmen arbeiten auf Grundlage des Relevanz-Graphen. Da das Verhalten des Verteidigers bei Phase 1 des Algorithmus mit einer Wahrscheinlichkeitsverteilung modelliert wird und bei Phase 2 die Strategie σ_A für den Angreifer bekannt ist, arbeiten beide Spieler nur auf der topologischen Sortierung ihrer Perspektive.

Definition 5.2

Sei $(D_1, \ldots D_n)$ eine topologische Sortierung des Relevanz- oder des Komponenten-Graphen für ein AID $\mathcal{A}$. Dann bildet die Perspektive auf dieser topologische Sortierung eine Abfolge $(D_1, \ldots, D_m)$ mit $m \leq n$ für einen Spieler a, welche für jedes Element mindestens einen Entscheidungsknoten des Spielers a beinhaltet.

5.1.1 Azyklischer Fall

Auch bei dem entwickelten Ansatz wird erneut zwischen dem zyklischen und azyklischen Fall unterschieden. Zunächst wird ein Algorithmus vorgestellt, welcher auf dem azyklischen Relevanz-Graphen basiert.

Algorithmus 5.1: Approximieren des Angreifers (Azyklisch)

Input : AID $\mathcal{A}$ with an acyclic relevance graph, a topological ordering D_1,...,D_n of the Attacker's relevance graph for $\mathcal{A}$ and number of iterations

1 **for** $k = 1$ *to iterations* **do**
2 Draw Utility values $f_u^k = (u_1, \dots u_{|\mathcal{U}_A|}) \sim (U_1^A, \dots, U_{|\mathcal{U}_A|}^A) = F_U$
3 Draw System probabilities $f_c^k = (c_1, \dots, c_{|\mathcal{X}|}) \sim (C_1^A, \dots, C_{|\mathcal{X}|}^A) = F_C$
4 Draw Beliefs about D $f_d^k = (d_1, \dots, d_{|\mathcal{D}_D|}) \sim (D_1^A, \dots, D_{|\mathcal{D}_D|}^A) = F_D$
5 Let $\mathcal{A}^k$ be the current instance in regard to f_u^k and f_c^k
6 Let σ_D^0 be the beliefs f_d^k and σ_A^0 be a arbitraty fully mixed profile
7 Let $\sigma^0 = (\sigma_D^0, \sigma_A^0)$
8 **for** $i = 1$ *to* n **do**
9 Compute pure decision rule δ_{i_k} for D_i which is optimal for $\mathcal{A}^k_{[\sigma^{i-1}]}$
10 Update $\sigma^i = (\sigma^{i-1}_{-D_i}, \delta_i)$
11 $\forall d \in D_i : \delta_i(d|\text{pa}) = \sum_{\forall k} \delta_{i_k}(d|\text{pa})/\textit{iterations}\ \forall i \in [1, \dots, n], \forall \text{pa} \in dom(Pa(D_i))$
12 Output $\sigma_A = (\delta_1, \dots, \delta_n)$ as aprroximated strategy for the Attacker

Algorithmus 5.1 erwartet zunächst eine topologische Sortierung des Relevanz-Graphen für die Angreiferperspektive für das zugehörige AID $\mathcal{A}$. Aus dieser ist die Optimierungsreihenfolge der einzelnen Entscheidungen abzuleiten. Von der Iterationsanzahl hängt die Güte des approximierten Angreiferverhaltens ab. Um das Verhalten des Angreifers vorherzusagen, wird für jede Iteration eine konkrete Instanz des AID Modells $\mathcal{A}_A$ ausgewertet - for-Schleife in Zeile 1. Die Wahrscheinlichkeitsverteilungen beschreiben die Werte des Angreifers (Zeile 2), die Wahrscheinlichkeiten des Systems (Zeile 3) und das zu erwartende Verhalten des Verteidigers (Zeile 4). Die konkrete Ziehungen bilden eine AID Instanz A^k (Zeile 5). Das initiale Strategieprofil σ^0 wird durch gemischte Entscheidungsregeln für den Angreifer σ_A^0 und Wahrscheinlichkeitsverteilungen für das Verhalten des Verteidigers σ_D^0 gebildet (Zeile 7). Im Anschluss wird in einer weiteren for-Schleife (Zeile 8) für jede Entscheidung des Angreifers die optimale Entscheidungsregel δ_{i_k} für die AID Instanz $\mathcal{A}^k$ unter dem aktuellen Strategieprofil σ^{i-1} berechnet (Zeile 9). Die gewonnene optimale Entscheidung wird anschließend in Zeile 10 dem Strategieprofil hinzugefügt. Dieser Vorgang wird für jede Entscheidung des Angreifers durchgeführt. Um nun das Verhalten des Angreifers zu approximieren, wird über jeden Entscheidungsprozess in Zeile 11 gemittelt. Die resultierende Strategie σ_A (Zeile 12) beinhaltet folglich das approximierte Verhalten über die Perspektive $\mathcal{A}_A$.

Algorithmus 5.2: Berechnung der optimalen Verteidigung (Azyklisch)

Input : AID $\mathcal{A}$ with an acyclic relevance graph, a topological ordering D_1,...,D_n of the Defender's relevance graph for $\mathcal{A}$ and the approximated strategy σ_A for the Attacker

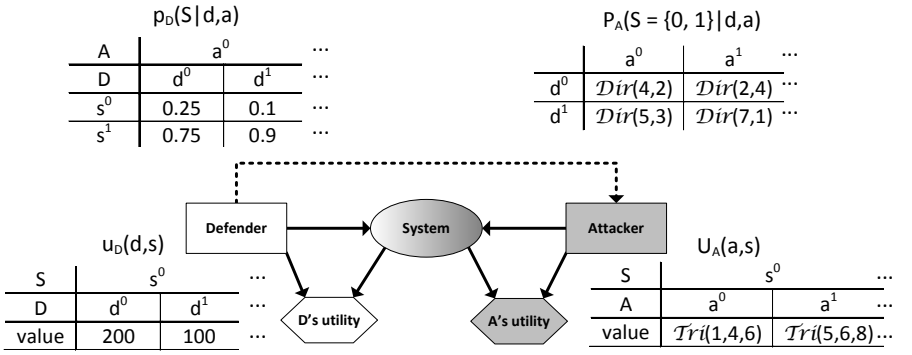
1 Let $\sigma^0 = (\sigma_A, \sigma_D)$ with σ_D fully mixed profile for the Defender
2 **for** $i = 1$ *to* n **do**
3 Compute decision rule δ for D_i that is optimal for σ^{i-1}
4 Update $\sigma^i = (\sigma^{i-1}_{-D_i}, \delta_i)$
5 Output σ^n as optimal strategy profile for the Defender

Der nächste Schritt der Analyse befasst sich mit der Wahl der optimalen Verteidigungsstrategie. Das Angreiferverhalten wurde durch Algorithmus 5.1 berechnet und dient als Eingabe für Algorithmus 5.2. Dieses wird durch die Strategie σ_A ausgedrückt und fließt in das initiale Strategieprofil σ^0 ein (Zeile 1). Die verbleibenden Entscheidungen vom Verteidiger werden durch gemischte Entscheidungsregeln ausgedrückt. Anschließend wird in Zeile 2 die topologische Sortierung des Relevanz-Graphen aus Sicht des Verteigers traversiert. Für jede Entscheidung innerhalb der Sortierung wird die optimale Entscheidung für das aktuelle Strategieprofil σ^{i-1} berechnet (Zeile 3) und im Folgenden Schritt dem neuen Strategieprofil hinzugefügt (Zeile 4). Dieser Vorgang wird für jede Entscheidung der Sortierung durchgeführt. Bei Terminierung der Schleife bildet das finale Strategieprofil σ^n die optimale Entscheidungsfolge des Verteidigers unter dem approximierten Angreiferverhalten.

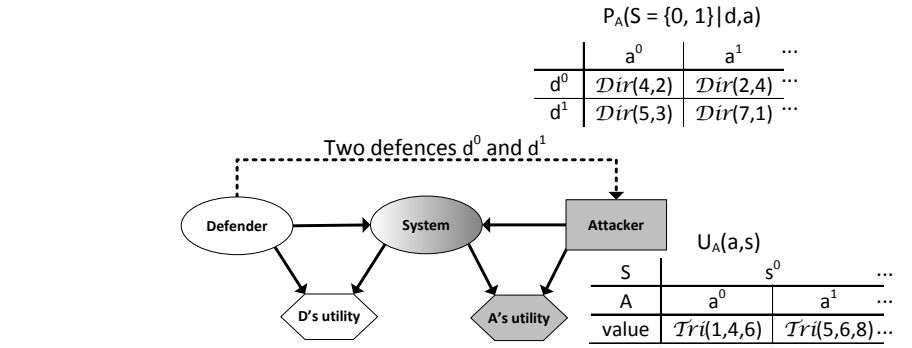
Beispieldurchführung

Um die Vorgehensweise des Algorithmus zu veranschaulichen, wird in Abbildung 5.1 eine beispielhafte Durchführung betrachtet. Das gezeigte Szenario beschreibt das sequentielle Verteidigungs-Angriffsmuster (vergleiche Abschnitt 5.2.1). Bei diesem wählt der Verteidiger zunächst eine Verteidigungsstrategie, welche anschließend in die Entscheidung des Angreifers einfließt. Abbildung 5.1(a) illustriert das Ausgangsmodell, welches vom Modellierer erstellt wird. Dabei ist zu erkennen, dass zwei Wahrscheinlichkeitstabellen für den Knoten `System` existieren. Diese stellen die unterschiedlichen Sichtweisen dar. Der Verteidiger besitzt eine konkrete Sicht und kennt damit die genauen Auswirkungen auf den Systemzustand ($p_D(S|d,a)$), wohingegen die Sicht des Angreifers durch die *Dirichlet-Verteilung* beschrieben wird ($P_A(S = \{0,1\}|d,a)$). Die Wertigkeiten der beiden Spieler werden mit den Tabellen $u_D(d,s)$ und $U_A(a,s)$ beschrieben, wobei dem Verteidiger die Werte genau bekannt sind. Der Angreifer wird erneut über eine Verteilung, hier die *Triangular-Distribution*, dargestellt. Im nächsten Schritt beginnt der eigentliche Analyseablauf. Zunächst wird das Verhalten des Angreifers approximiert (siehe Algorithmus 5.1). Dies geschieht, indem die Sicht des Angreifers eingenommen wird (vergleiche Abbildung 5.1(b)). Dabei existiert nur noch ein Spieler, der Verteidiger wird dabei als *Chance*-Knoten dargestellt.

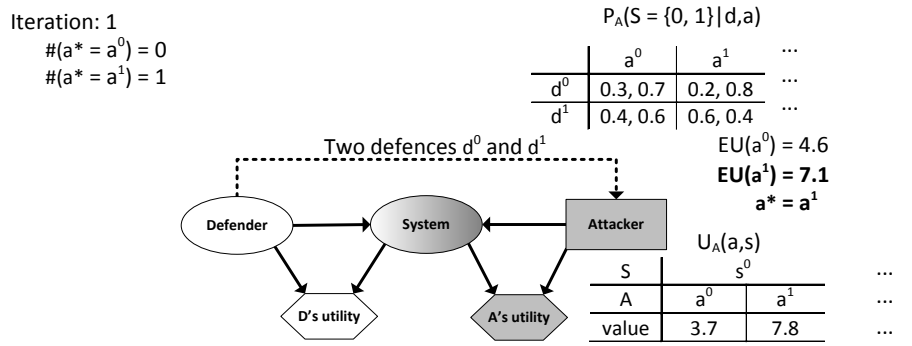
(a) Ausgangsmodell



(b) Sicht des Angreifers einnehmen



(c) Beste Entscheidung für eine Stichprobe berechnen



(d) Optimale Entscheidung für den Verteidiger berechnen

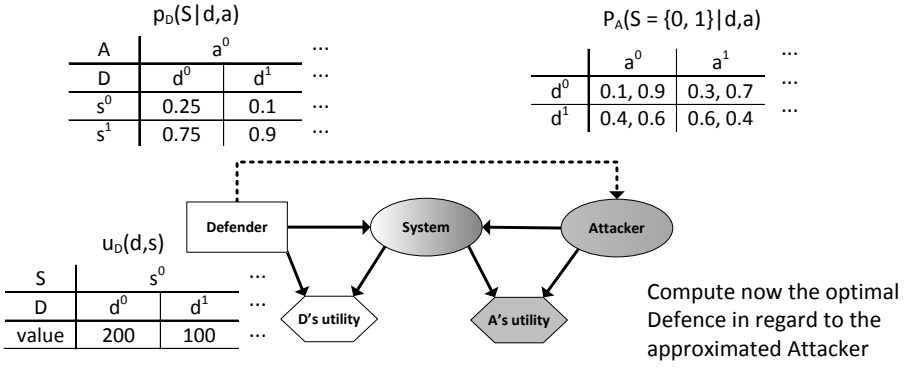
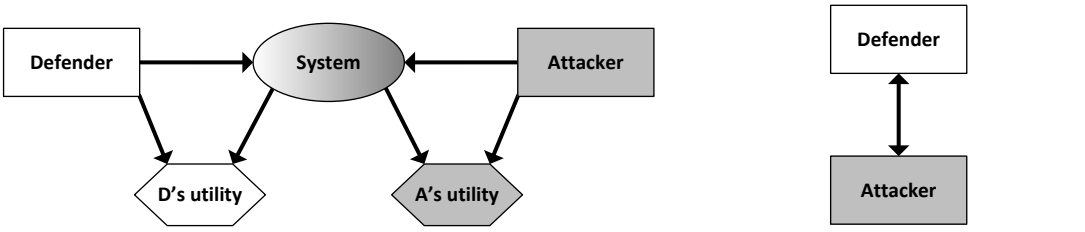


Abbildung 5.1: Beispielhafte Analyse eines AID-Modelles

An dieser Stelle fließen die Annahmen des Angreifers über das Verhalten des Verteidigers in die Analyse ein. Um jede möglichen Ablauf der Verteidigung zu betrachteten, kann beispielsweise angenommen werden, dass die Verteidigungsstrategien d^0 und d^1 gleich wahrscheinlich sind. Im nächsten Schritt (Abbildung 5.1(c)) werden Stichproben der AID-Instanz gezogen. Die Annahmen über das Verhalten des Verteidiger ändern sich nicht, da diese mit konkreten Werten belegt wurden (0.5 und 0.5). Nun sind seitens des Angreifers auch konkrete Werte verfügbar, welche zur Berechnung der optimalen Entscheidung genutzt werden. In diesem Fall gibt es zwei Entscheidungen a^0 und a^1. Da der Erwartungswert von a^1 höher ist als von a^0, bildet die optimale Entscheidung a^* für diese Instanz a^1. Der Zähler für die optimalen Entscheidungen kann dementsprechend aktualisiert werden. Dieser Schritt wird über eine angegebene Anzahl von Iterationen wiederholt. Ist dieser Arbeitsschritt beendet, wird das approximierte Verhalten für den Angreifer bestimmt. Dies geschieht, indem die Häufigkeit der Optimalität einer Entscheidung innerhalb eines Entscheidungsknotens gezählt und durch die Iterationszahl dividiert wird. Das Ergebnis ist eine valide Wahrscheinlichkeitsverteilung für den Entscheidungsknoten. Diese ermittelte Wahrscheinlichkeitstabelle wird für die Berechnung der optimalen Verteidigungsstrategie benötigt (Algorithmus 5.2) und fließt in die Abbildung 5.1(d) ein. Ähnlich zu dem vorherigen Analyseschritt wird erneut eine Perspektive eingenommen. In diesem Fall die des Verteidiger. Der Angreifer wird dementsprechend als Chance-Knoten mit der zuvor ermittelten Wahrscheinlichkeitsverteilung dargestellt. Mit diesem Wissen ist es dem Verteidiger nun möglich, die optimale Verteidigungsstrategie zu berechnen.

5.1.2 Zyklischer Fall

In dem vorherigen Abschnitt wurde ein Algorithmus vorgestellt, welcher den azyklischen Fall des Relevanz-Graphen abdeckt. Es kann allerdings auch vorkommen, dass der Relevanz-Graph Zyklen enthält und daher Algorithmus 5.1 nicht mehr anwendbar ist. Dies ist zum Beispiel bei dem Simulatenen ARA-Modell in Abbildung 5.2 der Fall. Der zugehörige Relevanz-Graph weist zwei Entscheidungsknoten auf, welche beide voneinander strategisch abhängen.



(a) Simultanes ARA-Modell (b) Zughöriger Relevanz-Graph

Abbildung 5.2: Beispielszenario für einen zyklischen Relevanz-Graphen

Zum Lösen solcher Strukturen wird auf eine abgewandelte Berechnungsvorschrift des simultanen Verteidigungs-Angriffsmusters zurückgegriffen. Algorithmus 5.3 beschreibt zunächst die Lösungsstrategie die seitens des Secomics-Projekt entwickelt wird [Río+13a]. Dabei ist das Ausgangsmodell analog zu 5.2. Da die Algorithmen speziell auf die Modelle abgestimmt sind, muss an dieser Stelle erwähnt werden, dass der Verteidiger, wie auch der Angreifer, über zwei Aktionen verfügen. Das System kann ebenfalls zwei Zustände annehmen. Zur allgemeinen Vorgehensweise ist zu erwähnen, dass zunächst eine Berechnung der Entscheidung des Verteidigers erfolgt. Dies geschieht allerdings aus der Sicht des Angreifers. Die entsprechenden Wertigkeiten des Verteidigers und das voraussichtliche Verhalten des Angreifers werden hierbei wieder durch Distributionen ausgedrückt. Mit dieser Berechnung ist es möglich, das Verhalten des Verteidigers aus der Sicht des Angreifers zu erhalten. Anschließend wird mit dieser approximierten Verhaltensstruktur des Verteidigers erneut eine Analyse ausgeführt, welche das Verhalten des Angreifers approximiert. Mit diesem Entscheidungsprofil kann im Anschluss dann die optimale Verteidigungsstrategie des Verteidigers berechnet werden.

