UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Clemens Renner

Sicherheit in Rechensystemen durch Lösung von Constraintproblemen

Diplomarbeit

27. Juli 2006

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl 6 (Informationssysteme und Sicherheit)
Fachbereich Informatik
Universität Dortmund

Gutachter:
Prof. Dr. Joachim Biskup
Dr. Ulrich Flegel

GERMANY · D-44221 DORTMUND

Danksagung

Keine Diplomarbeit entsteht im luftleeren Raum, auch diese nicht. Deshalb möchte ich an dieser Stelle all jenen Menschen danken, die mich in dieser letzten Etappe des Studiums unterstützt haben.

Ganz besonderer Dank gebührt denen, die die Rohfassungen dieser Arbeit gegengelesen haben und

- meinen Eltern für die fortwährende Unterstützung in nahezu allen Belangen während meines gesamten Studiums,
- Melanie, weil mit ihr alles leichter geht,
- Dominique Marc Burgard und Matthias Niggemeier für den regen Austausch im Themenkomplex,
- Joachim Biskup und Ulrich Flegel für die ausgesprochen umfassende und zeitnahe Betreuung,
- Marco Bakera für viele fruchtbare Diskussionen,
- den Autoren der zitierten Arbeiten, die gerne auf meine Fragen geantwortet haben.

Dortmund, im Juli 2006

Zusammenfassung

In Informations- bzw. Rechensystemen wird Sicherheit häufig als die Wahrung von Teilnehmerinteressen wie Verfügbarkeit, Vertraulichkeit, Integrität, Authentizität und zunehmend auch Anonymität beschrieben. Diese grundlegenden Anforderungen stehen jedoch häufig miteinander im Widerstreit und eine zufriedenstellende Lösung läuft in den meisten Fällen auf eine Kompromissfindung zwischen den erwähnten Interessen hinaus. Seit Beginn der 2000er Jahre haben einige Autoren vorgeschlagen, die Behandlung von sicherheitsrelevanten Fragestellungen als Constraint-Satisfaction-Probleme (CSPs) zu formulieren, einem ursprünglich für geometrische Probleme entwickelten Formalismus, der mittlerweile eine wichtige Rolle im Bereich der Künstlichen Intelligenz eingenommen hat.

Diese Arbeit führt zunächst in die notwendigen Grundlagen der CSPs und darauf aufsetzender Erweiterungen ein und befasst sich auch mit deren lokaler und verteilter Behandlung. Danach werden einige Veröffentlichungen besprochen, die CSPs für Probleme aus dem Bereich der Sicherheit in Rechensystemen verwenden. Darauf aufbauend wird versucht, gemeinsame Ideen und Merkmale zu finden, um zu präzisieren, für welche Arten von Sicherheitsproblemen sich die Behandlung mit CSPs anbietet.

Zwei parallel zur vorliegenden entstandene Diplomarbeiten beschäftigen sich ebenfalls mit der Anwendung von CSPs auf IT-Sicherheit: Burgard [Bur06] untersucht dazu die kontrollierte Anfrageauswertung für Datenbanken und Niggemeier [Nig06] die Eignung für Workflow-Management-Systeme (siehe auch Kapitel 4.4).

Abstract

In a computing environment, security is often described as the ability to maintain availability, confidentiality, integrity and authenticity for everyone involved. These basic concerns tend to contradict each other and in most cases, a trade-off is required to achieve a feasible solution. Since the early 2000s, several authors have proposed to view the handling of security-related decisions as constraint satisfaction problems (CSPs), a formalism that was originally developed to help tackle geometric problems and has since then turned into a promising field of artificial intelligence research.

This thesis starts with a description of the required foundations of CSPs and some of its extensions, along with an introduction to both local and distributed CSP solving. Subsequently, several current publications on the topic are presented and discussed. The thesis concludes with an attempt to find structural similarities in the publications with the goal of providing common characteristics of those security problems for which the application of CSPs appears to make sense.

Two theses which have been created along the present one also deal with the application of CSPs to a security context: Burgard [Bur06] addresses the controlled query evaluation for database systems and Niggemeier [Nig06] deals with the suitability of CSPs for secure workflow management (see chapter 4.4).

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

An Informations- bzw. Rechensysteme gibt es eine Vielzahl von Anforderungen. Neben der erwünschten Funktionalität, die zumeist die Motivation für die Implementierung derartiger Systeme darstellt, gilt es auch sekundäre Anforderungen zu erfüllen. Unter diesen zweitrangigen Vorgaben nehmen diejenigen, die sich mit Sicherheitseigenschaften eines Systems beschäftigen, einen zunehmend höheren Stellenwert ein. Zieht man in Betracht, dass viele traditionell auf zentralen Rechnern ausschließlich lokal zur Verfügung gestellte Dienste eine Erweiterung oder Abänderung erfahren, mit der sie über Netzwerke zugänglich werden, um einen größeren Nutzerkreis zu erreichen, so gewinnt der Wunsch nach Absicherung dieser Dienste an Wichtigkeit. Der dabei zu bewältigende Balanceakt bei der Abwägung verschiedener Interessen stellt die Entwickler und Betreuer dieser Systeme oft vor große Herausforderungen.