Algorithmus 5.3: Lösen des simultanes Verteidigungs-Angriffsmuster

1 **for** $k = 1$ *to* $\#iterations$ **do**
2 Draw $\pi_D^k \sim \Pi_D = \mathcal{U}(0,1)$
3 Draw $(u_D^k, p_D^k) \sim (U_D, P_D) = G$
4 Compute $d^k = \max_{d \in \mathcal{D}} \sum_{a \in \mathcal{A}} [\sum_{s \in \mathcal{S}} u_D^k(d,s) p_D^k(S = s|d,a)] \pi_D^k(A^1 = a)$
5 Approximate $\pi_A(D = d_1)$ through
$\hat{\pi}(D = d_1) = \#\{1 \leq k \leq \#iterations : d^k = d_1\}/\#itereations$
6 Set $\alpha = \hat{\pi}_A(D = d_1) \cdot 10$
7 Set $\hat{\Pi}_A(D = d_1) \sim Beta(\alpha, 10 - \alpha)$
8 **for** $j = 1to\#iterations$ **do**
9 Draw $\hat{\pi}_A^k \sim \hat{\Pi}_A$
10 Draw $(u_A^k, p_A^k) \sim (U_A, P_A) = F$
11 Compute $a^k = \max_{a \in A} \sum_{d \in \mathcal{D}} [\sum_{s \in \mathcal{S}} u_A^k(a,s) p_A^k(S = s|d,a)] \hat{\pi}_A^k(D = d)$
12 Approximate $\pi_D(A = a_1)$ through
$\hat{\pi}_D(A = a_1) = \#\{1 \leq k \leq \#iterations : a^k = a_1\}/\#iterations$

In der ersten for-Schleife (Zeile 1) des Algorithmus 5.3 wird das Verteidigerverhalten aus der Sicht des Angreifers berechnet. Hierbei wird das voraussichtliche Verhalten des Angreifers mit einer Gleichverteilung beschrieben ($\mathcal{U}(0,1)$). Die geschätzten Wertigkeiten und das Verhalten des Systems aus Sicht des Angreifers für den Verteidiger fließen in die Wahrscheinlichkeitsverteilungen $(U_D, P_D) = G$ ein (Zeile 3). Mit den gezogenen Stichproben aus den erwähnten Verteilungen wird jeweils die optimale Strategie des Verteidigers berechnet (Zeile 4). Das approximierte Verhalten $\hat{\pi}(D = d_1)$ wird anschließend in Zeile 5 bestimmt. Dies wird für den nächsten Analyseschritt noch weiter verrauscht (Zeile 6 und 7) und fließt dann in die Analyse des Angreiferverhaltens ein (ab Zeile 8). Ähnlich zu der vorherigen Analyse wird erneut das optimale Verhaltensprofil berechnet. Diesmal allerdings mit der Verteidigungsstragie und dem Ziel das Angreiferverhalten zu approxmieren. Sind die optimalen

Angreiferentscheidungen für jede unterschiedliche Instanz berechnet (Zeile 11), kann das approxmierte Angreiferverhalten in $\hat{\pi}(A = a_1)$ berechnet werden (Zeile 12). Mit diesem Profil ist es dann möglich, die optimale Verteidigung zu berechnen.
Dieses Vorgehen erlaubt eine Auflösung der zyklischen Abhängigkeit, bei der zwei Entscheidungen der jeweiligen Spieler voneinander abhängen. Zur Erkennung dieses Musters kann der Relevanz-Graph verwendet werden. Dazu werden die starken Zusammenhangskomponenten des zugehörigen Relevanz-Graphen berechnet. Dieser enthält nach Definition keinen Zyklus und daher kann eine topologische Sortierung berechnet werden. Der nachfolgende Algorithmus 5.4 beschreibt eine Möglichkeit, AID-Modelle mit einem zyklischen Relevanz-Graphen auszuwerten.

Algorithmus 5.4: Approximieren des Angreifers (Zyklisch)

Input : AID $\mathcal{A}$, a topological ordering $\mathcal{C}_1, \ldots, \mathcal{C}_m$ of the Attacker's component graph derived from the relevance graph of $\mathcal{A}$ and number of iterations

1 **for** $k = 1$ *to iterations* **do**
2 Draw Utility values $f_u^k = (u_1, \ldots u_{|\mathcal{U}_A|}) \sim (U_1^A, \ldots, U_{|\mathcal{U}_A|}^A) = F_U$
3 Draw Utility values $g_u^k = (u_1, \ldots u_{|\mathcal{U}_D|}) \sim (U_1^D, \ldots, U_{|\mathcal{U}_D|}^D) = G_U$
4 Draw System probabilities $f_c^k = (c_1, \ldots, c_{|\mathcal{X}|}) \sim (C_1^A, \ldots, C_{|\mathcal{X}|}^A) = F_C$
5 Draw Beliefs about D $f_d^k = (d_1, \ldots, d_{|\mathcal{D}_D|}) \sim (D_1^A, \ldots, D_{|\mathcal{D}_D|}^A) = F_D$
6 Let $\mathcal{A}^k$ be the current instance in regard to g_u^k, f_u^k and f_c^k
7 Let σ_D^0 be the beliefs f_d^k and σ_A^0 be a arbitraty fully mixed profile
8 Let $\sigma^0 = (\sigma_D^0, \sigma_A^0)$
9 **for** $i = 1$ *to* m **do**
10 **if** *SizeOf*($\mathcal{C}_i$) ***is*** 1 **then**
11 Compute pure δ_{i_k} for $D_i = \mathcal{C}_i$ which is optimal for $\mathcal{A}^k_{[\sigma^{i-1}]}$
12 **if** *SizeOf*($\mathcal{C}_i$) ***is*** 2 **then**
13 Let δ_A be a fully mixed decision rule for $D_A \in \mathcal{C}_i$
14 Compute pure δ_D for $D_D \in \mathcal{C}_i$ which is optimal for $\mathcal{A}^k_{[\sigma^{i-1}, \delta_A]}$
15 Compute pure δ_{i_k} for $D_A \in \mathcal{C}_i$ which is optimal for $\mathcal{A}^k_{[(\sigma^{i-1}, \delta_D)]}$
16 Update $\sigma^i = (\sigma^{i-1}_{-D_i}, \delta_i)$
17 $\forall d \in D_i : \delta_i(d|\text{pa}) = \sum_{\forall k} \delta_{i_k}(d|\text{pa})/iterations \ \forall i \in [1, \ldots, n]$, $\forall \text{pa} \in dom(Pa(D_i))$
18 Output $\sigma_A = (\delta_1, \ldots, \delta_n)$ as aprroximated strategy for the Attacker

Eingabe des Algorithmus 5.4 bildet die topologische Sortierung des Komponenten-Graphen für den Angreifer. Das heißt, es werden nur die Komponenten betrachtet, welche auch eine Entscheidung dieses Spielers beinhalten. Aus dieser Sortierung ist die Optimierungsreihenfolge der einzelnen Komponenten abzuleiten. Ähnlich zu Algorithmus 5.1, werden konkrete Instanzen des Modells ausgewertet. Es werden also Ziehungen der einzelnen Distributionen durchgeführt, welche im Anschluss der aktuellen AID Instanz zugeordnet werden (Zeile 2 bis 8). Der Unterschied zu dem bisherigen Ansatz bildet die Modellierung der Wertigkeiten des Verteidigers aus der Sicht des Angreifers (G_U). Im Anschluss werden in der weiteren for-Schleife (Zeile 9) die einzelnen Komponenten ausgewertet. Besteht die Komponente nur aus einer Entscheidung, kann die vorherige Lösungsstrategie angewandt werden (Zeile 10).

Beinhaltet die Komponente zwei Entscheidungen, handelt es sich automatisch um eine Entscheidung des Angreifers und des Verteidigers (Zeile 13). In diesem Fall wird zunächst die Verteidigungsstragie mit dem Wissen des Angreifers berechnet. Dabei fließen die Wertigkeiten des Verteidigers aus Sicht des Angreifers in die Analyse, sowie die Wahrscheinlichkeitsfunktionen des Angreifers. Die Entscheidungsregel für den Angreifer wird über δ_A modelliert. Ist die Entscheidungsregel für den Verteidiger berechnet (Zeile 15), kann im nächsten Schritt die Entscheidungsregel des Angreifers berechnet werden (Zeile 16). Dieser Vorgang wird für jede Komponente des Angreifers durchgeführt. Um nun das Verhalten des Angreifers zu approximieren, wird über jeden Entscheidungsprozess in Zeile 17 gemittelt. Die resultierende Strategie σ_A (Zeile 18) beinhaltet folglich das approximierte Angreiferverhalten.

Algorithmus 5.5: Berechnung der optimalen Verteidigung (Zyklisch)

Input : AID $\mathcal{A}$ with an cyclic relevance graph, a topological ordering $\mathcal{C}_1$,...,$\mathcal{C}_m$ of the Defender's component graph for $\mathcal{A}$ and the partial approximated strategy σ_A for the Attacker

1 Let $\sigma^0 = (\sigma_A, \sigma_D)$ with σ_D fully mixed profile for the Defender
2 **for** $i = 1$ *to* m **do**
3 **if** *SizeOf*($\mathcal{C}_i$) ***is*** 1 **then**
4 Compute pure decision rule δ_i for $D_i = \mathcal{C}_i$ which is optimal for σ^{i-1}
5 Update $\sigma^i = (\sigma^{i-1}_{-D_i}, \delta_i)$
6 **if** *SizeOf*($\mathcal{C}_i$) ***is*** 2 **then**
7 Compute pure decision rule δ_i for $D_D \in \mathcal{C}_i$ which is optimal for σ^{i-1}
8 Update $\sigma^i = (\sigma^{i-1}_{-D_i}, \delta_i)$
9 Output σ^n as optimal strategy profile for the Defender

Im nächsten Schritt der Analyse wird die optimale Verteidigungsstrategie für das approximierte Angreiferverhalten berechnet. Als Eingabe des Algorithmus 5.5 dient neben dem Komponentengraphen des Verteidigers die Strategie des Angreifers (σ_A). Die Berechnung gleicht dem vorherigen Vorgehen zur Berechnung der optimalen Verteidigung (siehe Algorithmus 5.2), nur das auf Komponenten gearbeitet wird.
Die vorgestellten Algorithmen zur Berechnung einer optimalen Strategie auf dem zyklischen Relevanz-Graphen behandeln derzeit nur Komponenten der Größen zwei und eins. Es ist aber durchaus möglich, dass größere Komponenten existieren. Für diese kann leider keine allgemeingültige Berechnungsvorschrift angegeben werden, ohne sämtliche Entscheidungsmöglichkeiten auszuwerten. Dabei ist zu erwähnen, dass der entwickelte Algorithmus von Daphne Koller und Brian Milch in diesem Fall auf externe Berechnungsmechanismen zurückgreift. Eine Entwicklung einer ähnlichen Unterprozedur scheint erforderlich. Ein anderer Weg bildet die Einarbeitung weiterer Templates, wie in dem Fall der Komponentengröße 2, welche allerdings auch nur schwer eine allgemeine Zusammenstellung abdecken können.

5.2 Fallstudien

Dieser Abschnitt widmet sich zwei Fallstudien um die Anwendbarkeit des zuvor entwickelten Algorithmus nachzuvollziehen. Die ausgewählten Szenarien stammen aus dem Seconomics Projekt [Col+13; Cas+13; Río+13a]. Bei beiden Modellen handelt es sich um eine vereinfachte Modellierung der tatsächlichen Annahmen. An der eigentliche Analyse ändert dies allerdings nichts, da nur die Datensätze verkleinert wurden. In Abschnitt 5.2.1 wird zunächst auf ein sequentielles Verteidigungs-Angriffsmuster eingegangen. Anschließend wird in Abschnitt 5.2.2 die umgekehrte Situation betrachtet.

5.2.1 Sequentielles Verteidigungs-Angriffsmodell

Die erste Fallstudie wird mit Hilfe eines sequentiellen Verteidigungs-Angriffsmodelles gelöst. Bei diesem wird angenommen, dass der Verteidiger zunächst seine Wahl bezüglich der Verteidigung trifft, woraufhin der Angreifer reagieren kann.
Dieses Szenario widmet sich den U-Bahn Stationen von Barcelona, welche von asozialem Verhalten betroffen sind. Verschiedene Arten von Vandalismus, wie Graffiti oder zerkratze und zerstörte Scheiben wurden von Angestellten der U-Bahn Gesellschaft dokumentiert. Die Taten konnten *Anti-System-Gruppen* zugeordnet werden, welche es auf die gezielte Zerstörung von Installationen und fremden Eigentums abgesehen haben. Um dies in Zukunft zu unterbinden beziehungsweise einzudämmen werden vier verschiedene Strategien d_1, d_2, d_3, d_4 betrachtet. Es ist der U-Bahn Gesellschaft möglich, mehr Sicherheitspersonal einzustellen. Konkret wird ein Personalzuwachs von 10, 20, 40 oder 50 Sicherheitsleuten betrachtet. Falls diesbezüglich eine Auswahl getroffen ist, muss davon ausgegangen werden, dass der Angreifer zwangsläufig über die Handlung des Verteidigers informiert wird. Dies kann zum Beispiel über die Medien geschehen, oder einer schlichten Beobachtung über häufiger patrouillierende Sicherheitsleute. Zum generellen Verhalten des Angreifers wird angenommen, dass dieser den größtmöglichen Schaden anrichten möchte. Dies kann in Form von Attackieren des Sicherheitspersonals, der Züge, wie auch anderen Installationen geschehen. Dabei stehen dem Angreifer drei gesteigerte Angriffsmöglichkeiten a_1, a_2, a_3 zur Verfügung. Ein Angreifer kann seine Hände für die Beschädigung verwenden (a_1). Das zweite Angriffsszenario (a_2) sieht neben den Händen auch physische Waffen, wie Steine oder Hämmer vor, um beispielsweise die Scheiben zu beschädigen. Die dritte Variante (a_3) erweitert beide vorangegangen Möglichkeiten um chemische Waffen, wie Molotowcocktails, Farben oder Säuren. Der Ausgang eines Angriffs wird in drei unterschiedliche Kategorien unterteilt. Zunächst kann ein Angriff in einem Fehlschlag enden ($s = 0$), dabei ist kein nennenswerter Schaden zu dokumentieren. Ein Angriff kann jedoch auch erfolgreich sein. Der Zustand $s = 1$, beschreibt einen Ausgang mit mäßigem Schaden und $s = 2$ mit schwerem Schaden.

Dem Verteidiger ist es möglich die Daten aus Tabelle 5.1 zu erheben. Dabei beschreibt die Wertefunktion $u_D(d, s)$ (Tabelle 5.1(a)) die steigenden Sicherheitskosten und weitere mögliche Einflussfaktoren. Die möglichen Ausgänge werden in Abhängigkeit zu den Entscheidungen des Angreifers und des Verteidigers modelliert (siehe Tabellen 5.1(b) und 5.1(c)). Die Wahrscheinlichkeitstabelle $p_D(S = 0|d, a)$ ist dadurch implizit gegeben.

Tabelle 5.1: Annahmen des Verteidigers (VA-Szenario)

(a) $u_D(d, s)$

	$s = 0$	$s = 1$	$s = 2$
d_1	200	50	10
d_2	100	20	10
d_3	80	10	0
d_4	50	0	0

(b) $p_D(S = 1|d, a)$

	a_1	a_2	a_3
d_1	0.25	0.45	0.5
d_2	0.2	0.25	0.35
d_3	0.15	0.2	0.25
d_4	0.1	0.15	0.2

(c) $p_D(S = 2|d, a)$

	a_1	a_2	a_3
d_1	0.3	0.35	0.4
d_2	0.25	0.3	0.35
d_3	0.2	0.25	0.3
d_4	0.05	0.1	0.15

(d) $U_A(a, s)$

	$s = 0$	$s = 1$	$s = 2$
a_1	0	$Tri(50, 60, 80)$	$Tri(80, 100, 100)$
a_2	0	$Tri(40, 50, 60)$	$Tri(60, 80, 90)$
a_3	0	$Tri(25, 40, 50)$	$Tri(60, 70, 90)$

(e) $P_A(S = \{0, 1, 2\}|d, a)$

	a_1	a_2	a_3
d_1	$Dir(4.5, 2.5, 3)$	$Dir(2, 4.5, 3.5)$	$Dir(1, 5, 4)$
d_2	$Dir(5.5, 2, 2.5)$	$Dir(4.5, 2.5, 3)$	$Dir(3, 3.5, 3.5)$
d_3	$Dir(6.5, 1.5, 2)$	$Dir(5.5, 2, 2.5)$	$Dir(4.5, 2.5, 3)$
d_4	$Dir(8.5, 1, .5)$	$Dir(7.5, 1.5, 1)$	$Dir(6.5, 2, 1.5)$

Der Angreifer wird über eine ermittelte Verteilung $(U_A, P_A) \sim F$ beschrieben (Tabellen 5.1(e) und 5.1(d)). Dabei werden die Wahrscheinlichkeiten von $P_A(S = \{0, 1, 2\}|d, a)$ durch eine *Dirichlet*-Verteilung $Dir(\alpha_1, \alpha_2, \alpha_3)$ ausgedrückt. Die geschätzten Wertigkeiten $U_A(a, s)$ mit Hilfe der *Triangular*-Verteilung $Tri(min, mode, max)$.