Idealerweise findet der Gedanke an mögliche Sicherheitsinteressen schon in der Entwicklungsphase Beachtung, sodass frühzeitig aktuellen Erkenntnissen und bekannten Angriffsmöglichkeiten Rechnung getragen werden kann. Beispielsweise stellt die unzureichende Prüfung von Nutzereingaben speziell bei über das Internet zugänglichen Diensten nach wie vor eine der größten Schwachstellen dar. Wird der nötige Aufwand der Eingabevalidierung von Beginn an bedacht, lassen sich viele Probleme lösen, bevor sie tatsächlich auftreten.

Was verstehen wir nun unter Sicherheitsinteressen? In der Literatur[1] zu Sicherheit in Informationssystemen finden sich folgende, sogenannte Schutzziele:

- Die eingangs erwähnte, gewünschte Funktionalität ist der Grund, warum Informationssysteme eingesetzt werden. Ein System muss jedoch auch tatsächlich *verfügbar* sein, damit Nutzer darauf zugreifen und von dessen Funktionalität profitieren können.
- Werden in einem Informationssystem Daten abgelegt, so wird zuweilen ein sehr umfangreicher Zugriffsschutz erwartet. Ein Nutzer möchte unter Umständen selbst festlegen können, wer auf die von ihm ins System eingestellten Daten zugreifen darf. Unter *Vertraulichkeit* verstehen wir, dass Informationen nur berechtigten Nutzern zugänglich sind.

[1] Mögliche Einstiegspunkte in den Themenkomplex geben Anderson [And01] (theoretische Grundlagen und Implementierungsrichtlinien sicherer Systeme), Schneier [Sch96] (vornehmlich Kryptographie und deren Anwendung) und Eckert [Eck06] (umfassendes deutsches Einführungswerk).

– Abgelegte Informationen sollen nicht ohne entsprechende Erlaubnis verändert werden können. *Integrität* ist vor allem für die Nachrichtenübermittlung und Datenarchivierung von besonderer Bedeutung.

– Es soll gegebenenfalls zweifelsfrei feststellbar sein, von welchem Teilnehmer Daten oder Nachrichten stammen. Dieses Schutzziel nennen wir *Authentizität*.

– Schließlich ist in einigen Szenarien wiederum die *Anonymität* der Teilnehmer ein schützenswertes Interesse, d. h., dass Handlungen im System nicht ohne weiteres individuellen Nutzern zugeordnet werden können.

Es ist leicht einzusehen, dass nicht alle Schutzziele gleichzeitig im maximal wünschenswerten Ausmaß erfüllt werden können, da diese häufig im Konflikt zueinander stehen. In einigen Fällen sind einzelne Schutzziele auch vernachlässigbar, was die Kompromissfindung vereinfacht.

Schon in der Entwicklungsphase eines Systems sollten also einige Gedanken auf grundlegende Sicherheitskonzepte verwendet werden. Die Summe der schließlich von einem System zur Verfügung gestellten Sicherheitsmechanismen nennen wir Sicherheitsarchitektur. Doch auch im eigentlichen Betrieb des Systems steht die Beschäftigung mit Sicherheit an. Grundlegend für die tatsächlich erworbene Sicherheit eines in Nutzung befindlichen Systems ist daher neben der Sicherheitsarchitektur die verwendete Sicherheitspolitik. Diese gibt vor, welche funktionalen (etwa lesender bzw. schreibender Zugriff auf Objekte des Systems) und kontrollierenden (etwa die Berechtigung zur Weitergabe eigener Erlaubnisse) Privilegien den Teilnehmern zugesprochen werden und welche Operationen damit möglich werden. Ausgereifte Politiken erlauben zudem, zusätzliche Bedingungen einfließen zu lassen, etwa um zwei im Konflikt miteinander stehende Operationen nicht von dem selben Teilnehmer ausführen zu lassen (z. B. sollte ein Richter nach gängiger Auffassung keine Gesetze erlassen dürfen).

Ein adäquates Sicherheitskonzept aus Architektur und Politik ist oftmals sehr individuell zu entwickeln, da bestehende Lösungen nicht ohne weiteres integriert werden können – ein Konzept, das in dem Umfeld, für das es entwickelt wurde, hervorragend funktioniert, kann ungeahnte Mängel aufweisen, wenn sich der Einsatzort ändert. Zudem treten auch bei der Komposition bestehender sicherer Systeme zu Netzwerken bisweilen neue Risiken in Erscheinung. Da sich die notwendigen Überlegungen zur Absicherung eines Systems nicht nennenswert automatisieren lassen, sind zumindest allgemeine Werkzeuge wünschenswert, die für bestimmte, wiederkehrende Probleme bei der Lösung behilflich sind.

Ein Kandidat für ein solches allgemeines Werkzeug sind Constraint-Satisfaction-Probleme (zuweilen auch als Constraintnetzwerke, CSPs oder Constraintprobleme bezeichnet). Die grundlegende Idee hinter diesem Verfahren lässt sich zusammenfassen als: *Beschreibe das Problem mit seinen Eigenschaften und denke nicht über die Lösungsmethoden nach.* Insbesondere sollen dabei keine konkreten Algorithmen zur Lösung bestimmter Proble-

me entwickelt oder implementiert, sondern das Problem möglichst scharf und knapp beschrieben werden. Seit Beginn der 2000er Jahre gibt es erste Bemühungen, CSPs zur Bearbeitung von Fragestellungen aus dem Bereich der Sicherheit in Rechensystemen zu verwenden. Zudem scheint sich dieses Thema auch wachsender Popularität zu erfreuen: 2005 und 2006 fanden zeitgleich mit der etablierten Konferenz *Constraint Processing* auch erste Tagungen statt, die sich explizit mit der Bedeutung von Constraintlösungstechniken für Probleme der IT-Sicherheit beschäftigen[2].

Im Folgenden widmen wir uns zunächst der Behandlung lokaler CSPs, um darauf aufbauend auch Methoden für verteiltes Lösen von Constraintproblemen zu untersuchen. Anschließend werden einige Ansätze vorgestellt, die sich den Formalismus der CSPs zu Nutze machen, um für die Sicherheit in Rechensystemen relevante Fragestellungen zu bearbeiten. Die Anwendungsfälle reichen von rechnergestützter Terminvereinbarung unter Berücksichtigung der Privatsphäre bis hin zur Verifikation kryptographischer Protokolle und Ideen für dynamische Workflow-Management-Systeme. Abschließend untersuchen wir, inwieweit CSPs als generelles Werkzeug im hier beschriebenen Kontext der Sicherheit dienen können. Dazu werden die strukturellen Merkmale von CSPs anhand der vorgestellten Ansätze untersucht.

[2] CPSec 2006 – 2nd International Workshop on Applications of Constraint Satisfaction and Programming to Computer Security: http://www.sci.unich.it/~bista/organizing/cpsec/index.html.

2 Lokale Constraintprobleme

Als lokale Constraint-Satisfaction-Probleme (im Folgenden auch CSPs oder Constraintprobleme genannt), bezeichnen wir solche Probleme, die lokal von einem Agenten bearbeitet werden. Die vorliegende Beschreibung der Constraintprobleme orientiert sich an denen von Mackworth [Mac87], Yokoo [Yok00] und Dechter [Dec92].

Definition 2.1 *Ein lokales Constraint-Satisfaction-Problem* (X, D, C) *besteht aus den Variablen* $X := \{x_1, \ldots, x_n\}$*, den zugehörigen Definitionsbereichen (domains)* $D := \{D_1, \ldots, D_n\}$ *und den Constraints* C*, die die möglichen Kombinationen von Variablenwerten einschränken.*

□

Es gilt eine Belegung für die Variablen aus X zu finden, sodass alle Constraints erfüllt sind.

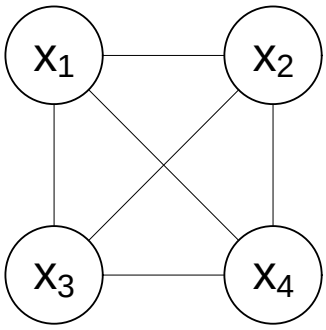

Abbildung 2.1: Constraint-Netzwerk für das 4-Damen-Problem

Oft wird ein Constraintproblem auch durch ein Netzwerk (*constraint network*) repräsentiert (vgl. u. a. [Dec92]). Hierbei stehen die Knoten für Variablen und die Kanten für Constraints zwischen diesen Variablen. Abbildung 2.1 zeigt ein Constraintnetz (auch Constraintgraph genannt) für das 4-Damen-Problem, wobei in diesem Beispielnetzwerk die üblicherweise an den Kanten annotierten Constraints fehlen. Zum Teil werden auch die Knotenbeschriftungen um die geltenden Domains erweitert. Derartige Netzwerke enthalten in der Regel *keine* reflexiven Kanten.

2.1 Semantik der Variablen

Die Variablen in Constraintproblemen erhalten eine zunächst nicht offensichtliche Doppelsemantik. Zum einen stellen schon die Namen der Variablen selbst Informationen dar. So steht beispielsweise in der Untersuchung von kaskadierenden Pfaden in MLS-Netzwerken (siehe dazu auch Kap. 4.3) eine Variable mit dem Namen S_4^s für ein System, das an der vierten Position in einem Pfad auftritt und eine sendende Funktion übernimmt. Welches der im konkreten Szenario betrachteten Systeme *tatsächlich* an der vierten Position im Pfad als Sender auftritt, ist damit jedoch noch nicht festgelegt, denn erst die Zuweisung eines konkreten Systems an die Variable lässt auch den Pfad konkret werden.

Als explizit definierte Notation für Belegungen sind uns lediglich die von Tsang [Tsa93] verwendeten *Labels* bekannt. Ein Label $\langle x, d \rangle$ beschreibt dabei die Zuweisung des Werts d an die Variable x.

Wird der Name einer Variablen genannt, so ist in der Literatur oft damit der eigentliche Wert der Variablen gemeint: $x_1 = 4$ bedeutet demnach, dass die Variable x_1 den Wert 4 erhalten hat. Wird in einem Constraint die Ungleichheit zweier Variablen gefordert, ist es üblich, dies als $x_1 \neq x_2$ zu schreiben. Was bedeutet aber $x \neq y$, wenn x und y Variablen sind? Während bei der Schreibweise mit Indizes noch deutlich ist, dass für verschiedene Variablen verschiedene Werte gefordert werden, ist im Fall $x \neq y$ nicht klar, ob die Ungleichheit auf syntaktischer Ebene (verschiedene Variablen) oder semantischer Ebene (verschiedene Werte) erwünscht ist. Aus diesem Grund wenden wir in Fällen, bei denen diese genaue Differenzierung sinnvoll erscheint, folgende Definition an:

Definition 2.2 *Für eine Variable x aus einem Constraintproblem bezeichnet $[\![x]\!]$ den Wert von x.*