Um das Entscheidungsproblem des Verteidigers zu lösen, muss zunächst die Wahrscheinlichkeitsfunktion $\hat{p}_D(a_j|d_i)$ ermittelt werden. Diese gibt dem Verteidiger Aufschluss über die Reaktion des Angreifers, also welcher Angriff unter einer gegebenen Verteidigung gewählt wird, beziehungsweise wie wahrscheinlich diese ist. Mit diesem Wissen kann im Anschluss die optimale Verteidigungsstrategie berechnet werden. Algorithmus 5.6 zeigt eine Berechnungsvorschrift auf, um die benötige Wahrscheinlichkeitsfunktion $\hat{p}_D(a_j|d_i)$ mittels *Monte Carlo*-Verfahren zu approximieren.

Algorithmus 5.6: Bestimmung von $\hat{p}_D(a_j|d_i)$ für jedes d_i

1 **for** $i = 1$ *to* $\#Defences$ **do**
2 **for** $k = 1$ *to* $\#Simulations$ **do**
3 **for** $j = 1$ *to* $\#Attacks$ **do**
4 Draw $(u_A^k, p_A^k) \sim (U_A, P_A) = F$ for all outcomes $s \in \mathcal{S}$
5 Compute $\psi_A^k(d_i, a_j) = \sum_{s \in \mathcal{S}} p_A^k(s|d_i, a_j) u_A^k(a_j, s)$
6 Compute $a^* = argmax_{x \in \mathcal{A}} \psi_A^k(d_i, x)$
7 $\hat{p}_D(a^*|d_i) = \hat{p}_D(a^*|d_i) + 1$
8 $\hat{p}_D(a_j|d_i) = \hat{p}_D(a_j|d_i)/\#\text{Simulations}, \forall j$

Dafür wird für jede mögliche Wahl der Verteidigung (Zeile 1) das Angriffsverhalten approximiert. Dies geschieht mit einer vorgegeben *Simulationsanzahl* (Zeile 2). In der dritten `for`-Schleife wird in Zeile 4 zunächst eine Stichprobe aus der ermittelten Verteilung `F` gezogen. Für die konkreten Ziehungen (u_A^k, p_A^k) werden dann in Zeile 5 alle möglichen Erwartungswerte seitens des Angreifers ermittelt. Da von einem maximierenden Verhalten ausgegangen werden kann, wird in Zeile 6 der Angriff ausgewählt, welcher den höchsten Erwartungswert besitzt. Die einzelnen Verhaltensweisen werden in $\hat{p}_D$ in Zeile 7 gesammelt und anschließend in Zeile 8 durch die Anzahl der Simulationen gemittelt. Die Variable $\hat{p}_D$ enthält somit bei Terminierung eine konkrete Wahrscheinlichkeitsverteilung.

Mit der ermittelten Distribution $\hat{p}_D$ kann nun im nächsten Schritt die optimale Verteidigungsstrategie mit Algorithmus 5.7 berechnet werden. Zunächst wird in Zeile 1 jede mögliche Verteidigungsstrategie betrachtet und in Zeile 2 unter Hinzunahme der zuvor ermittelten Wahrscheinlichkeitsfunktion $\hat{p}_D(a_j|d_i)$, der Systemfunktion $p_D(s|d_i, a_j)$ und der Wertefunktion $u_D(d_i, s)$ die einzelnen Erwartungswerte $\psi_D(d_i)$ berechnet. Die optimale Verteidigung wird durch den höchsten Erwartungswert in Zeile 3 ermittelt.

Algorithmus 5.7: Berechnung der optimalen Verteidigung

1 **for** $i = 1$ *to* $\#Defences$ **do**
2 Compute $\psi_D(d_i) = \sum_{j=1}^{n} \hat{p}_D(a_j|d_i) \left[\sum_{s \in S} p_D(s|d_i, a_j)\right] u_D(d_i, s)$
3 Compute $d^* = argmax \psi_D(d_i)$

Die Ergebnisse des Seconomics Projektes und des entwickelten Werkzeuges sind in Tabelle 5.2 dargestellt. Die Anzahl der Simulationen betrug jeweils 10.000. Tabelle 5.2(a) zeigt die approximierte Funktion $\hat{p}_D(a|d)$ des Seconomics Projektes und 5.2(d) die daraus resultierende optimale Verteidigung. Das Ergebnis des Werkzeuges bezüglich der Funktion $\hat{p}_D(a|d)$ ist in Tabelle 5.2(b) dokumentiert und die dazu gehörende optimale Verteidigung in Tabelle 5.2(d). Insgesamt lassen sich nur kleine Abweichungen feststellen, sodass die Ergebnisse als reproduzierbar angenommen werden können.

Tabelle 5.2: Auswertung der Analyse (VA-Szenario)

(a) $\hat{p}_D(a|d)$ Paper

	a_1	a_2	a_3
d_1	0.25	0.4	0.35
d_2	0.31	0.28	0.41
d_3	0.29	0.3	0.41
d_4	0.16	0.31	0.53

(b) $\hat{p}_D(a|d)$ Tool

	a_1	a_2	a_3
d_1	0.25	0.4	0.35
d_2	0.33	0.27	0.4
d_3	0.3	0.3	0.4
d_4	0.13	0.32	0.55

(c) Optimum Paper

$d^* = d_1$

(d) Optimum Tool

$d^* = d_1$

5.2.2 Sequentielles Angriffs-Verteidigungsmodell

Die zweite Fallstudie wird mit Hilfe eines sequentiellen Angriffs-Verteidigungsmodelles gelöst. Bei diesem wird angenommen, dass der Angreifer zunächst seine Wahl bezüglich des Angriffs trifft, woraufhin der Verteidiger reagieren kann.
Dieses Szenario widmet sich dem *UK National Grid*, welches durch terroristische Aktivitäten bedroht wird. Hierfür sind seitens der Regierung zwei wesentliche Infrastrukturen zu schützen: Gas und Strom. Die terroristische Vereinigung hat insgesamt vier verschiedene Entscheidungsmöglichkeiten a_1, a_2, a_3, a_4. Das erste Angriffsszenario (a_1) beschreibt einen konzentrierten Angriff auf die Gas Infrastruktur. Die zweite Möglichkeit a_2 bildet die selbe Angriffsstärke auf die Strom-Infrastruktur. Es kann ebenfalls mit a_3 kein Angriff erfolgen. Falls beide Infrastrukturen Ziel des Angriffs werden, kann dieser nicht mit der gleichen Intensität wie die Angriffe a_1 oder a_2 erfolgen, dies bildet Möglichkeit a_4. Der Regierung, beziehungsweise dem Verteidiger, stehen drei unterschiedliche Aktionen zur Verfügung (d_1, d_2, d_3), um die Schäden, welche durch den Angreifer verursacht werden, einzudämmen. Die erste Variante d_1 bildet einen Plan mit niedrigen, d_2 mit mittleren und d_3 mit hohen Kosten. Je höher die Investitionen, desto schneller und effektiver ist es möglich den Ursprungszustand wiederherzustellen. Ein kostenintensiver Plan belastet jedoch gleichzeitig das zur Verfügung stehende Budget. Werden jedoch keine Ausgaben bezüglich der Verteidigung gemacht, kann es im Falle eines erfolgreichen Angriffs zu erheblichen Einbußen führen. Der Erfolg (1) und Misserfolg (0) eines Angriffs wird jeweils durch die binäre Variable S beschrieben. Dem Verteidiger ist es möglich die Daten aus Tabelle 5.3 zu erheben. Dabei beschreibt die Wertefunktion $u_D(d, s)$ (Tabelle 5.3(a)) die Wertigkeiten der gewählten Verteidigungsstrategie in Abhängigkeit zum Resultat eines Angriffs. Die möglichen Ausgänge werden in Abhängigkeit zu den Entscheidungen des Angreifers und des Verteidigers modelliert (siehe Tabelle 5.3(b)). Die Wahrscheinlichkeitstabelle $p_D(S = 0|d, a)$ ist dadurch implizit gegeben.

Tabelle 5.3: Annahmen des Verteidigers (AV-Szenario)

(a) $u_D(d, s)$

	$s = 0$	$s = 1$
d_1	100	20
d_2	80	10
d_3	50	0

(b) $p_D(S = 1|d, a)$

	a_1	a_2	a_3	a_4
d_1	0.7	0.6	0	0.3
d_2	0.4	0.3	0	0.1
d_3	0.05	0.05	0	0.05

(c) $U_A(a, s)$

	$s = 0$	$s = 1$
a_1	0	$Tri(60, 80, 90)$
a_2	0	$Tri(50, 70, 80)$
a_3	0	0
a_4	0	$Tri(45, 60, 70)$

(d) $P_A(S = 1|d, a)$

	a_1	a_2	a_3	a_4
d_1	$Be(6, 4)$	$Be(6, 4)$	0	$Be(4, 6)$
d_2	$Be(4, 6)$	$Be(4, 6)$	0	$Be(3, 7)$
d_3	$Be(2, 8)$	$Be(2, 8)$	0	$Be(1, 9)$

(e) $P_A(d|a)$

	a_1	a_2	a_3	a_4
$\{d_1, d_2, d_3\}$	$Dir(2, 3, 5)$	$Dir(2, 3, 5)$	$Dir(7, 2, 1)$	$Dir(2.5, 3.5, 4)$

Der Angreifer wird über eine ermittelte Verteilung $(U_A, P_A) \sim F$ beschrieben (Tabellen 5.3(d) und 5.3(c)). Dabei werden die Wahrscheinlichkeiten von $P_A(S = \{0, 1\}|d, a)$ durch eine *Beta*-Verteilung $Beta(\alpha, \beta)$ ausgedrückt. Die geschätzten Wertigkeiten $U_A(a, s)$ werden erneut mit Hilfe der *Triangular*-Verteilung $Tri(min, mode, max)$ beschrieben.

Die Ergebnisse des Seconomics Projektes und des entwickelten Werkzeuges sind in Tabelle 5.4 dargestellt. Die Anzahl der Simulationen betrug jeweils 10.000. Tabelle 5.4(a) zeigt die approximierte Funktion $\hat{p}_D(a|d)$ des Seconomics Projektes und 5.4(c) die daraus resultierende optimale Verteidigung. Das Ergebnis des Werkzeuges bezüglich der Funktion $\hat{p}_D(a|d)$ ist in Tabelle 5.4(b) dokumentiert und die dazu gehörende optimale Verteidigung in Tabelle 5.4(c). Insgesamt lassen sich nur kleine Abweichungen feststellen, sodass die Ergebnisse als reproduzierbar angenommen werden können.

Tabelle 5.4: Auswertung der Analyse (AV-Szenario)

(a) $p_D(a|d)$ Paper

	a_1	a_2	a_3	a_4
$p_D(a)$	0.5	0.47	0	0.03

(b) $p_D(a|d)$ Tool

	a_1	a_2	a_3	a_4
$p_D(a)$	0.52	0.44	0	0.04

(c) Optimum Paper

	a_1	a_2	a_4
$d^*(a)$	d_2	d_2	d_1

(d) Optimum Tool

	a_1	a_2	a_4
$d^*(a)$	d_2	d_2	d_1

6 Entwicklung der Toolunterstützung

Das Kapitel *Entwicklung der Toolumsetzung* gibt einen einführenden Einblick in die Umsetzung des prototypischen Werkzeugs. Dabei werden zunächst in Abschnitt 6.1 die zugrunde liegenden Technologien vorgestellt. Im nachfolgenden Abschnitt 6.2 wird die Entwicklung der wesentlichen Komponenten beleuchtet. Der letzte Abschnitt 6.3 widmet sich der Qualitätssicherung des entstandenen Programmes.

6.1 Grundlagen

Das entwickelte Werkzeug setzt auf bestehenden Frameworks auf. Um eine bessere Einsicht zu erhalten, werden die wesentlichen Komponenten und Hintergründe in diesem Abschnitt kurz vorgestellt. Dafür wird zunächst in Abschnitt 6.1.1 auf die Hauptkomponente *Eclipse* und die *Plug-In* orientierte Entwicklung eingegangen. Das zugrundeliegende *Entitätsmodell* wird mit dem *Eclipse Modeling Framework* erstellt, welches in Abschnitt 6.1.2 vorgestellt wird. Um einen grafischen Editor bereitzustellen, der eine benutzerfreundlichere Modellierungsmöglichkeit als vorhandene EMF-Editoren bietet, wird das Framework *Graphiti* (siehe Abschnitt 6.1.3) verwendet.

6.1.1 Eclipse und Plug-ins

Die Architektur von der *integrierten Entwicklungsumgebung* (IDE) Eclipse basiert seit der Version 3.0 auf dem *Equinox*-Kernel. Dieser wurde von der *Eclipse Foundation* als Java-Framework entwickelt und implementiert die *OSGi* Kernspezifikation, welche eine hardwareunabhängige und dynamische Softwareplattform beschreibt, die es ermöglicht, die Anwendungen per *Komponentenmodell* zu verwalten. Daher existiert Eclipse, seit der Version 3.0, selber nur noch als Kern, welcher die Funktionalitäten, in Form von *Plug-Ins*, nachlädt. Ein Plug-In ist im Allgemeinen eine abgeschlossene, modulare Softwarekomponente, welche während der Laufzeit entdeckt und eingebunden werden kann. Im weiteren Verlauf wird die Bezeichnung Plug-In im Eclipse-Kontext verwendet. In der Regel erweitert ein Eclipse Plug-In vorhandene Funktionalitäten. Dabei wird sukzessiv eine Abhängigkeitsstruktur aufgebaut, die in einer *Manifest*-Datei in *Extensible Markup Language* (*XML*) beschrieben wird. In dieser werden neben Plugin-Abhängigkeiten weitere Eigenschaften, wie der Name und die Versionsnummer festgelegt. Eclipse erlaubt neben der Standard Manifest Datei, die Deklaration einer *Plugin-Manifest*, ebenfalls in XML, welche die konkrete Plattformerweiterung, sowie weitere Abhängigkeiten beschreibt. Die *ExtensionPoints* dienen als Schnittstelle (*Application Programming Interface, API*), welche die Erweiterung von anderen Plug-Ins an diesen *ExtensionPoints* erlauben.

Dabei stellen *Extensions* das Gegenstück zu den ExtensionsPoints dar. Dieser Vorgang kann mit einem Kommunikationsvertrag zwischen den beiden Komponenten verglichen werden, bei denen die Plug-Ins somit nicht zwangsläufig in Gegenwart des Gegenstücks entwickelt werden müssen [Ecla].

6.1.2 Eclipse Modeling Framework

Das *Eclipse Modeling Framework (EMF)* ist ein Open Source Projekt der Eclipse Gemeinschaft und dient der automatischen Erzeugung von Java Code auf Grundlagen von strukturellen Modellen. Das resultierende Programm ist in der Lage *Instanzen* des zuvor definierten Modelles zu erzeugen, dieses abzufragen und zu manipulieren. Eine Möglichkeit der Serialisierung wird durch den *XML Metadata Interchange (XMI)* Standard geboten. Die Validierung der Instanzen ist ebenfalls mit dem generierten Quellcode einfach möglich. Die Definition der *Metamodelle* kann auf verschiedene Arten erfolgen. Zum einen werden UML-Modellierungstools wie *Rational Rose* unterstützt, welche auch den XMI-Standard verwenden um Modelle auszutauschen. Zum anderen ist auch eine Quellcode nahe Variante durch spezielle *Annotationen* und Definition von Java-*Interfaces* vorgesehen, welche im Anschluss durch EMF realisiert werden. Eine weitere Möglichkeit bildet die Definition per *XML Schema (XSD)*. Dabei vereinigt EMF die drei unterschiedlichen Vorgehensweisen um abstrakte Modelle zu beschreiben. Egal welcher Schritt bei der Definition der Modelle gewählt wird, es resultiert immer das EMF-Modell, welches die anderen beiden Ansätze verbindet. Das EMF-Modell wird auch als *ecore*-Modell bezeichnet. Ecore selbst ist wieder ein EMF-Modell und daher auch sein eigenes *Metamodell*. Mit EMF ist es also möglich, einen modellbasierten Entwicklungsweg (*Model driven development, MDD)* einzuschlagen. Allerdings bietet das Framework im weiteren Vorgehen auch Möglichkeiten den resultierenden Quellcode manuell weiter anzupassen. Dabei ergänzen sich beide Entwicklungswege. Somit ist es möglich, weitere Modellanpassungen vorzunehmen und im Anschluss wieder Quellcode zu generieren, ohne das die manuellen Änderungen verworfen werden. Damit bringt EMF viele Vorteile des modellgetriebenen Entwicklungsprozesses mit, liefert aber auch gleichzeitig eine Flexibilität bei der Implementierung. Das Eclipse Modeling Framework ist folglich als modellgetriebener Entwicklungsansatz zu verstehen, bietet dem Programmierer aber auch Möglichkeiten manuelle Änderungen vorzunehmen - ein Mittelweg zwischen den Extremen des reinen Programmierens und Modellierens. Weitere Vorteile bestehen in der Integration weiterer Eclipse Werkzeuge um beispielsweise schneller Anwendungsschnittstellen (*User Interfaces, UI*) zu entwerfen. Ein Beispiel hierfür bildet Graphiti (siehe Abschnitt 6.1.3). Somit steigert das Eclipse Modeling Framework die Produktivität bei der Entwicklung und vermindert gleichzeitig Fehlerquellen [Eclb].

6.1.3 Graphiti

Das Eclipse Modeling Framework bietet bereits eine Möglichkeit Instanzen innerhalb eines *Baumeditors* zu erstellen und zu bearbeiten. Die Modellgröße, viele Referenzen und die Komplexität mancher Metamodelle beeinträchtigen jedoch schnell die Lesbarkeit der Modelle. Die Verwendung einer echten 2-dimensionalen Benutzerober-

fläche stellt in vielen Fällen eine Vereinfachung dar. Graphiti ist ein weiteres Eclipse Projekt, welches eine schnelle und hochwertige Möglichkeit bietet, grafische Editoren für *Domainmodelle* zu entwerfen. Dabei können EMF-basierte Modelle, sowie frei implementierte Java-Objekte, als Domainmodell fungieren. Die Hauptkomponenten von Graphiti bilden zwei weitere Eclipse Projekte: Das *Graphical Editing Framework (GEF)* und *Draw2D* um grafische Diagramme zu zeichnen. Konkrete Kenntnisse dieser beiden Komponenten sind allerdings nicht erforderlich. Es werden ausschließlich Kenntnisse in der Entwicklung mit Graphiti und EMF benötigt. Sämtliche *Rendering*-Funktionalität wird von Graphiti übernommen. Eine Separierung des Diagrammcodes und des entworfenen Modelles, welches mit dem Diagrammcode verbunden ist, sorgt für eine zusätzliche Mobilität. So ist es beispielsweise möglich, eine leichtgewichtige Renderingkomponente zur Verfügung zu stellen, die lediglich das Diagramm anzeigt, ohne konkret mit dem eigentlichen Domainmodell arbeiten zu müssen. Graphiti selbst ist für ein schnelles *Prototyping* ausgerichtet. Viele Standardfunktionen stehen schon bei Beginn der Entwicklung zur Verfügung und können während des Entwicklungsprozesses weiter angepasst und verfeinert werden. Um Funktionen innerhalb des Editors zu Verfügung zu stellen, werden *Features* definiert. Der vollständige Lebenszyklus eines Modellelementes kann mit Hilfe dieser Features beschrieben werden. So werden Elemente beispielsweise durch ein *CreateFeature* erzeugt, falls das Modellelement bereits vorhanden ist und nur die grafische Repräsentation hinzugefügt werden soll, wird dies über das *AddFeature* realisiert. Ein Vorteil von Graphiti, welcher auch erneut das schnelle Prototyping unterstützt, ist eine große Menge von Standardfeatures, die bereits verwendet werden können. Im Laufe der Softwareentwicklung können diese dann sukzessiv angepasst, beziehungsweise ersetzt werden [Eclc].

6.1.4 SMILE und GeNIe

Die Bibliothek SMILE (*Structural Modeling, Inference, and Learning Engine*) bietet ein weitgehend plattformunabhängiges Werkzeug um wahrscheinlichkeits- und entscheidungstheorietische Modelle, wie Bayes'sche Netze zu analysieren. Die gesamte Funktionalität ist über eine SMILE API definiert, welche das Erstellen, Editieren, Speichern und Auswerten von Domainmodellen ermöglicht. SMILE ist in C++ implementiert, allerdings in einer plattformunabhägigen Weise. So werden alle gängigen Betriebssysteme (Mac OS, iOS, Android, Linux, Windows) abgedeckt. Standardmäßig wird SMILE unter Windows als *Dynamic Link Library* (*DLL*) zur Verfügung gestellt. Es existieren jedoch zahlreiche *Wrapper*, welche eine weitgehend unabhängige Anbindung gewährleisten sollen, wie SMILE.NET (.NET interface), SMILEX (Active X), und jSMILE (Java interface). Für diese Arbeit wurde auf den Wrapper jSMILE zurückgegriffen.
Das Windows-Werkzeug *GeNIe* (*Graphical Network Interface*), ist die grafische Benutzeroberfläche von SMILE und erlaubt es auf eine einfache Weise entscheidungstheoretische Modelle zu entwerfen und auszuwerten. GeNIe 2.0 ist die aktuelle Version von GeNIe und löste die Version 1.0 ab, welche im Jahr 1998 veröffentlicht wurde. SMILE, wie auch GeNIE, sind ein weit akzeptiertes Werkzeug, welches in der Forschung als auch in der Industrie Anwendung finden [Dec].

6.2 Konzeption und Implementierung

Dieser Abschnitt widmet sich der Konzeption des prototypischen Programmes. Dabei wird zunächst in Abschnitt 6.2.1 auf das *Entitätsmodell* eingegangen, welches zur Definition der Modelle verwendet wird. Die anschließenden Abschnitte 6.2.2 - 6.2.5 geben einen Einblick in die wesentlichen Komponenten. Die beispielhafte Implementierung zur Berechnung eines *Nash-Gleichgewichtes* kann in Abschnitt 6.2.6 nachvollzogenen werden. Der letzte Abschnitt 6.2.7 gibt ein Gesamtbild der Architektur.

6.2.1 Entitätsmodell

Ein *Entitätsmodell* beschreibt das zu serialisierende Datenmodell. Dieses wird entworfen, um eine einheitliche Definition des Modelles zu erhalten, welches dem Programm zur Verfügung steht.
EMF (siehe Abschnitt 6.1.2) eröffnet eine effiziente Möglichkeit ein individuelles Metamodell zu erstellen, welches zur Generierung der nötigen *Entitätsklassen* verwendet werden kann. Ein Vorteil des generierten Quellcodes bietet unter anderem die bereits mitgelieferte *Serialisierbarkeit*, sodass erstellte Modelle sofort gespeichert und wieder geladen werden können. Die Abbildung 6.1 zeigt das in EMF entworfene Entitätsmodell.

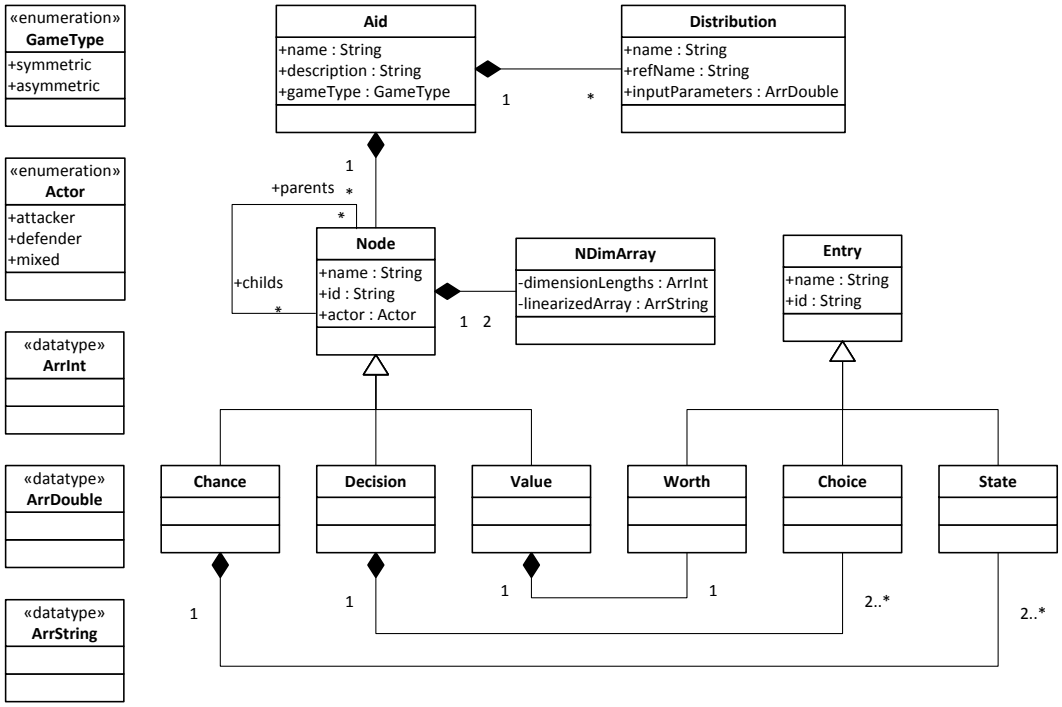


Abbildung 6.1: Entitätsmodell

Wurzelelement dieses Metamodelles ist die Klasse `Aid` (*Adversarial Influence Diagram*), welche als *Containerelement* für alle weiteren Elemente dient. Neben einem *Namen* (`name`) und einer *Beschreibung* (`description`) ist es möglich, den definierten *Spieltyp* (`gameType`) festzulegen. Das *Attribut* `gameType` ist ein *Aufzählungstyp*

(*Enumeration*) und kann die Werte *symmetrisch* (`symmetric`) und *asymmetrisch* (`asymmetric`) annehmen. Bei einem *symmetrischen* Spiel verfügen alle Akteure über dieselben Wahrscheinlichkeitstabellen, wohingegen ein asymmetrisches Spiel unterschiedliche Informationslagen der Akteure erlaubt (ARA). Da ARA-Modelle *Distributionen* beziehungsweise Wahrscheinlichkeitsverteilungen verwenden, müssen die verwendeten Wahrscheinlichkeitsverteilungen auch im Modell gespeichert werden. Dies wird mit der `Distribution`-Klasse gelöst. Ein `Distribution`-Objekt enthält neben dem *Namen* (`name`), der für die *Referenzierung* benutzt wird, auch eine Beschreibung des *Typs* (`refName`), also um welchen Typ von Distribution es sich handelt. Da eine konkrete Distribution in der Regel *Eingabeparameter* benötigt, werden diese im Attribut `inputParameters` hinterlegt. Ein Knoten innerhalb eines Graphen wird durch die Klasse `Node` definiert. Dieser enthält jeweils eine Referenzliste `parents` und `children` um die *Eltern-* und Kindbeziehungen aufzeigen zu können. Neben einem *Namen* und einer eindeutigen *ID* (`id`), enthält ein `Node`-Objekt ein Attribut `actor`, welches den Knoten zu einem Spieler zuordnet. Derzeit werden drei unterschiedliche `Actor` Elemente unterstützt: `Attacker`, `Defender` und `Mixed`. Hierbei handelt es sich aber nur bei den Elementen `Attacker` und `Defender` um wirkliche Spieler, die bei `Decision` und `Value` Knoten frei gesetzt werden können. Ein `Chance`-Objekt ist immer vom Typ `mixed`. Dieser Typ kann keinem anderen Knoten zugeordnet werden. Die Klasse `Node` ist jeweils der Generalisierung der Klassen `Chance`, `Decision` und `Value`, welche die eigentlichen Elemente des `Aids` bilden. Ein `Chance`, wie auch ein `Decision`-Objekt, müssen jeweils zwei `Entries` enthalten, welche die Auswahlmöglichkeiten beschreiben. Wohingegen ein `Value`-Objekt immer nur ein `Worth`-Objekt enthält. Ein `Entry`-Objekt, sowie alle Subklassen, enthalten einen *Namen* (`name`) und eine eindeutige *ID* (`id`). Die Wahrscheinlichkeitstabellen eines Knotens werden jeweils für den Angreifer und den Verteidiger innerhalb eines `NDim`-Objektes gespeichert. Die Wahrscheinlichkeitstabelle wird für diesen Schritt *linearisiert* (`linearizedArray`). Die Anordnung der Elternknoten, beziehungsweise die Dimensionengröße dieser, werden zur Wiederherstellung benötigt und im `dimensionLenghts` Objekt abgelegt.

6.2.2 Wahrscheinlichkeitstabelle

Die wesentlichen Analyseinformationen werden aus den Wahrscheinlichkeitstabellen jedes einzelnen Knotens gewonnen. Serialisiert werden diese im *Entitätsmodell* durch die `NDimArray` Klasse. Dabei wird die Tabelle in zwei Komponenten zerlegt. Eine eindimensionale Verkettung der Wahrscheinlichkeitseinträge (`linearizedArray`) und die Dimensionen der Elternknoten (`dimensionLenghts`), damit die Einträge in einem späteren Verlauf wieder korrekt referenziert werden können. Um einen effizienten Umgang mit den Wahrscheinlichkeitstabellen jedes Knotens zu gewährleisten, verfügt jede Tabelle über eine *Assoziationsklasse*, den `CPDManager`, welcher viele Funktionalitäten bündelt und einheitlich zur Verfügung stellt. Der `CPDManager` steht jedem `Node`-Objekt zur Verfügung und dient auch der automatischen Aktualisierung der Wahrscheinlichkeitstabelle, da diese in Abhängigkeit zu den Elternknoten steht. Um dies effizient zu Realisieren wird die Referenz `parents` auf Änderungen überwacht. Wird im Verlauf der Modellierung ein Knoten der Elternmenge hinzugefügt

oder entfernt, so passt der `CPDManager` die `Wahrscheinlichkeitstabelle` an, sodass diese sich immer in einem konsistenten Zustand befindet. Beim Löschen eines Elternknotens werden die zugehörigen Einträge der Tabelle entfernt. Wird ein Knoten der Elternmenge hinzugefügt, werden neue Spalten mit einer Standardbelegung eingefügt.

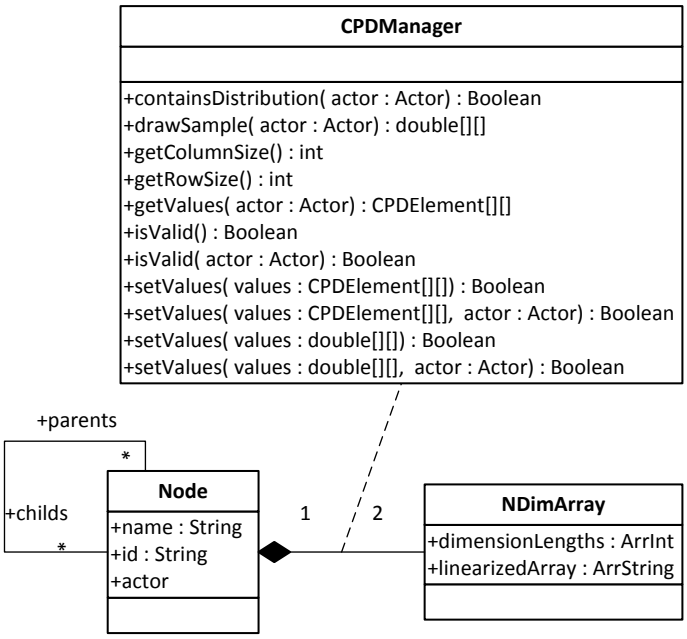


Abbildung 6.2: CPD Manager

Im Folgenden werden die angebotenen Methoden des `CPDManagers` skizziert. Es handelt sich hierbei um eine Auswahl, um die wesentlichen Funktionen zu illustrieren. Da die aktuelle Größe einer Wahrscheinlichkeitstabelle von den Elternknoten und den eigenen internen Zuständen abhängt, kann die Größe einer Tabelle über die Methoden `getColumnSize` und `getRowSize` ermittelt werden. Diese geben jeweils die Spalten- und Zeilengröße der Tabelle wieder. Die Elemente innerhalb einer Wahrscheinlichkeitstabelle werden durch die Klasse `CPDElement` beschrieben (siehe Abbildung 6.3). Die vollständige Tabelle eines Spielers kann dabei mit der Methode `getValues(actor)` ermittelt werden. Da die einzelnen Elemente neben konkreten Zahlenwerten auch Distributionen widerspiegeln, ist es mit der Methode `drawSample(actor)` möglich, eine Wahrscheinlichkeitstabelle eines Spielers auszuwerten, wodurch die Distributionen durch konkrete Ziehungen ersetzt werden. Ob eine Wahrscheinlichkeitstabelle eines Spielers überhaupt Distributionen enthält, kann mit der Methode `containsDistribution(actor)` geprüft werden. Um eine leichtere Validierung der Tabelle zu ermöglichen, verfügt der `CPDManager` über diverse Validierungsmethoden. Eine der Hauptmethoden bilden dabei die `isValid(actor)`, beziehungsweise die `isValid()` Methoden, welche für einen Spieler oder alle Spieler die Wahrscheinlichkeitstabelle auf Inkonsistenzen prüfen. So muss beispielsweise sichergestellt werden, dass die Spaltensumme immer eins ergibt und auch das jedes Element innerhalb einer Tabelle interpretiert werden kann. Dies betrifft insbesondere die Distributionen (vergleiche Abschnitt 6.2.3). Das Setzten von neuen Werten

ist jeweils mit einer der `setValues` Methoden möglich, wobei bei Nichtangabe eines Spielers, sämtliche Wahrscheinlichkeitstabellen mit den neuen Werten aktualisiert werden.