□

2.2 Constraints

An die Struktur der Constraints wird allgemein keine besondere Anforderung gestellt. Einfache arithmetische Vergleiche sind ebenso denkbar wie komplexe Prädikate, wenn es darum geht, die strukturelle Beziehung zwischen Variablenwerten zu prüfen. Dieser intensionalen Beschreibung der Constraints steht die extensionale Darstellung gegenüber, bei der in sogenannten *Nogoods* eine verbotene Kombination von Variablenwerten konkret notiert wird. Während Nogoods entsprechend mehr Platz benötigen, so haben sie doch den Vorteil, dass derartige Constraints mit verschwindend geringer Rechenzeit überprüft werden können. Zu beachten ist dabei, dass ein Nogood in positiver Form aufgeschrieben wird, d. h. die nicht erlaubte Kombination von z. B. $x_1 = 1$ und $x_2 = 6$ wird als Nogood durch $\{(x_1 = 1; x_2 = 6)\}$ und nicht als $\{(x_1 \neq 1; x_2 \neq 6)\}$ ausgedrückt. Da alle Men-

gen von Variablenbelegungen, die Nogoods enthalten, ebenfalls Nogoods sind, ist man üblicherweise nur an *minimalen Nogoods* interessiert. Das bedeutet, dass keine Teilmenge des minimalen Nogoods selbst ein Nogood ist, da nur das simultane Auftreten *aller* im Nogood angegebenen Zuweisungen verboten ist. Andersherum bedeutet ein leeres Nogood, dass Überbeschränkung vorliegt (siehe auch Kap. 2.9), denn jede Hinzunahme einer beliebigen Menge von Variablenbelegungen zur Lösung enthält dennoch die leere Menge und ist damit wiederum verboten.

Ein- und zweistelligen Constraints kommt mitunter besondere Bedeutung zu:

Unäre Constraints werden vor allem dann verwendet, wenn der potenziell zugelassene Definitionsbereich aufgrund von während der Berechnung gefundenen Konflikten eingeschränkt werden soll, also für x_i nur eine Teilmenge $D_i' \subset D_i$ der prinzipiell erlaubten Werte tatsächlich in Frage kommt.

Binäre Constraints bestehen zwischen genau zwei Variablen. Da einige Algorithmen nur mit binären Constraints arbeiten können, ist es unter Umständen notwendig, das fragliche CSP in ein binäres CSP (d. h. eines, das nur noch unäre oder binäre Constraints enthält) umzuwandeln. Bacchus und van Beek [BB98] beschreiben zwei Verfahren, die eine derartige Transformation ermöglichen. Binäre Constraints lassen sich in zweidimensionalen Constraintgraphen als Kante darstellen (während höherstellige Constraints entsprechende Hyperkanten und Hypergraphen erfordern).

Ein sogenanntes *alldifferent*-Constraint ist auf einer Menge von Variablen $V' \subseteq V$ definiert und besagt: $\forall\, v, v' \in V'$ mit $v \neq v'$. $[\![v]\!] \neq [\![v']\!]$.

2.3 Konsistenzeigenschaften

Bei der Wahl der Belegungen für die verschiedenen Variablen eines CSPs kommt es zu Konflikten, wenn diese Belegungen Constraints verletzen und daher nicht miteinander verträglich sind. Für die Lösung eines CSPs ist man natürlich an einer Belegung ohne Konflikte interessiert, die als *konsistente* Belegung bezeichnet wird und in der folgenden Definition genauer beschrieben ist:

Definition 2.3 *Eine Belegung, die jedem $x_i \in X' \subseteq X$ einen Wert aus seinem zugehörigen Definitionsbereich D_i zuordnet und alle zwischen den Variablen aus X' geltenden Constraints erfüllt, wird als* ***konsistent*** *bezeichnet. Durch verletzte Constraints entstehen* ***Konflikte****.*

□

Offensichtlich erhält man für $X' = X$ mit einer konsistenten Belegung eine *Lösung* für das Problem. Der Konsistenzbegriff wird zusätzlich zu dieser allgemeinen Verwendung

auch schärfer definiert benutzt [Fre82]:

Definition 2.4 *Lässt sich für jede Belegung beliebiger (verschiedener) $k-1$ Variablen eines CSPs, in der alle Constraints zwischen diesen Variablen erfüllt sind, für jede k-te Variable eine Belegung derart finden, dass zwischen den nunmehr k Variablen alle Constraints erfüllt sind, so wird dieses CSP als* ***k-konsistent*** *bezeichnet. Ist ein CSP k-konsistent und j-konsistent $\forall\ j < k$, so spricht man von* ***starker k-Konsistenz*** *(strong k-consistency).*

□

Es mag der Anschein erweckt werden, dass das Vorhandensein von k-Konsistenz stets auch starke k-Konsistenz impliziert. Das folgende Gegenbeispiel [Fre82] zeigt, dass dies nicht der Fall ist. Wir verwenden eine Variante des Graph-Färbbarkeitsproblems GC-2 (siehe auch Kap. 2.5.2), bei dem – wie in Abb. 2.2 zu erkennen – die Domains von zwei der drei Knoten auf nur noch eine zur Verfügung stehende Farbe eingeschränkt sind. Das CSP ist als Constraintnetz dargestellt.