Wie zuvor erwähnt, werden die Elemente innerhalb einer Wahrscheinlichkeitstabelle jeweils durch eine Realsierung der abstrakten Klasse `CPDElement` beschrieben. Siehe dazu Abbildung 6.3. Derzeit sind zwei Realisierungen vorgesehen. Ein konkreter Wert innerhalb einer Tabelle wird durch ein Objekt der Klasse `CPDValueElement` definiert. Dies enthält ein Attribut `value`, welches den konkreten Wert widerspiegelt. Wohingegen eine Distribution durch ein `CPDDistributionElement` beschrieben wird. Dieses referenziert aus dem Enititätsmodell (Abbildung 6.1) ein `Distribution` Objekt, welches die Wahrscheinlichkeitsverteilung beschreibt. Der `outputIndex` dient zur Identifizierung des Ausgabeparameters der Distribution. Ob es sich bei der Distribution um eine Inverse handelt, also eine konkrete Ziehung der Verteilung minus eins, wird durch das `isInverse` Attribut bestimmt.

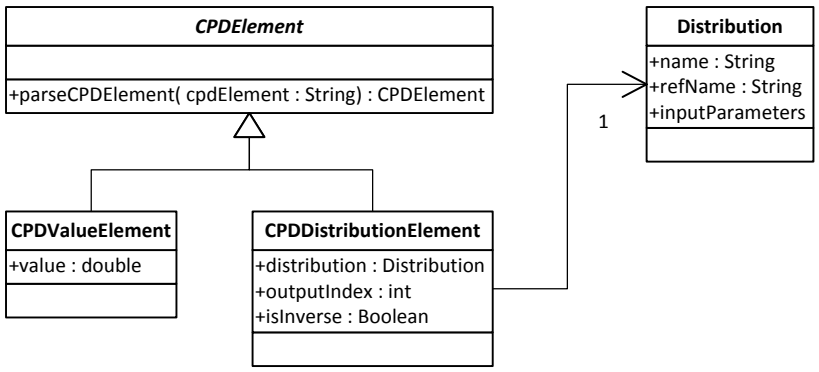


Abbildung 6.3: CPD Element

Das Einlesen eines Elementes wird durch die Methode `parseCPDElement` realisiert. Dabei bildet der Parameter `cpdElement` die `String`-Repräsentation eines konkreten `CPDElements`. Diese Repräsentation ist bei einem `CPDValueElement` kanonisch und wird durch den Wert selbst beschrieben. Ein `CPDDistributionElement` ist dagegen komplexer. Zunächst kann durch ein optionales *Ausrufezeichen* der gezogene Wert invertiert werden. Dieses Vorgehen kann zum Beispiel bei Knoten angewendet werden, welche zwei Zustände beinhalten und einer durch eine Verteilung beschrieben wird. Dadurch ist automatisch der zweite Eintrag durch die Inverse Distribution bestimmt. Nach dem optionalen Ausrufezeichen erfolgt die Referenzierung der Distribution mit Hilfe des angegebenen *Namens*. Anschließend folgt ein *Doppelpunkt*, welcher die Definition des Ausgabeindex kennzeichnet.
Abbildung 6.4 zeigt die Benutzeroberfläche, die zur Verwaltung einer Wahrscheinlichkeitstabelle dient. Fehlerhafte Eingaben und Inkonsistenzen werden bereits während der Eingabe farblich hervorgehoben. Interne Zustände eines Knotens, `States` und `Choices`, können durch den `AddEntry` Button hinzugefügt werden. Zur Entfernung eines Zustands wird die entsprechende Zeile innerhalb der *Main CPD* selektiert und anschließend durch den Klick auf den Button `Remove Entry` entfernt.

Die minimale Menge bei `Decision` und `Chance` Knoten ist dabei stets zwei. Um Distributionen zu verwalten, wird der `DistributionManager` verwendet. Siehe dazu Abschnitt 6.2.3. Um die Richtigkeit einer Wahrscheinlichkeitstabelle leichter zu verifizieren, kann eine Ziehung der Distributionen erfolgen. Das Ergebnis wird dabei in einem gesonderten Fenster dargestellt (`Sample CPD`). Bei teils redundanten Tabellen kann die `Unify Tables` Funktionalität verwendet werden, bei der die Wahrscheinlichkeitstabelle des Angreifers auf den Verteidiger dupliziert wird. Die Speicherung der Tabelle im `NDim` Objekt wird durch den `Save` Button veranlasst.

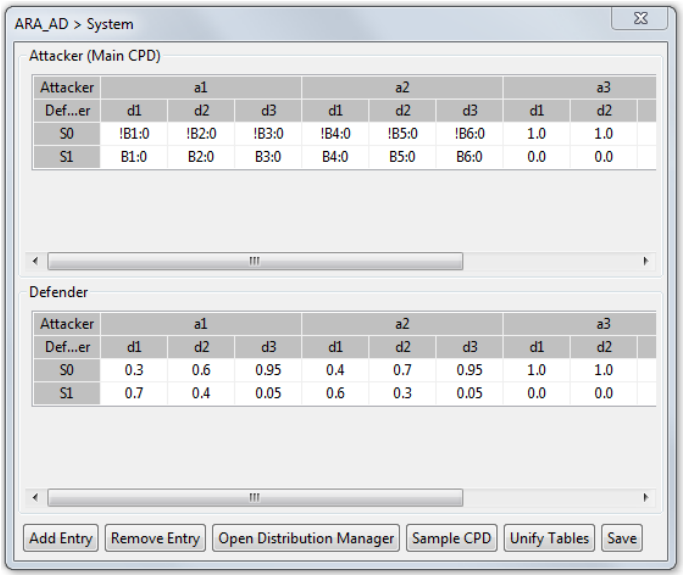


Abbildung 6.4: Benutzeroberfläche für die CPD

6.2.3 Wahrscheinlichkeitsverteilung

Das Wissen jedes einzelnen Spielers kann im Rahmen der ARA-Analyse divergieren. Um dennoch das ungefähre Wissen eines Spielers zu modellieren, werden *Distributionen*, beziehungsweise *Wahrscheinlichkeitsverteilungen*, eingesetzt. Eine Möglichkeit diese zu beziehen bilden statistische Erhebungen. Die Qualität der Analyse und damit das *approximierte* Verhalten eines Spielers, hängt von diesen Daten ab. Um in dem Werkzeug Distributionen zu unterstützen, wird ein `Interface` definiert, welches den gebrauchten Funktionsumfang für einen Analyseprozess beschreibt. Die Abbildung 6.5 illustriert das Interface mit den derzeitig realisierten Wahrscheinlichkeitsverteilungen.
Jede Distribution besitzt einen *Namen* (`name`), über welchen dieser mit dem Entitätsmodell (siehe 6.1) via `refName` verknüpft wird. Die Methode `getInputNames()` liefert die Namen der benötigen *Eingabeparameter*. Die `TriangularDistribution` benötigt beispielsweise die standardisierten Eingabeparameter `a`, `b` und `c`, wohingegen die `DirichletDistribution` nur einen Namen für den Eingabeparameter zurückgibt, nämlich a_i. Da die Anzahl der Namen der Eingabeparameter nicht zwangsläufig auf die akzeptierende Anzahl der Parameter hindeuten, gibt es die Methoden `getLowerInputSize` und `getUpperInputSize`. Diese beschreiben die jeweiligen Ober-

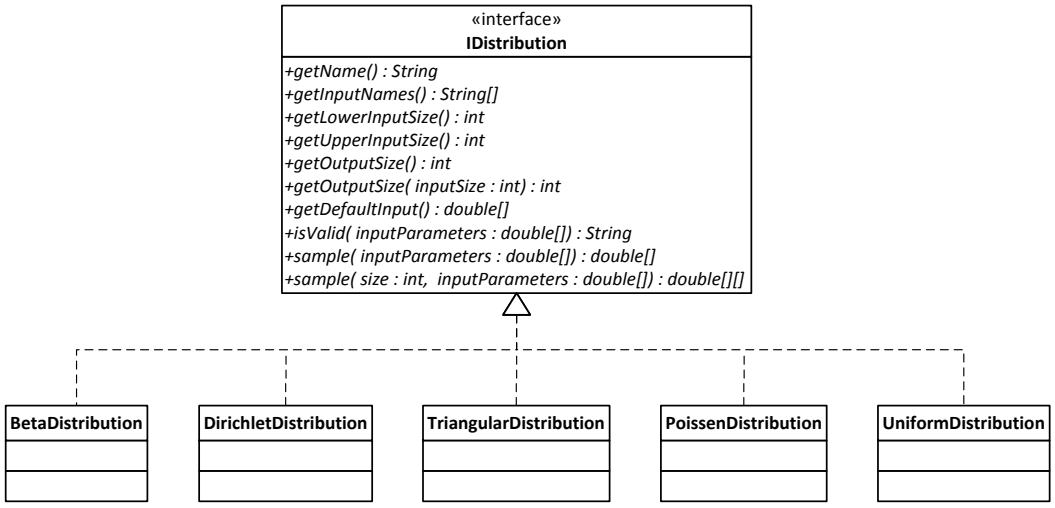
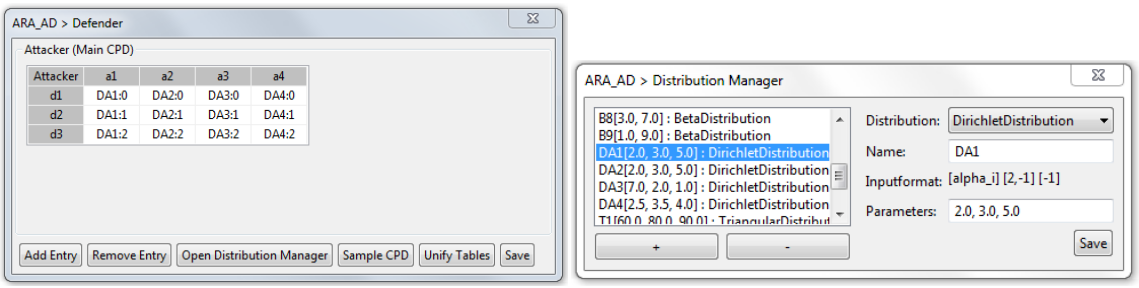


Abbildung 6.5: Interface und Realisierungen von Distributionen

und Untergrenzen. Falls keine Obergrenze existiert, ist der erwartete Rückgabewert `-1`. Im Falle einer *multivariaten Verteilung* oder auch mehrdimensionalen Verteilung, hängen die *Ausgabewerte* von den *Eingabewerten* ab. Es gibt also nicht immer eine fest definierte Anzahl von *Ausgabeparametern*. Um dies abzudecken, sind die Methoden `getOutputSize` und `getOutputSize(inputSize)` definiert. Letztere, um die Anzahl der Ausgabeparameter in Abhängigkeit zu den Eingabeparametern zu erhalten. Falls es sich um eine relative Anzahl von Ausgabeparametern handelt, ist der erwarte Rückgabewert bei `-1`. Die Methode `getDefaultInput` liefert stets eine konforme Belegung für die Verteilung. Um einen späteren Validierungsprozess zu vereinfachen, wurde die Methode `isValid` entworfen, welche eine Eingabebelegung auf dessen Gültigkeit überprüft. Bei der Realisierung dieser Methode wird beispielsweise die korrekte Anzahl der Parameter überprüft, sowie weitere Parameterbedingungen. Im Falle der `TriangularDistribution` muss beispielsweise die Einschränkung `a` $\leq$ `b` $\leq$ `c` erfüllt werden. Die beiden `sample` Methoden liefern jeweils konkrete Ziehungen zurück. Wobei `sample(size, inputParameters)` eine definierte Menge (`size`) an Ziehungen zurückgibt.



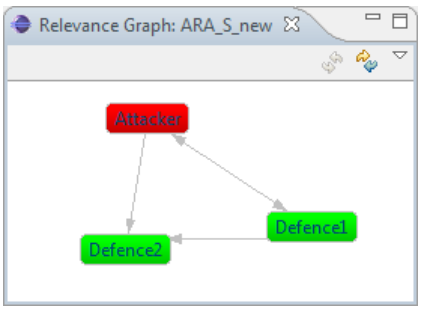
(a) CPD-Table (b) Distribution Manager

Abbildung 6.6: Verwendung von Distributionen im Modell

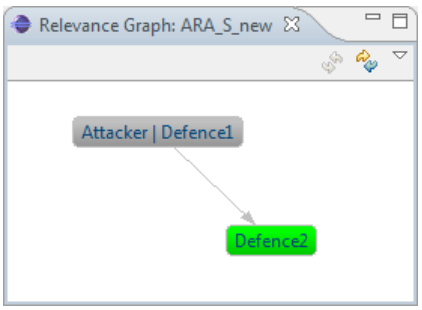
Die Distributionen werden mit Hilfe des *DistributionManagers* in das Modell eingepflegt (siehe Abbildung 6.6(b)). Dieser verfügt über alle im Programm registrierten Distributionen und bietet diese in der entsprechenden `ComboBox` an. Der frei wählbare Name `DA1` dient zur Referenzierung der angelegten Distribution innerhalb einer CPD-Table (siehe Abbildung 6.6(a)). Innerhalb einer Wahrscheinlichkeitstabelle ist die Distribution über den Namen und den Index des Ausgabeparameters abrufbar. So liefert der Ausdruck `DA1:0` den ersten Wert der `DirichletDistribution(2.0, 3.0, 5.0)`. Dabei wird innerhalb einer Wahrscheinlichkeitstabelle immer nur eine Ziehung für ein Verteilung durchgeführt, welche dann den entsprechenden Einträgen zur Verfügung steht. Dieses Vorgehen ist wichtig, da beispielsweise die `Dirichlet-Distribution` eine Wahrscheinlichkeitsbelegung zurückgibt. Die Summe aller Werte ergibt somit 1. Um diese Eigenschaft zu erhalten, erfolgt eine Zuordnung.

6.2.4 Relevanz-Graph

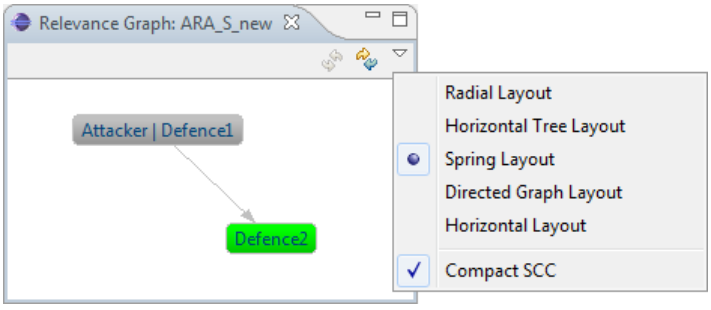
Der *Relevanz-Graph* ist eine wichtige Datenstruktur, welche während der Analyse zur Bestimmung der Optimierungsreihenfolge Verwendung findet. In Modellen kann so die Abhängigkeit zwischen einzelnen Entscheidungen effizient ermittelt werden. Da diese Informationen auch außerhalb der Analysephase von Vorteil sind und gerade bei größeren Modellen schnell Aufschluss über die Abhängigkeiten liefern, steht eine gesonderte Berechnungs- und Visualisierungsfunktion zur Verfügung. Diese kann innerhalb des Graphiti-Editors oder auch direkt auf einer Diagrammdatei innerhalb der Eclipse-Umgebung aufgerufen werden. Die folgenden Abbildungen demonstrieren die Visualisierungsmöglichkeiten.



(a) Relevanz-Graphen mit Zyklus



(b) Starken Zusammenhangskomponenten



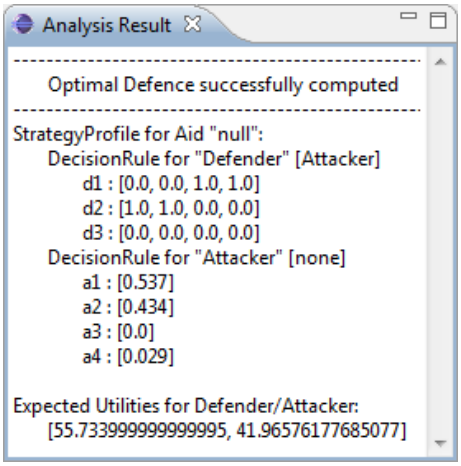
(c) Verschiedene Visualisierungen

Abbildung 6.7: Funktionsumfang des RelevanceGraphs

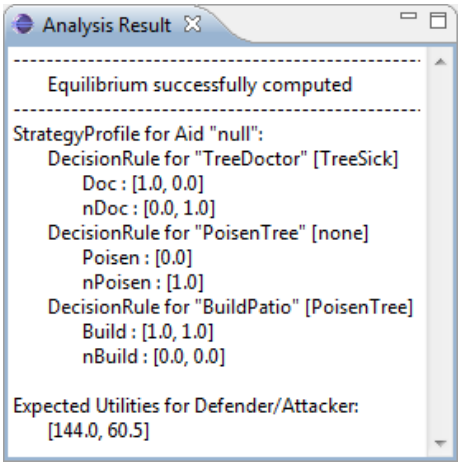
Abbildung 6.7(a) zeigt einen einfachen Relevanz-Graphen, bestehend aus drei Knoten. Für eine bessere Übersicht werden die Knoten bezüglich des Spielers eingefärbt. Nach Definition sind somit die Entscheidungen `Attacker` und `Defence1` strategisch relevant für die Entscheidung `Defence2`. `Attacker` und `Defence1` sind jedoch beide voneinander abhängig. Um die möglichen Abhängigkeiten innerhalb des Relevanz-Graphen zu verdeutlichen, können die *starken Zusammenhangskomponenten* auf diesem berechnet werden. Diese werden im Anschluss als einzelne Knoten dargestellt. Dabei werden die betroffenen Kanten verschmolzen. Abbildung 6.7(b) zeigt den Graphen aus Abbildung 6.7(a) mit starken Zusammenhangskomponenten. Die Entscheidungen `Attacker` und `Defence1` sind somit zu einem Knoten verschmolzen. Der visualisierte Relevanz-Graph kann für eine bessere Darstellung beliebig angeordnet werden. Um jedoch eine schnelle Übersicht zu liefern, bietet dieser neben der Zusammenfassung der starken Zusammenhanskomponenten (`Compact SCC`), vergleiche Abbildung 6.7(c), auch verschiedene Layout-Optionen. Eine Auswahl von mehreren Anordnungsmöglichkeiten ist gerade bei Modellen mit einer großen Knotenanzahl von Vorteil.