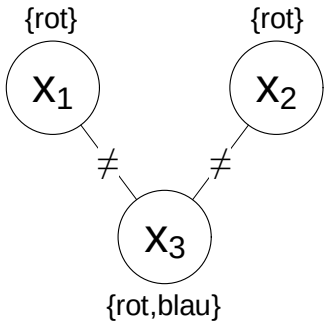

Abbildung 2.2: Eine Variante des GC-2-Problems mit drei Knoten

Dieses CSP ist zwar 3-konsistent, nicht jedoch 2-konsistent (wie durch starke k-Konsistenz gefordert). Anders gesagt: Ist eine konsistente Belegung für zwei der drei Knoten gefunden, so lässt sich auch eine mit diesen Belegungen verträgliche Belegung für den jeweils dritten Knoten finden. Wählt man jedoch „rot“ als Wert für x_3 – was eine konsistente Belegung für diesen Knoten darstellt – so ist für keinen der anderen beiden Knoten eine konsistente Belegung mehr möglich.

In der Literatur sind folgende Begriffe für Spezialfälle der k-Konsistenz üblich. Die Namen gehen auf die Behandlung von Constraintnetzen zurück:

$k = 1$**: Knotenkonsistenz** (node consistency) bezeichnet die Tatsache, dass in D_i nur Werte vorkommen, die keine unären Constraints für x_i verletzen. Es lässt sich also für jede Variable ein zulässiger Wert finden.

$k = 2$**: Kantenkonsistenz** (arc consistency) sagt aus, dass für eine knotenkonsistente Belegung von x_i auch eine mit den Constraints verträgliche Belegung für x_j gefunden werden kann, wenn x_i und x_j durch eine Kante im Constraintnetz verbunden sind (zwischen ihnen also ein Constraint zu erfüllen ist). Mit anderen Worten: Ist eine gültige Belegung für x_i gefunden, so lassen sich auch gültige Belegungen für alle Nachbarn x_j von x_i im Constraintnetz finden, ohne dass Constraints verletzt werden.

$k = 3$**: Pfadkonsistenz** (path consistency) ist schließlich gegeben, wenn von einem Knoten für Nachbarn bis in Tiefe 2 Werte zu finden sind, sodass zwischen den nunmehr 3 Knoten alle Constraints erfüllt sind. Dies entspricht *starker* 3-Konsistenz.

k-Konsistenz wird in der Regel dadurch erreicht, dass aus den D_i Werte entfernt werden, die für k relevante Konflikte verursachen.

Algorithmen, die bestimmte Konsistenzeigenschaften für ein Problem herstellen, werden häufig als Vorbereitung für einen Backtracking-Algorithmus verwendet, um den möglichen Suchraum einzuschränken. k-Konsistenzalgorithmen überführen dabei ein gegebenes CSP in eine äquivalente, k-konsistente Beschreibung.

Der Äquivalenzbegriff wird bei CSPs in der Regel so ausgelegt, dass zwei als äquivalent bezeichnete CSP-Formulierungen P_1 und P_2 eines Problems die selben Lösungen haben müssen. Rossi, Petrie und Dhar [RPD90] betrachten für eine allgemeiner gefasste Äquivalenz eine Überführung der Lösung von P_1 in eine Lösung für P_2 (und umgekehrt). Die Autoren sprechen von gegenseitiger Reduzierbarkeit (mutual reducibility), falls sich entsprechende Abbildungen zwischen den Variablen und Werten von P_1 und P_2 konstruieren lassen.

Definition 2.5 *Ist in einem Constraintnetzwerk die Wahl einer Belegung aus D_i für x_i abhängig davon, welchen Wert eine andere Variable x_j erhält, sagen wir verkürzend „x_i hängt von x_j ab“.*

Lassen sich die Abhängigkeiten zwischen den Variablen eines Constraintnetzwerks durch eine vollständige Ordnung auf den Knoten ausdrücken, wird das Netzwerk als ***geordnet*** *bezeichnet. Für die Vergleichsrelation dieser Ordnung verwenden wir das Symbol $\preceq$.*

□

Liegt ein geordnetes Constraintnetzwerk gemäß Def. 2.5 vor, so lassen sich in günstigen Fällen mittels k-Konsistenz sehr einfach Lösungen konstruieren (entsprechend Def. 2.6 und Satz 2.7 [Fre78, Fre82, Yok00]).

Definition 2.6 *Die* ***Breite eines Knotens*** *$width(x_i)$ in einem geordneten Constraintnetz gibt an, wieviele Kanten von x_i zu Knoten $x_j \prec x_i$ führen, die also gemäß der Ordnung kleiner als x_i sind. Die* ***Breite des Netzwerks*** *ist durch die größte Breite über*

alle Knoten bestimmt:

$$width(network) = max\,\{width(x) \mid x \in X\} \quad .$$

□

Satz 2.7 *Sei w die Breite eines stark k-konsistenten, geordneten Constraintgraphen. Ist $k > w$, so kann eine Lösung für das CSP durch Backtracking-freies Suchen entlang der Ordnung gefunden werden.*

Beweis Unabhängig davon, welche Variable x_i man betrachtet, es lässt sich stets für alle anderen Variablen $x_j \preceq x_i$, deren Belegung Einfluss auf die Wahlmöglichkeiten für x_i haben, eine konsistente Belegung untereinander finden. Da die Anzahl dieser Einfluss nehmenden Variablen maximal w ist und $k \geq w + 1$, lässt sich auch eine mit den bis dahin gefundenen Belegungen für die relevanten $x_j \preceq x_i$ konsistente Belegung für x_i finden.

□

2.4 Komplexität und Schwierigkeit

Im Grunde besteht die Lösung eines CSPs aus der Lösung eines Suchproblems. Unter allen möglichen Belegungen sind diejenigen interessant, die keine Constraints verletzen bzw. alle Constraints erfüllen. Verschiedene Fälle sind denkbar ($||M||$ bezeichnet dabei die Kardinalität der Menge M):

- Besitzt jede Variable x_i eine Domain D_i, so ist die Anzahl zu begutachtender Belegungen gegeben durch das kartesische Produkt der Domains $||D_1 \times D_2 \times \ldots \times D_n||$. Der tatsächliche Berechnungsaufwand hängt dann davon ab, wie schwierig die Auswertung der Constraints ist, da diese für jede Belegung zu überprüfen sind.
- Teilen sich alle Variablen in der Menge V ein und dieselbe Domain D, gibt es $||V \times D||$ Belegungen, die unter Umständen betrachtet werden müssen. Wie aufwändig die Begutachtung jeder Belegung ist, hängt wiederum von der Gestalt der Constraints ab.
- Das 3-SAT-Problem, für das die NP-Vollständigkeit nachgewiesen ist, lässt sich als CSP darstellen. Man muss also im schlechtesten Fall mindestens mit einem NP-vollständigen Problem bei der Lösung eines CSPs rechnen.
- Sind keine Constraints definiert oder ist starke n-Konsistenz gegeben, so ist jede mögliche Belegung eine erfüllende Belegung und eine Lösung lässt sich in $O(||V||)$ finden.

Letztendlich ist auch zu unterscheiden, ob *eine* Lösung genügt oder ob *alle* Lösungen gefunden werden sollen.

Intuitiv könnte man vermuten, dass CSPs umso schwieriger zu bewältigen sind, je mehr Constraints die möglichen Belegungen einschränken. Gibt es nur sehr wenige Constraints, lassen sich auch entsprechend viele gültige Belegungen finden. Eine große Anzahl Constraints jedoch lässt die um Heuristiken ergänzten Suchalgorithmen relativ schnell auf einen Konflikt stoßen und die Suche an dieser Stelle abbrechen, sodass auch diese Probleme eher einfach werden. Die schwierigsten CSPs finden sich im sogenannten *Phasenübergang* (*phase transition region*), wenn die Wahrscheinlichkeit, eine Lösung zu finden, unter 50% fällt.

2.5 Beispielprobleme

Um die in den Kapiteln 2.6 (Backtracking-) und 2.8 (Konsistenzverfahren) beschriebenen Algorithmen zu veranschaulichen, werden wir in einigen Fällen ein Beispiel für den Ablauf des jeweiligen Verfahrens geben. Dabei greifen wir auf eines der nachfolgend beschriebenen Beispielprobleme zurück.

2.5.1 Das N-Damen-Problem

Ein häufig verwendetes Beispielproblem für CSPs ist das *N-Damen-Problem* (*n-queens problem*), das aus der Aufgabe besteht, N Damen auf einem $N \times N$-Schachbrett so zu platzieren, dass sie sich gegenseitig nicht bedrohen (siehe auch zugehöriges Constraintnetz in Abb. 2.1). Es dürfen sich also keine zwei Damen eine horizontale, vertikale oder diagonale Linie auf dem Feld teilen. Wir betrachten nun eine einfache Formulierung dieses Szenarios als CSP, in der wir pro Zeile auf dem Spielfeld eine Variable verwenden; es ist also $X := \{x_1, \ldots, x_N\}$. Die Domains der einzelnen Variablen sind identisch, da für jede Dame zunächst alle Spalten in Frage kommen. Daher betrachten wir vereinfachend nur eine Domain $D := \{1, 2, \ldots, N\}$.[1]

Durch die zeilenweise Verwendung der Variablen ist bereits ausgeschlossen, dass sich zwei Damen in der selben Zeile befinden können. Die Constraints

$$\forall\, x, x' \in X \text{ mit } x \neq x'. \quad [\![x]\!] \neq [\![x']\!]$$

und

$$\forall\, r, r' \in D \text{ mit } r \neq r'. \quad \left|r - r'\right| \neq \left|[\![x_r]\!] - [\![x_{r'}]\!]\right|$$

schränken schließlich die vertikale (erstes Constraint) und diagonale Belegung (zweites Constraint) ein. Das erste Constraint fordert, dass für zwei Damen in unterschiedlichen

[1] Für die tatsächliche algorithmische Behandlung werden im Allgemeinen einzelne Domains benötigt. Die Vereinfachung dient nur dem Verständnis des Problems.

Zeilen auch unterschiedliche Spalten verwendet werden. Durch das zweite Constraint wird ausgeschlossen, dass der horizontale Abstand zweier Damen dem vertikalen Abstand entspricht, da in diesem Fall beide Figuren auf der selben Diagonalen liegen.

Im 4-Damen-Problem lässt sich dieses „Diagonalconstraint“ für x_1 durch eine Menge von Nogoods beschreiben: $\{(x_1 = 1, x_2 = 2), (x_1 = 2, x_2 = 1), (x_1 = 2, x_2 = 3), (x_1 = 3, x_2 = 2), (x_1 = 3, x_2 = 4), (x_1 = 4, x_2 = 3)\}$.