6.2.5 Analyse-Ergebnis

Das Ergebnis einer Hauptanalyse bildet immer ein *Strategieprofil*, welches die gewünschten Eigenschaften auf dem Modell erfüllt. Eine minimalistische textuelle Ausgabe dient nach dem erfolgreichem Abschluss einer Analyse als Ergebnisausgabe (siehe Abbildung 6.8).



(a) Analyse-Ergebnis für *Attack-Defend*



(b) Analyse-Ergebnis für *TreeKiller*

Abbildung 6.8: Textuelle Darstellung der Analyse-Ergebnisse

Bei der Textualisierung wird standardmäßig auf die `toString`-Methode des Strategie-Objektes zurückgegriffen, welche einen schnellen Überblick über die derzeitige Belegung ermöglicht. Die Abbildung 6.8(a) zeigt das Analyse-Ergebnis eines *ARA-Attack-Defend* Szenarios. Dabei bilden die Entscheidungen des Verteidigers immer reine Belegungen ($0 \dot{\vee} 1$), wohingegen der Angreifer approximiert wird und möglicherweise gemischte Entscheidungsregeln enthält ($[0 \ldots 1]$). Bei der Berechnung ei-

nes Nash-Gleichgewichtes (siehe Abbildung 6.8(b)) werden immer reine Belegungen für beide Seiten erzeugt. Die ausgegebenen Wahrscheinlichkeiten, für jeden internen Zustand, bilden zusammengesetzt wieder eine Wahrscheinlichkeitstabelle für den jeweiligen Entscheidungsknoten. Hierzu ist die Reihenfolge der Elternknoten zu beachten, welche nach dem Namen der Entscheidung, innerhalb der Ausgabe, angegeben wird. Die Darstellungsform der CPD-Tabelle ist analog zu Abbildung 6.2.

6.2.6 Implementierungsbeispiel: Nash-Gleichgewicht

Der folgende Abschnitt widmet sich einem Implementierungsbeispiel, um die Funktionalität der entwickelten Komponenten zu demonstrieren. Hierfür wird der vorgestellte Algorithmus zur Berechnung eines Nash-Gleichgewichtes für den azyklischen Fall ausgewählt, da dieser im Wesentlichen alle wichtigen Prozeduren enthält und gleichzeitig eine kompakte Darstellungsform liefert (siehe Quellcode 6.1).
Zunächst wird in Zeile 3 und 4 der Relevanz-Graph erzeugt. Auf dieser Grundlage kann die Optimierungsreihenfolge mit Hilfe einer topologischen Sortierung in Zeile 5 und 6 bestimmt werden. In Zeile 9/10 wird zunächst ein Strategieprofil für das `aidModel` erzeugt, für welches im Anschluss in Zeile 11 ein gemischtes Strategieprofil berechnet wird. Dies enthält also für jede Entscheidung eine Entscheidungsregel, in welcher wiederum jede Entscheidung im Sinne der Wahrscheinlichkeitstheorie möglich ist. In der nachfolgenden `for`-Schleife wird das aktuelle Strategieprofil immer angewandt (Zeile 15/16). Das resultierende `aidModel` ist eine Kopie der eingegebenen Instanz, wobei die Wahrscheinlichkeitstabellen mit den enthaltenen Entscheidungsregeln ersetzt werden. Da zur Berechnung der Erwartungswerte die SMILE Bibliothek (siehe Abschnitt 6.1.4) verwendet wird, müssen zunächst für die korrekte Berechnung der Erwartungswerte die konträren `Value`-Knoten entfernt werden, da diese im späteren Verlauf nicht mehr dem Spieler zugeordnet werden können. Dies geschieht in Abhängigkeit zu dem Spieler jeweils ab Zeile 17 oder 23.

```
1  public final void computeAcyclicEquilibrium() {
2    // Compute Topological Ordering
3    Graph<Decision> relevanceGraph =
4      BayesUtil.computeRelevanceGraph(aidModel);
5    List<Decision> topologicalOrdering =
6      GraphUtil.sortTopologically(relevanceGraph);
7
8    // Create arbitrary fully mixed strategy profile
9    AidStrategyProfile strategyProfile =
10     new AidStrategyProfile(aidModel);
11   strategyProfile.computeFullyMixedStrategyProfile();
12
13   // Find Optimum for every decision
14   for (Decision decision :  topologicalOrdering) {
15     Aid appliedAid = AidUtil
16           .applyStrategyProfile(aidModel, strategyProfile);
17     if (decision.getActor().equals(Actor.ATTACKER)) {
18       List<Value> valueNodes = AidUtil
```

```
19              .getValueNodes(appliedAid, Actor.DEFENDER);
20        for (int i = 0; i < valueNodes.size(); i++) {
21          EcoreUtil.delete(valueNodes.get(i));
22        }
23      } else {
24        List<Value> valueNodes = AidUtil
25                  .getValueNodes(appliedAid, Actor.ATTACKER);
26        for (int i = 0; i < valueNodes.size(); i++) {
27          EcoreUtil.delete(valueNodes.get(i));
28        }
29      }
30
31      // Ignore the other actor's perspective: Symmetric game
32      double[][] expectedUtilites =
33            SmileHelper.computeExpectedUtilities(
34              appliedAid, decision.getId(), Actor.ATTACKER);
35      int[] maxIndex = new int[expectedUtilites[0].length];
36      for (int i = 0; i < expectedUtilites.length; i++) {
37        for (int j = 0; j < expectedUtilites[i].length; j++) {
38          if (expectedUtilites[i][j] >
39                    expectedUtilites[maxIndex[j]][j]) {
40            maxIndex[j] = i;
41          }
42        }
43      }
44
45      // Store the optimal decision in the strategy profile
46      AidDecisionRule decisionRule =
47            new AidDecisionRule(decision);
48      double[][] choiceValues =
49            new double[expectedUtilites.length]
50                      [expectedUtilites[0].length];
51      for (int i = 0; i < maxIndex.length; i++) {
52        choiceValues[maxIndex[i]][i] = 1;
53      }
54      decisionRule.setChoiceValues(choiceValues);
55      strategyProfile.updateDecisionRule(decisionRule);
56    }
57  }
```

Quellcode 6.1: Algorithmus zur Berechnung eines Nash-Gleichgewichtes auf einem azyklischen Relevanz-Graphen

Die Berechnung des Erwartungswertes für den aktuellen Spieler erfolgt schließlich in Zeile 33 mit Hilfe der SMILE Bibliothek. Die anschließenden `for`-Schleifen dienen der Bestimmung des maximalen Erwartungswertes für jeden möglichen Zustand. Sind die Indices der bestmöglichen Entscheidungen hinterlegt, kann in Zeile 46 eine neue Entscheidungsregel für die aktuelle Entscheidung erstellt werden, welche gemäß der höchsten Erwartungswerte initialisiert wird. Zuletzt wird die optimale Entscheidung

dem Strategieprofil in Zeile 55 hinzugefügt beziehungsweise aktualisiert. Dieses Vorgehen wird für jeden Entscheidungsknoten durchgeführt. Bei Terminierung erfüllt das Strategieprofil die Eigenschaft eines Nash-Gleichgewichtes.

6.2.7 Architektur

Das entwickelte Analysewerkzeug baut auf einem *Graphiti*-Editor auf, welcher im Rahmen des *Seconomics*-Projektes entwickelt wird. Der Editor sowie das zugehörige Entitätsmodell werden im `eu.seconomics.tool` Paket zur Verfügung gestellt. Für eine bessere Wartbarkeit und Übersicht wurde eine komponentenorientierte Architektur gewählt. Die Analysekomponente befindet sich im gleichnamigen Paket `eu.seconomics.analyzer`, welche für die Berechnung des Erwartungswertes beziehungsweise der besten Entscheidung, die Bibliothek *SMILE* verwendet (siehe Abschnitt 6.1.4).

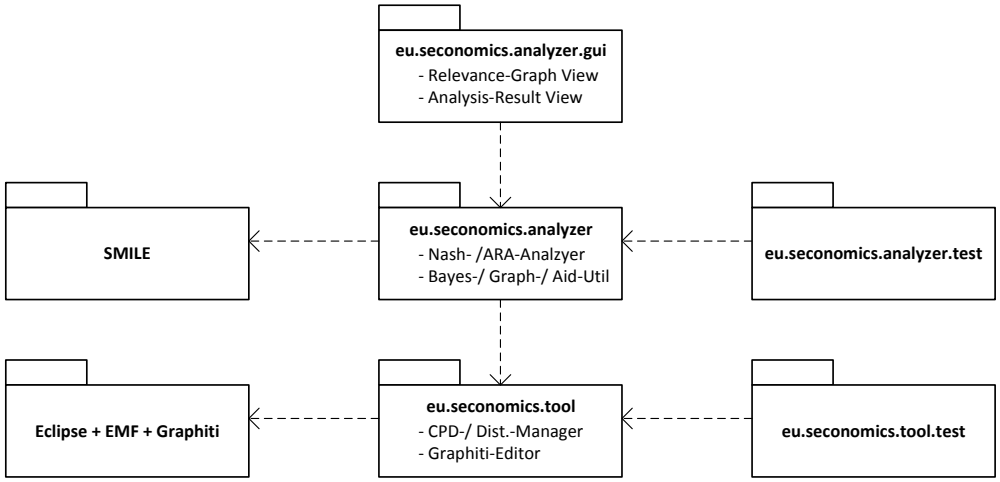


Abbildung 6.9: Komponentenübersicht

Der `eu.seconomics.analyzer` stellt neben den vorgestellten Algorithmen zur Berechnung eines *Nash-Gleichgewichtes* oder einer optimalen Verteidigungsstrategie, viele grundlegende Funktionen zur Verfügung. Ein Beispiel hierfür bildet die Berechnung des *Relevanz-Graphen* oder der *starken Zusammenhangskomponenten*, auf die spätere Ansätze leicht zurückgreifen können. Um die wesentlichen Werkzeuge innerhalb Eclipse und dem Graphiti-Editor anbieten zu können, wurde eine Benutzeroberfläche entwickelt, welche im `eu.seconomics.analyzer.gui` Paket zu finden ist. Mit dieser können die beiden Hauptanalysen innerhalb des Graphiti-Editor gestartet werden. Das Testen der Arbeitsweise ist schon während der Implementierung ein wichtiger Bestandteil (vergleiche Abschnitt 6.3). Die Tests werden ebenfalls als Plug-In Projekte entwickelt, um eine *feingranulare* Komponententrennung zu gewährleisten und sind jeweils in den `eu.seconmomics.analyzer.test` und `eu.seconmomics.tool.test` zu finden.

6.3 Qualitätssicherung

Die *Qualitätssicherung* eines Programmes ist ein wichtiger Aspekt um die korrekte Arbeitsweise und zukünftige Weiterentwicklung zu gewährleisten. Daher wurden schon während der Implementierung verschiedene Techniken eingesetzt, um hohen Anforderungen gerecht zu werden. Die Vorgehensweise und die verwendeten Werkzeuge werden jeweils in den nachfolgenden Abschnitten vorgestellt.

6.3.1 Dokumentation

Die Dokumentation des Programmes ist ein wichtiger und meist ungeliebter Aspekt der Entwicklung, welcher durch letzteres oft vernachlässigt wird. Daher wurde bereits während der Implementierung eine Dokumentation des Quelltextes gepflegt. Dies ist besonders ratsam, wenn ein Programm im Team entwickelt oder später als Bibliothek zur Verfügung gestellt wird. Also viel Zeit mit dem Lesen des Quellcodes oder der möglichen Anbindung verbracht wird. Im entworfenen Quelltext sind alle Schnittstellen, Klassen, Methoden, Aufzählungen, sowie Variablen vollständig dokumentiert. Damit möglichst schnell eine gute Übersicht über die Arbeitsweise der Implementierung gewonnen werden kann. Um stets eine aktuelle Dokumentation des Quelltextes zu erhalten, wurde die Kommentierung innerhalb des Programmcodes vorgenommen. Dies hat gegenüber der externen Dokumentation den Vorteil, dass mögliche Quellcodeänderungen leichter und schneller auch in der Dokumentation vorgenommen werden können. Ein Nachteil bietet diese Prozedur nicht, da mittels *JavaDoc* auch eine externe Dokumentation generiert werden kann.

6.3.2 CheckStyle

Ein einheitlicher *Programmierstil* (*Code Convention*) erhöht die Qualität der Software hinsichtlich der Verständlichkeit und Wartbarkeit. Dafür werden einzelne Regeln, zum Beispiel über den *Einrückungsstil* oder der *Namenskonvention*, festgelegt. Die Zusammenfassung aller definierten Regeln beschreibt den Programmierstil. Um einen durchgängigen Stil der Implementierung zu garantieren wird das Werkzeug *Checkstyle* [Che] verwendet. Dieses bietet eine automatisierte Lösung an, um den Quelltext auf die Einhaltung eines festgelegten Programmierstils zu prüfen. Dabei bringt das Werkzeug als Eclipse Plug-In standardmäßig zwei Stile mit. Die *Sun Coding Convention* und eine davon leicht modifizierte Eclipse Version. Während der Implementierung wurde auf das Eclipse-Derivat zurückgegriffen.

6.3.3 JUnit

Die Funktionalität und Korrektheit des entwickelten Programmes wurde mit dem Werkzeug *JUnit* [JUn] getestet. Dieses wird dazu verwendet um die einzelnen Bestandteile (*Units*) eines Programmes, wie Klassen oder Methoden einer Funktionsprüfung zu unterziehen. Wie auch bei der Dokumentation wurden bereits während der Implementierung Testszenarien entwickelt um möglichst früh Fehlerquellen zu identifizieren und zu beseitigen. Bei einem Testdurchlauf gibt es im Wesentlichen zwei Testresultate: Erfolg und Fehlschlag, wobei ein Fehlschlag noch weiter in die

verschiedenen Ursachen unterteilt werden kann. Die Konzeption und Durchführung von Testszenarien ist eines der wichtigsten Qualitätssicherungen im Hinblick auf die korrekte Arbeitsweise. Einen vollständigen Nachweis, ob das entwickelte Programm fehlerfrei ist, kann der Softwaretest allerdings nicht zwangsläufig liefern, da nur einzelne Szenarien abgedeckt werden. Dennoch kann durch die Wahl von guten Teststrategien eine große Testabdeckung erreicht werden. Im Rahmen der Qualitätssicherung wird jede öffentliche, nicht triviale Methode (*getter* und *setter*) getestet.

6.3.4 FindBugs

Das Eclipse Plug-In *FindBugs* [Fin] sucht Fehlermuster mittels statischer Analyse im Java-Bytecode. Der Zugang zum Quelltext wird daher nicht benötigt. Mit diesem Werkzeug ist es somit einfach und schnell möglich, häufige Fehlerquellen zu identifizieren. So wurden beispielsweise Variablen ausfindig gemacht, welche möglicherweise nicht zwangsläufig initialisiert werden. Ein weiteres gefundenes Fehlermuster bezieht sich auf das Überschreiben der *equals* Methode ohne die *hashCode* Methode ebenfalls anzupassen und vice versa. Die Falscherkennungsrate liegt laut Dokumentation unter 50%.

6.3.5 CodePro

Googles *CodePro AnalytiX* [Goo] ist ein weiteres Werkzeug zur Qualitätssicherung. Im Gegensatz zu den vorangegangenen Programmen bündelt CodePro viele Funktionen der zuvor vorgestellten Werkzeuge. So kann beispielsweise die Einhaltung eines Programmierstils überprüft, wie auch statische Analysen auf dem Quelltext durchgeführt werden. Ein integrierter JUnit Editor sorgt für eine gute Übersicht der Java Testszenarien. Die besondere Stärke von CodePro liegt neben der bereits erwähnten Bündelung der Funktionen in einer guten Berichterstattung. So ist es mit CodePro möglich, die entworfenen Testszenarien, welche mit JUnit (siehe Abschnitt 6.3.3) überprüft werden, einer Güteprüfung zu unterziehen. Hierbei wird ermittelt, welche Zeilen des Quelltextes, bei einem definierten Testszenario, durchlaufen werden. Dies ist insbesondere bei komplizierten `If`-Abfragen von Vorteil, um jeden möglichen Pfad innerhalb des Programmcodes abzudecken. Wie viele Zeilen des Quelltextes durch die Testfälle abgedeckt werden, gibt der Wert der *Testübderdeckung* wieder. Dieser liegt bei dem entworfenen Werkzeug bei über 80%. Eine Verbesserung des Quelltextes wird ebenfalls durch das Aufspüren und Vermeiden von nicht aufrufbaren Programmcode herbeigeführt. *Toter Code* kann beispielsweise durch nicht erfüllbare Bedingen entstehen, welche auf einen Fehler der Abfrage hindeuten. In dem entwickelten Programmcode sind nur öffentliche Methoden als toter Code ermittelt worden, welche derzeit nicht verwendet werden. Diese bilden allerdings eine alternative Anbindung und stehen auch weiterhin zur Verfügung. Das Finden von ähnlichem Programmcode kann ebenfalls zur Qualitätssteigerung beitragen. Redundanter Quelltext deutet auf eine gleiche Funktionsweise hin, welcher beispielsweise in eine eigene Methode ausgelagert werden kann. Dies hat den Vorteil das Änderungen beziehungsweise eventuelle Fehlerkorrekturen auch global zur Verfügung stehen.

7 Evaluation

Dieses Kapitel widmet sich der Evaluation des entwickelten Konzeptes und des Werkzeugs. Dabei wird in Abschnitt 7.1 zunächst auf den entwickelten Ansatz eingegangen. Der darauf folgende Abschnitt 7.2 betrachtet das entwickelte Werkzeug. In diesem werden im speziellen die Architektur, die Analyseprozedur und die Modellierung betrachtet. Abschließend wird ein Leistungstest vorgestellt.

7.1 Evaluation des Ansatzes

Der entwickelte Ansatz beruht auf dem Relevanz-Graphen beziehungsweise der strategischen Abhängigkeit zwischen Entscheidungen. Mit Hilfen dieser wird die Reihenfolge der Optimierung ermittelt. Der azyklische Fall beruht somit auf der bereits bekannten *Backtracking*-Methode für Entscheidungsdiagramme. Wohingegen der zyklische Fall des Relevanz-Graphen bisher nur partiell für Komponenten der Größe zwei gelöst werden kann, bei der eine Entscheidung des Angreifers und eine Entscheidung des Verteidigers in diesem Knoten vorhanden sind. Das Lösen des simultanen Entscheidungsprozesses geschieht mittels eines aufgestellten ARA-Templates. Der Relevanz-Graph wird in diesem Fall zur Erkennung dieses Musters verwendet. An dieser Stelle können weitere Muster eingepflegt werden, um in Zukunft weitere Fälle abzudecken.
Die in dieser Arbeit vorgestellte Lösungsstrategie benötigt in jedem Fall eine feste Menge an Wahrscheinlichkeitstabellen, mit denen das Verhalten der jeweiligen Spieler ermittelt wird. Es kann aber durchaus vorkommen, dass die benötigten Wahrscheinlichkeiten nicht vorhanden sind, sodass der entwickelte Ansatz nicht verwendet werden kann. In diesem Fall müssten Vorverarbeitungsschritte durchgeführt werden, damit die benötigte Eingabemenge abgeleitet werden kann. Dieser Schritt der Vorverarbeitung ist stark abhängig von den vorliegenden Daten und wird im Rahmen dieser Ausarbeitung nicht abgedeckt. Ein weiterer wichtiger Punkt liegt in der Korrektheit der erstellten Modelle. Da das ermittelte Analyseergebnis von den eingegeben Wahrscheinlichkeiten abhängt, ist die korrekte Erhebung dieser Daten zwangsläufig der wichtigste Schritt. Neben der Korrektheit der Daten ist auch die korrekte Modellierung der Instanz von hoher Wichtigkeit (vergleiche Abschnitt 7.2.3), da falsch eingegebene Wahrscheinlichkeiten zu einer Änderung der Entscheidungsfolge führen können. Dieses Verhalten konnte gerade bei großen Wahrscheinlichkeitstabellen nachvollzogen werden, bei der die Übersichtlichkeit schnell verloren gehen kann.

7.2 Evaluation des Werkzeugs

Die Evaluation des Werkzeugs wird in unterschiedlichen Bereichen durchgeführt. Zunächst wird auf die Architektur in Abschnitt 7.2.1 eingegangen. Der eigentliche Analyseprozess wird in Abschnitt 7.2.2 betrachtet. Ein wesentlicher Teil der Analyse fließt in das Konzipieren der Modelle ein, dies wird in Abschnitt 7.2.3 behandelt. Ein Leistungstest des Werkzeugs, der zur Einschätzung der Anwendbarkeit verwendet werden kann, wird in Abschnitt 7.2.4 vorgestellt.

7.2.1 Architektur

Die Architektur des entwickelten Programmes wird in Abschnitt 6.2.7 erläutert. Hierbei wurde eine komponentenorientierte Entwicklung gewählt, welche eine Trennung von Analysekomponenten und Benutzerschnittstellen vorsieht. Dieses Vorgehen sorgt für eine Trennung der Logik und der Anzeigekomponente, welches zu einer besseren Wartbarkeit und einer geringeren Fehleranfälligkeit führt. Einen weiteren großen Vorteil dieses Programmierparadigmas bildet die leichte Wiederverwendbarkeit von einzelnen Komponenten. Da zu Beginn der Masterarbeit die Entwicklung der Hauptkomponente bereits begonnen hatte, stand der Architekturentwurf des Kernprojektes bereits fest. In dem entwickelten `eu.seconomics.tool` wird der grafische Editor, das Entitätsmodell und bereits diverse Analysefunktionen angeboten. Eine klassische Trennung zwischen Entitäten, Logiklassen und Darstellungskomponenten erfolgt nur auf Paketebene und ist an einigen Stellen nicht sauber realisiert. Da jedoch auch im Rahmen dieser Masterarbeit viel grundlegende Funktionalität entwickelt wurde, bietet es sich an diese in die Hauptkomponente zu integrieren. Bei diesem Schritt können auch weitere Änderungen in die Architektur einfließen um auch zukünftig eine effiziente Weiterentwicklung des Werkzeugs zu gewährleisten.

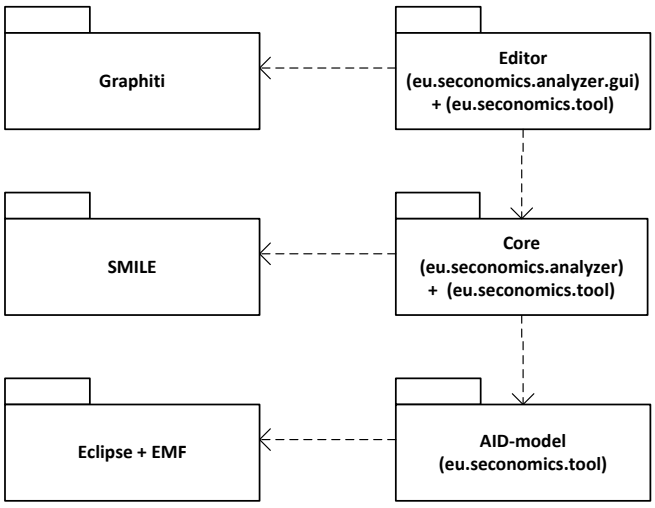


Abbildung 7.1: Vorschlag einer neuen Architektur

Abbildung 7.1 zeigt einen Vorschlag für eine überarbeitete Architektur. Bei dieser wird das Entitätsmodell vollständig ausgelagert. Die `Core`-Komponente beinhaltet die Funktionalität der entwickelten Analysekomponente und ist mit den weiteren Logikklassen des `eu.seconomics.tools` verschmolzen. Die gebündelte Funktionsvielfalt kann so den grafischen Benutzeroberflächen (*Graphical User Interfaces GUIs*) zur Verfügung gestellt werden. Diese sind in eine separate GUI-Komponente ausgelagert worden, welche unter anderem den AID-Editor realisiert. Der neue Architekturentwurf ermöglicht eine gezielte Trennung der Funktionen. Ferner werden mit diesem Ansatz auftretende Probleme aus dem Abschnitt 7.2.3 gelöst.

7.2.2 Analyse

Das Ermitteln der optimalen Entscheidung erfolgt derzeit über die Auswertung der möglichen Erwartungswerte. Die Prozedur erfolgt mit Hilfe der Bibliothek SMILE (vergleiche Abschnitt 6.1.4). Im Laufe der Implementierung wurde jedoch auch ein anderes Verfahren von Shachter and Peot entdeckt [SP92], welches sich besonders durch eine schnelle Entscheidungsauswertung auszeichnet. Der vorgestellte Algorithmus ist auch in SMILE implementiert und könnte alternativ genutzt werden, um die Laufzeit zu verbessern.

7.2.3 Modellierung

Die Modellierung nimmt einen wesentlichen Teil der Arbeit in Anspruch und daher ist es erforderlich diesen Aspekt gezielt zu betrachteten. Der Graph der AID-Instanz wird mit dem bereits vorgestellten Graphiti-Editor modelliert. Für die Interaktion mit Wahrscheinlichkeitstabellen wird auf die in dieser Masterarbeit entwickelten Benutzeroberfläche zurückgegriffen (siehe Abschnitt 6.2.2). Dabei werden Fehler in der Wahrscheinlichkeitstabelle hervorgehoben, sodass diese leicht gefunden werden können. Eine zusätzliche Ziehungsfunktion für die Wahrscheinlichkeitsverteilungen ermöglicht das Überprüfen der korrekten Modellierung. Die Distributionen selbst werden mit dem Distributions-Manager eingepflegt (siehe Abschnitt 6.2.3). Je nach Spielart variieren die verfügbaren Wahrscheinlichkeitstabellen. So ist beispielsweise bei einem symmetrischen Spiel die Angabe von Wahrscheinlichkeiten innerhalb eines Entscheidungsknotens nicht möglich, wohingegen im asymmetrischen Fall seitens des Verteidigers, Wahrscheinlichkeitsverteilungen angegeben werden können. Da die vorgestellte Analysetechnik auf den Relevanz-Graph beruht und diese sich nach den Eigenschaften des Graphen richtet, werden unterschiedliche Wahrscheinlichkeitstabellen verwendet. So benötigt beispielsweise die asymmetrische, zyklische Analyse, die Angabe der jeweiligen Wertigkeiten des Verteidigers aus der Sicht des Angreifers. Dies gilt für die Value Knoten, welche durch die zyklischen Komponenten erreichbar sind. Es kann also vorkommen, dass nicht alle angegebenen Wahrscheinlichkeitstabellen verwendet werden. Dies kann zur Unsicherheit während der Modellierungsphase führen und sollte vermieden werden. Die Nutzung von Wahrscheinlichkeitstabellen kann mit dem Relevanz-Graphen und der zugehörigen Benutzeroberfläche ermittelt werden (siehe Abschnitt 6.2.4). Eine automatisierte Lösung ist hier von Vorteil. Allerdings lässt die derzeitige Architektur es nicht zu, die Struktur

des Relevanz-Graphen in die Benutzeroberfläche des graphischen Editors einfließen zu lassen (vergleiche Abschnitt 6.2.7). Ein neuer Architetkurentwurf könnte diesem Problem entgegenwirken (vergleiche Abschnitt 7.2.1).

7.2.4 Leistungstest

Im Rahmen der Evaluation wurde ebenfalls ein Leistungstest durchgeführt. Dies war von Interesse, da das entwickelte Werkzeug später im Seconomics Projekt Anwendung finden wird. Die in der Arbeit vorgestellten Modelle sind lediglich kleine Ausschnitte aus den entwickelten Modellen, sodass die Komplexität je nach Modellgröße stark zunimmt. Die Laufzeit soll mit einem Beispielmodell untersucht werden. Hierfür wird auf das Verteidigungsangriffsmodell aus Abbildung 7.2 zurückgegriffen.

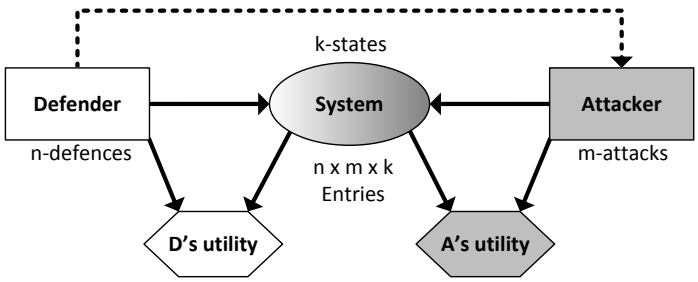


Abbildung 7.2: Evaluationsmodell

Der entwickelte Leistungstest sieht zunächst jeweils zwei Entscheidungen in den Verteidiger- und Angreiferknoten vor. Das System enthält ebenfalls zwei Zustände. Die Wahrscheinlichkeitstabellen werden mit randomisiertem Dirichlet-Distributionen gefüllt. Die Wertigkeiten jeweils mit randomisierten Triangularverteilungen. Ist diese Vorarbeit abgeschlossen, wird die eigentliche Analyse gestartet und die Zeit gemessen. Hierfür wird die Graphstruktur ausgewertet, der zugehörige Relevanz- beziehungsweise Komponentengraph erstellt, um die Optimierungsreihenfolge zu bestimmen und im Anschluss das Verhalten des Angreifers simuliert. Daraufhin kann die optimale Verteidigungsstrategie berechnet werden. Dies bildet den Abschluss der Analyse und die Zeit wird gestoppt. Um die Komplexität schrittweise anzuheben, wird bei jedem weiteren Analyseaufruf eine Entscheidung im Verteidiger- und Angreiferknoten hinzugefügt. Der Systemknoten enthält so in der i-ten Analyse $2 \cdot (i+2)^2$ Zustände. Die folgende Abbildung 7.3 bildet die Auswertung des Leistungstests. Die Y-Achse zeigt die Anzahl der Zustände in dem Knoten System, sowie die gemessene Zeit in Sekunden. Die X-Achse spiegelt die Anzahl der Zustände in einem Entscheidungsknoten wieder. Diese ist wie zuvor erwähnt bei beiden Knoten stets gleich.

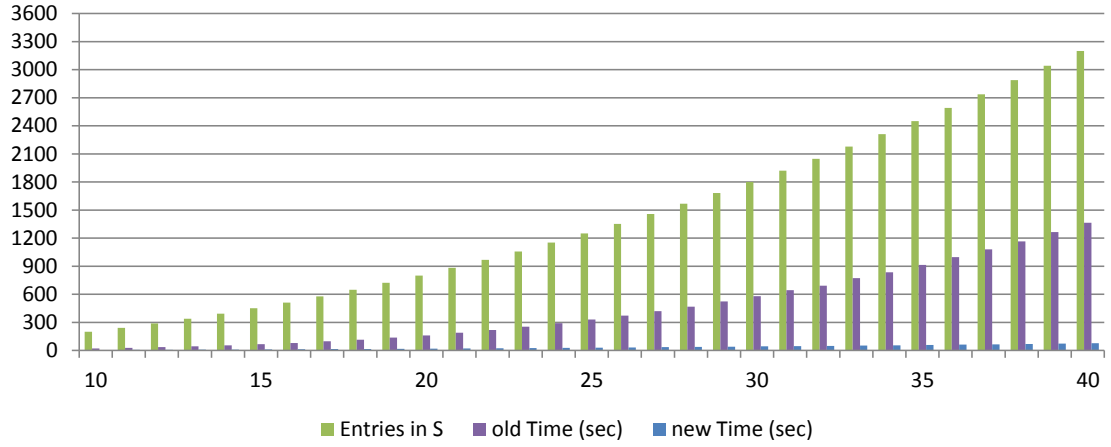


Abbildung 7.3: Zeitmessung verschiedener Analysegrößen

Als die erste Messreihe abgeschlossen wurde, lag die Analysedauer für 30 Aktionen pro Spieler bei circa zehn Minuten. Das heißt, die Tabellengröße von dem Knoten System belief sich auf 1800 Elemente. Es war jedoch möglich die Laufzeit mittels *Caching* um ein vielfaches zu senken. Das Problem wurde durch eine ineffiziente Implementierung des `CPD-Managers`, welcher für jede Wahrscheinlichkeitstabellenanfrage die benötigte Tabelle aus dem Entitätsmodell neu geladen hat, verursacht. Jeder Aufruf war mit einem zeitaufwendigen *Parsen* der einzelnen Elemente verbunden. Da ein wesentlicher Teil der Analyse die Approximation des Angreifers bildet, wurden dementsprechend viele Tabellenaufrufe ausgeführt, was jedoch unnötig war, da sich die eigentlichen Wahrscheinlichkeitstabellen während der Analyse nicht mehr ändert. Mit dieser Methodik konnte die Laufzeit bei 30 Aktionen pro Spieler um 85%, auf circa 43 Sekunden, gesenkt werden.

8 Zusammenfassung und Ausblick

Dieses Kapitel bildet den Abschluss der Masterarbeit und dient der kompakten Zusammenfassung der erreichten Ziele. Anschließend erfolgt ein kurzer Ausblick, der Gedanken zu möglichen Weiterentwicklungen vorstellt.