2.5.2 Das Graph-Färbbarkeitsproblem

Die Entscheidbarkeitsvariante des k-Färbbarkeitsproblems besteht darin, für einen Graphen festzustellen, ob sich die Knoten mittels k Farben so einfärben lassen, dass keine zwei durch eine Kante miteinander verbundenen Knoten die gleiche Farbe erhalten. Die Optimierungsvariante beschäftigt sich mit der Suche nach der minimal notwendigen Anzahl von Farben.

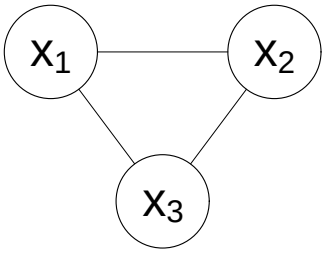

Abbildung 2.3: Constraintnetzwerk: GC-2 mit drei Knoten

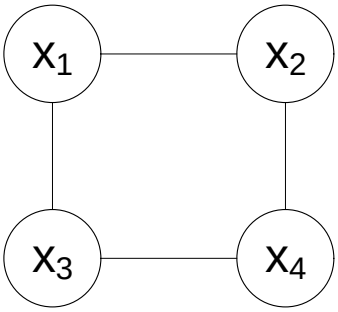

Abbildung 2.4: Constraintnetzwerk: GC-2 mit vier Knoten

Die Abbildungen 2.3 und 2.4 zeigen Instanzen für das 2-Färbbarkeitsproblem (GC-2), dargestellt als Constraintnetz, wobei die Variante mit drei Knoten zu stark beschränkt und damit keine entsprechende Belegung mit zwei Farben möglich ist.

Der zu färbende Graph lässt sich als Constraintnetz auffassen, welches – wie eingangs

in diesem Kapitel erwähnt – Variablen durch Knoten und Constraints durch Kanten repräsentiert. Da im Färbbarkeitsproblem je zwei durch eine Kante verbundene Knoten nicht die gleiche Farbe erhalten dürfen, verlangen alle Constraints, dass die an die entsprechende Kante angrenzenden Knoten nicht den selben Wert erhalten.

Ebenso wie beim N-Damen-Problem stehen auch hier für jede Variable prinzipiell alle Werte zur Verfügung. Daher enthält jedes D_i jeweils alle in der Eingabe des Färbbarkeitsproblems spezifizierten Farben.

2.6 Backtracking-Algorithmen

Backtracking ist eine Form der (erschöpfenden) Suche nach gültigen Belegungen. Das Verfahren entspricht einer natürlichen Herangehensweise an das Problem: Es wird eine partielle Lösung konstruiert, wobei in dieser alle relevanten Constraints erfüllt sind. Man beginnt mit einem Wert für x_i (aus D_i). Es werden nach und nach weitere Variablen hinzugenommen und derart mit Werten belegt, dass alle Constraints zwischen den gewählten Variablen erfüllt sind. Lässt sich keine konsistente Belegung für die neue Variable mehr finden, so wird für eine zuvor hinzugenommene Variable ein anderer Wert gewählt. Es ensteht so eine Tiefensuche auf dem Baum der möglichen Belegungen.

Entsteht ein Konflikt, ist die Wertauswahl der bisher zur partiellen Lösung gehörenden Variablen ungünstig und muss revidiert werden (*backtracking*). Um dabei möglichst die tatsächlich für das Problem verantwortliche Variable in ihrem Wert zu verändern, gibt es verschiedene Ansätze, die hier kurz vorgestellt werden sollen (vgl. dazu auch [Yok00]).

2.6.1 Min-Conflict-Backtracking

Bei der Initialisierung dieses Verfahrens [MJPL92] werden die Variablen mit Probewerten belegt, wobei in der Regel zufällige oder triviale (etwa $\forall\ i.\ x_i := d_1 \in D_i$) Probebelegungen verwendet werden. Anschließend werden nach und nach Variablen zur partiellen Lösung hinzugenommen und dabei gegebenenfalls revidiert. Ist eine solche Änderung im Bezug auf den Initialwert nötig, wird ein mit den vorläufigen Werten möglichst verträglicher Kandidat gewählt. Je mehr Constraints er erfüllt, bzw. je weniger er verletzt, desto verträglicher ist der Wert.

Dabei ist zu beachten, dass die *partielle Lösung immer konfliktfrei* sein muss, also Konflikte nur mit noch nicht in der Lösung befindlichen Variablen auftreten können. Lässt sich eine partielle Lösung nicht erweitern, weil die nächste Variable bei allen möglichen Werten Konflikte mit der bisher gefundenen Lösung produziert, wird die bis dahin gewonnene Lösung aufgegeben und als neues Nogood registriert. Dadurch wird verhindert, dass ein problematischer Suchpfad mehrfach in Betracht gezogen wird.