8.1 Zusammenfassung

In der vorliegenden Arbeit wird eine Berechnungsvorschrift für die Risikoanalyse gemäß des ARA-Konzeptes entwickelt. Hierfür werden zunächst die Grundlagen der Bayes'schen Netze vermittelt, welche eine Repräsentation für den Zusammenhang von Zufallsvariablen und derer bedingten Wahrscheinlichkeiten ermöglichen. Aufbauend für die Modellierung der Risikoanalyse-Instanzen werden die Mutlti-Agent Influence Diagrams vorgestellt, welche auf den Influence Diagrams basieren. Diese führen Entscheidungs- und Wertigkeitsknoten innerhalb der Bayes'schen-Netze ein. In der Regel gilt es die Wertigkeiten mit Hilfe von getroffenen Entscheidungen zu maximieren. Die MAIDs erweitern diesen Ansatz um eine Mehrspielerlösung. Weitergehend konnte gezeigt werden, dass die MAIDs sich für den Einsatz von Risikoanalysen eigenen. Das vorgestellte ARA-Konzept sieht hierbei intelligente Gegenspieler vor. Im Rahmen der Spieltheorie gibt es bereits verschiedene Ansätze um ein Nash-Gleichgewicht zu berechnen, es wurden verschiedene Ansätze vorgestellt und ein Ansatz von Daphne Koller und Brian Milch ausgewählt, welcher auf strategische Relevanz zwischen den Entscheidungen beruht. Mit dieser Eigenschaft ist es möglich einen Relevanz-Graphen zu erzeugen, welcher mittels topologischer Sortierung die Reihenfolge für die Optimierung liefert. Im zyklischen Fall wird der Relevanz-Graph mittels starker Zusammenhangskomponenten zu einem Komponentengraph zusammengefasst, sodass erneut eine topologische Sortierung gebildet werden kann. Die einzelnen Komponenten werden mit Hilfe bekannter Ansätze aus der Spieltheorie gelöst. Da die ermittelten Komponenten während der Analyse sukzessiv kleiner werden, kann die Effizienz, im Vergleich zu bisherigen Ansätzen, verbessert werden. Dieses Vorgehen kann auch auf die ARA-Analyse übertragen werden. Dafür werden für beide Spieler eigene Wahrscheinlichkeitstabellen modelliert. Dem Verteidiger stehen konkrete Werte zur Verfügung, wohingegen das Wissen des Angreifers mit Distributionen beschrieben wird. Der Ansatz beschreibt zwei Phasen. In der ersten Phase wird das Verhalten des Angreifers approximiert. Dies geschieht mit Hilfe von Wahrscheinlichkeitsdistributionen, welche das Wissen des Angreifers beschreiben. In der zweiten Phase wird das berechnete Angreiferverhalten verwendet, um die Verteidigungsstrategie zu berechnen. Der entwickelte Algorithmus basiert ebenfalls auf der Optimierungsreihenfolge, welche durch den Relevanz-Graph bestimmt wird.

Im azyklischen Fall des Relevanz-Graphen wird die Reihenfolge durch die topologische Sortierung bestimmt. Falls der Relevanz-Graph jedoch zyklische Strukturen aufweist, wird auf ein bestehendes ARA-Template zur Auflösung der Struktur zurückgegriffen, welches eine simultane Angreifer-Verteidiger Situation beschreibt. Der entwickelte Ansatz wird durch verschiedene Fallstudien nachvollzogen und auch in einem prototypischen Werkzeug realisiert. Hierzu wird eine Eclipse basierende Architektur vorgestellt, welche auf einem bestehenden Graphiti-Editor für AID-Instanzen aufbaut. Mit Hilfe dieses Editors und der entwickelten Analysekomponente ist es möglich, die Ergebnisse dieser Arbeit nachzuvollziehen. Das Werkzeug bietet neben dem implementierten Algorithmus auch die Darstellung des Relevanz- und des Komponentengraphen an, sodass vorhandene Modelle effizient nach Abhängigkeiten untersucht werden können.

8.2 Ausblick

Der im Rahmen dieser Masterarbeit entwickelte Ansatz ist in der Lage azyklische AID-Instanzen zu lösen. Bei zyklischen Strukturen wird ein vorgestelltes ARA-Template verwendet, welches zwei simultane Aktionen der jeweiligen Spieler behandelt. Folglich sind nicht alle möglichen zyklischen Strukturen abgedeckt. Ein weitergehendes Ziel bildet somit ein Lösungsansatz für allgemeine zyklische AID-Instanzen. Bei dem entwickelten Algorithmus werden derzeit nur zwei Spieler betrachtet. Eine Ausweitung des Konzeptes, mit den benötigten Wahrscheinlichkeitsverteilungen, kann zu einer breiteren Einsatzfähigkeit beitragen. Ein weiteres Problem bildet die Erhebung der Wahrscheinlichkeitsverteilungen, diese müssen nicht unbedingt direkt aus den Datensätzen hervorgehen. Ein alternative Eingabemöglichkeit, welche vor der Analyse die benötigten Verteilungen berechnet, würde eine leichtere Anbindung ermöglichen. Als Eingabeform werden derzeit nur Wahrscheinlichkeitstabellen akzeptiert. Bei größeren Modellen ist es teils sehr umständlich und fehleranfällig diese direkt anzugeben. Eine funktionale Eingabe könnte das Problem lösen. Dabei werden die Wertigkeiten und Wahrscheinlichkeiten mit Hilfe einer Funktion dargestellt, welche die nötigen Werte zu Analysebeginn zur Verfügung stellt. Diese Art der Darstellung liefert insbesondere bei großen Knoten eine schnellere Einsicht. Eine weitere Möglichkeit die Interaktion zu verbessern, bietet die kontinuierliche Eingabe von Zuständen oder Entscheidungen, welche bei der Analyse diskretisiert werden. Auch dieser Schritt ermöglicht eine kompaktere Darstellung der Knoteneigenschaften. Die Ausgabe der Analyseergebnisse erfolgt derzeit in Textform. Die grafische Aufbereitung der Ergebnisse und die Darstellung dieser im Editor selbst, führen an dieser Stelle zu einer besseren Integration. Die Darstellung der getroffenen Entscheidungen erfolgt in der Textausgabe als Matrix. Auch hier geht bei großen Knoten die Übersicht schnell verloren, sodass eine Zuordnung der einzelnen Entscheidungsprozesses schwierig ist. Eine separate graphische Darstellung, zum Beispiel als Baumdiagramm, kann zu einer wesentlich besseren Darstellungsform führen. Zuletzt ist noch die Integration in die vorhandene Kernkomponente zu erwähnen, um viele entwickelte Funktionen, wie beispielsweise dem Relevanz-Graph, dem Editor zur Verfügung zu stellen. Hierzu kann der vorgestellte Architekturentwurf im Evaluierungskapitel herangezogen werden.

A Anhang

A.1 Inhalt der CD

Dieser Masterarbeit liegt eine CD bei, die folgende Dateien enthält:

- Diese Ausarbeitung im PDF-Format
- Ein Eclipse-Workspace, welches die vorgestellten Beispielmodelle enthält
- Zwei lauffähige Eclipse-Umgebungen (x86 und x64), getestet unter Windows 7
- Quellcode des entwickelten Werkzeugs

Abkürzungsverzeichnis

AID ..... Adversarial Influence Diragam
API ..... Application Programming Interface
ARA ..... Adversarial Risk Analysis

BN ..... Bayesian Network

CPD ..... Conditional Probability Distribution

DAG ..... Directed Acyclic Graph
dll ..... Dynamic Link Library

EF ..... Extensiv-form Game
EMF ..... Eclipse Modeling Framework

GEF ..... Graphical Editing Framework
GeNIe ..... Graphical Network Interface
GT ..... GameTree
GUI ..... Graphical User Interface

ID ..... Influence Diagram
IDE ..... Integrated Development Environment
ISST ..... Fraunhofer-Institut für Software- und Systemtechnik

JVM ..... Java Virtual Machine

MAID ..... Multi Agent Influence Diragam
MDD ..... Model driven development

NF ..... Normal-form Game

OMG ..... Object Management Group
OSGi ..... OSGi Alliance

RV ..... Random Variable

SCC ..... Strongly connected component
SMILE ..... Structural Modeling, Inference, and Learning Engine

UI ..... User Interface
UML ..... Unified Modeling Language

XMI ..... XML Metadata Interchange
XML ..... Extensible Markup Language
XSD ..... XML Schema

Symbolverzeichnis

(σ_n, σ_{-n})	Strategieprofil für welches Agent n nach Strategie σ_n handelt und alle anderen Agenten nach σ_{-n}
δ	Entscheidungsregel
δ^*	Entscheidungsregel, welche für ein gegebenes Strategieprofil σ optimal ist
$\mathcal{D}$	Menge aller Decision-Variablen
$\mathcal{D}_n$	Menge aller Entscheidungsknoten eines Agenten n
$\mathcal{M}$	MAID
$\mathcal{R}_\mathcal{M}$	Relevanz-Graph von MAID $\mathcal{M}$
$\mathcal{S}$	Menge der starken Zusammenhangskomponenten
$\mathcal{U}$	Menge aller Werte-Variablen
$\mathcal{U}_n$	Menge aller Werteknoten eines Agenten n
$\mathcal{X}$	Menge aller Chance-Variablen
σ	Strategieprofil
σ_n	Strategie für Agent n
σ_{-n}	Strategieprofil σ ohne Agent n
D_n^i	Entscheidungsknoten D mit Index i gehörig zum Agent n
$dom(X)$	Domain eines Knotens X
$E[U_a(\sigma)]$	Erwartungswert eines Agenten a für das Strategieprofil σ
N	Menge aller Agenten
$Pa(X)$	Elternknoten des Knotens X
$Pr(.)$	Wahrscheinlichkeitsfunktion
$U(\mathbf{pa})$	Utility für konkrete Instanziierung $\mathbf{pa}$
U_n^i	Utility U mit Index i gehörig zum Agent n
$\mathbf{pa}$	Konkrete Instanziierung

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Algorithmenverzeichnis

Literatur

[BCA09] Vicki M. Bier, Louis A. (Tony) Cox und M. Naceur Azaiez. *Game Theoretic Risk Analysis of Security Threats*. Hrsg. von Vicki M. Bier und M. Naceur Azaiez. Bd. 128. International Series in Operations Research & Management Science. Springer, 2009, S. 1–13.

[BSK06] Ben Blum, Christian R. Shelton und Daphne Koller. „A Continuation Method for Nash Equilibria in Structured Games“. In: *Journal of Artificial Intelligence Resarch* 25 (2006), S. 457–502.

[Cas+13] S. Castellvi u. a. *D3.2 – Urban public transport requirements first version*. Techn. Ber. Atos, UNITN, URJC, UNIABDN, TMB, SNOK, ISAS CR., 2013. URL: `http://seconomicsproject.eu/sites/default/files/content-files/deliverables/D3_2_Urban_Public_Transport_Requirements_First_Version_v3.2_0.pdf`.

[Che] Checkstyle development team. *Checkstyle*. URL: `http://checkstyle.sourceforge.net/`.

[Col+13] R. Coles u. a. *D2.3 – National Grid Requirements*. Techn. Ber. NGRID, UNIABDN, UNITN, Fraunhofer, 2013. URL: `http://seconomicsproject.eu/sites/default/files/content-files/deliverables/D2.3_National_Grid_Requirements_-_Final_Version_0.pdf`.

[Com08] National Research Council Committee on Methodological Improvements to the Department of Homeland Security's Biological Agent Risk Analysis. *Department of Homeland Security Bioterrorism Risk Assessment: A Call for Change*. The National Academies Press, 2008. ISBN: 9780309120289. URL: `http://www.nap.edu/openbook.php?record_id=12206`.

[Dav09] Mercelo Paternostro und Ed Merks Dave Steinberg Frank Budinsky. *EMF Eclipse Modling Framework*. Bd. 2. Addison-Wesley, 2009.

[Dec] Decision Systems Laboratory, University of Pittsburgh. *SMILE (Structural Modeling, Inference, and Learning Engine) and GeNIe (Graphical Network Interface)*. URL: `http://genie.sis.pitt.edu/`.

[Eav72] B.Curtis Eaves. „Homotopies for computation of fixed points“. In: *Mathematical Programming* 3.1 (1972), S. 1–22. ISSN: 0025-5610.

[Eav84] B.Curtis Eaves. *A Course in Triangulations for Solving Equations with Deformations*. Hrsg. von M. Beckmann und W. Krelle. Springer-Verlag, 1984.

[Ecla] Eclipse Foundation. *Eclipse*. URL: `http://www.eclipse.org/org/`.

[Eclb] Eclipse Foundation. *Eclipse Modeling Framework*. URL: `http://www.eclipse.org/modeling/emf/`.

[Eclc] Eclipse Foundation. *Graphiti - a Graphical Tooling Infrastructure*. URL: `http://www.eclipse.org/graphiti/`.

[Fin] FindBugs development team. *FindBugs*. URL: `http://findbugs.sourceforge.net/`.

[FT91] Drew Fudenberg und Jean Tirole. *Game Theory*. MIT Press, 1991.

[Goo] Google. *CodePro Analytix*. URL: `https://developers.google.com/java-dev-tools/codepro/doc/`.

[GP90] Dan Geiger und Judea Pearl. „On the Logic of Causal Models“. In: *Uncertainty in Artificial Intelligence* 4 (1990), 136—147.

[GW03] Srihari Govindan und Robert Wilson. „A global Newton method to compute Nash equilibria“. In: *Journal of Economic Theory* 110.1 (2003), S. 65–86. ISSN: 0022-0531.

[GW04] Srihari Govindan und Robert Wilson. „Computing Nash equilibria by iterated polymatrix approximation“. In: *Journal of Economic Dynamics and Control* 28.7 (2004), S. 1229–1241. ISSN: 0165-1889.

[JJD94] Frank Jensen, Finn V. Jensen und Søren L. Dittmer. „From Influence Diagrams to Junction Trees“. In: *Proceedings of the tenth Conference on Uncertainity an Artificial Intelligence*. Morgan Kaufmann, 1994, S. 367–373.

[JUn] JUnit development team. *JUnit*. URL: `http://junit.org/`.

[KM03] Daphne Koller und Brian Milch. „Multi-Agent Influence Diagrams for Representing and Solving Games“. In: *Games and Economic Behavior* 45.1 (2003). Full version of paper in IJCAI '03, S. 181–221.

[KM86] Elon Kohlberg und Jean-Francois Mertens. „On the Strategic Stability of Equilibria“. In: *Econometrica* 54.5 (Sep. 1986), S. 1003–37.

[KN05] Sven O. Krumke und Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Hrsg. von Ulrich Sandten und Kerstin Hoffmann. 1. Aufl. ISBN 3-519-00526-3. Teubner Verlag, Mai 2005, S. 407.

[LH64] C. E. Lemke und J. T. Howson. „Equilibrium Points of Bimatrix Games“. In: *Journal of the Society for Industrial and Applied Mathematics* Vol. 12, No. 2 (Jun., 1964), S. 413–423.

[MM96] Richard D. McKelvey und Andrew McLennan. „Computation of Equilibria in Finite Games“. In: *Handbook of Computational Economics*. Elsevier, 1996, S. 87–142.

[MWG] Fabio Massacci, Julian Williams und Alicia García. *Seconomoics - Socio-economics meets Security*. URL: `http://seconomicsproject.eu/`.

[Nud+04] Eugene Nudelman u. a. „Run the GAMUT: A Comprehensive Approach To Evaluating Game-Theoretic Algorithms“. In: *In AAMAS-04*. 2004, S. 880–887.

[PKP13] Achim Pollert, Bernd Kirchner und Javier Morato Polzin. *Duden Wirtschaft von A bis Z: Grundlagenwissen für Schule und Studium, Beruf und Alltag*. Hrsg. von Bundeszentrale für politische Bildung. Bibliograph. Instit. Gmbh, 2013.

[Ref09] BMI Referat KM 4. *Nationale Strategie zum Schutz Kritischer Infrastrukturen (KRITIS-Strategie)*. Techn. Ber. Bundesministerium des Innern, 2009.

[Río+13a] D. Ríos u. a. *D5.1 - Basic Models for Security Risk Analysis*. Techn. Ber. URJC, UNITN, Fraunhofer, 2013. URL: `http://seconomicsproject.eu/sites/default/files/content-files/deliverables/D5.1_-_Basic_Models_for_Security_Risk_Analysis.pdf`.

[Río+13b] D. Ríos u. a. *Security Economics: A Multiobjective Adversarial Risk Analysis Approach to Airport Protection*. Techn. Ber. Department of Statistics and Operations Research, Rey Juan Carlos University u. a., 2013.

[RRB09] David R. Insua, Jesus Rios und David Banks. „Adversarial Risk Analysis“. In: *Journal of the American Statistical Association* 104.486 (2009), S. 841–854.

[Sha88] Ross D. Shachter. „Probabilistic Inference and Influence Diagrams“. In: *Oper. Res.* 36.4 (Juli 1988), S. 589–604.

[Sha90] Ross D. Shachter. „An ordered examination of influence diagrams“. In: *Networks* 20.5 (1990), S. 535–563.

[Sha98] Ross D. Shachter. „Bayes-ball: Rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams)“. In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. UAI'98. Madison, Wisconsin: Morgan Kaufmann Publishers Inc., 1998, S. 480–487. ISBN: 1-55860-555-X.

[Sma76] Steve Smale. „A convergent process of price adjustment and global newton methods“. In: *Journal of Mathematical Economics* 3.2 (Juli 1976), S. 107–120.

[SP92] Ross D. Shachter und Mark A. Peot. „Decision Making Using Probabilistic Inference Methods“. In: *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*. UAI '92. 1992, S. 276–283.

[Tar72] Robert Endre Tarjan. „Depth-First Search and Linear Graph Algorithms“. In: *SIAM J. Comput.* 1.2 (1972), S. 146–160.

[Tur13] Theodore L. Turocy. *Gambit: Software Tools for Game Theory*. 2013. URL: `http://www.gambit-project.org`.

Sofern *URLs* angegeben worden sind, wurden diese am 11.12.2013 überprüft.