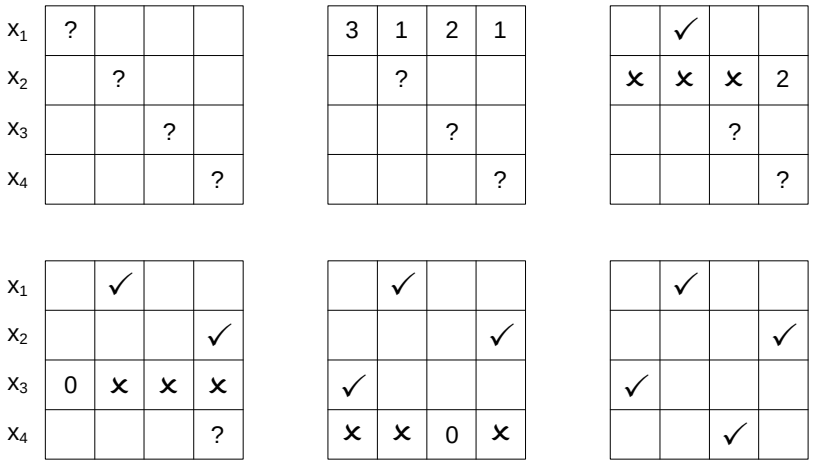

Abbildung 2.5: Min-Conflict-Backtracking am 4-Damen-Problem

Abb. 2.5 zeigt einen Beispiellauf für das 4-Damen-Problem. Häkchen stehen dabei für Teile der partiellen Lösung, Kreuze für Konflikte mit dieser (diese Möglichkeiten sind demnach verboten) und Zahlen geben jeweils die Anzahl der Konflikte mit den Probewerten an. Der Ablauf zeigt deutlich die Stärke dieser Heuristik: x_1 wählt direkt die zweite Spalte und findet sofort einen passendeneinen passenden Wert, was dazu führt, dass kein Backtracking angestoßen werden muss.

Es bleibt anzumerken, dass für die Bestimmung der Verträglichkeit je nach Problem möglicherweise sehr viele Constraints geprüft werden müssen. Ist die Auswertung der Constraints aufwändig, so kann der an dieser Stelle anfallende Rechenaufwand den Vorteil der Heuristik schnell zunichte machen.

2.6.2 Forward Checking

Beim *forward checking* [HE80] kommt der Reihenfolge bei der Hinzunahme weiterer Variablen in die partielle Lösung besondere Bedeutung zu. Problematische Suchpfade sollen möglichst früh Konflikte produzieren (*first-fail principle*), sodass unnötig spätes Backtracking vermieden werden kann. Ist beispielsweise die zuerst gewählte Variable verantwortlich für einen Konflikt, der erst sehr tief im Suchpfad auftritt, muss ein großer Teil des Suchraums durchlaufen werden, bevor die verursachende Variable einen anderen Wert erhält.

Um dem vorzubeugenwird ein Maß für die Beschränkungsstärke einer Variablen gebraucht. Drei Ideen dafür sind:

– Die Anzahl der Constraints, in der diese Variable vorkommt. Hierbei ist die bisher

gefundene Teillösung jedoch irrelevant.

- Die Anzahl der Kanten im Constraintnetz, die von einer möglichen, nächsten Variable zu Knoten aus der aktuellen Teillösung führen. Ein hoher Wert bedeutet hier auch höheres Konfliktpotenzial mit der Teillösung.
- Es werden Mengen für jede Variable verwaltet, die die mit der Teillösung verträglichen Werte beinhalten. Je kleiner die Kardinalität einer bestimmten Menge dieser Art ist, desto stärker ist die zugehörige Variable beschränkt.

Die Forward-Checking-Heuristik bedient sich der dritten Idee, wobei durch die Verwaltung derartiger Listen offensichtlich Overhead entsteht, ähnlich wie bei den in Kapitel 2.3 beschriebenen Konsistenzalgorithmen.

2.6.3 Backtracking mit Abhängigkeiten

Während die Forward-Checking-Heuristik versucht, die kritischsten Variablen zuerst zu bearbeiten, setzt das Backtracking mit Abhängigkeiten (*dependency-directed backtracking*) [SS77] an, wenn die Suche nach einer Belegung in eine Sackgasse geraten ist, d. h. es lässt sich kein mit der Teillösung verträglicher Wert für die nächste Variable finden.

Ist eine solche Sackgasse erreicht, sucht der Algorithmus nach der Ursache für die entstehenden Konflikte und setzt an der so ermittelten Variable an, um von dort aus erneut eine Suche zu starten. Da die genaue Fehlerbestimmung sehr aufwändig sein kann, ist ein Kompromiss zwischen der Genauigkeit der Analyse und der dafür benötigten Zeit zu finden.

2.7 Iterative Verbesserung

Unter dem Begriff *iterative Verbesserung* (*iterative improvement, hill-climbing*) werden im Kontext der CSPs Algorithmen gefasst, die weder Backtracking- noch Konsistenzverfahren sind. Dabei werden ähnlich wie beim Backtracking mit Abhängigkeiten (Kap. 2.6.3) alle Variablen zunächst versuchsweise mit vorläufigen Initialwerten belegt. Danach wird – bis eine Lösung gefunden wurde – eine konfliktbehaftete Belegung bearbeitet, die durch Änderung von Variablenwerten Schritt für Schritt verbessert wird. Etwas präziser: Ausgehend von einer Belegung, in der einige Constraints erfüllt und einige verletzt sind, wird nach einer Möglichkeit gesucht, durch die Änderung *eines* Variablenwerts möglichst viele weitere Constraints zu erfüllen.

Dadurch, dass immer nur *eine* Variable betrachtet wird, können sogenannte *lokale Minima* (*local minima*) entstehen, wenn die Änderung eines Werts zwar weitere Constraints

erfüllt, aber auch wieder bisher erfüllte verletzt. Durch die Änderung eines einzigen Werts kann in diesem Fall unter Umständen keine Verbesserung erreicht werden. Es bieten sich in einer solchen Situation zwei Methoden an:

- Der Algorithmus startet mit einer anderen Initialbelegung neu.
- Es wird die im Folgenden beschriebene *Breakout*-Technik benutzt, um aus dem lokalen Minimum auszubrechen.

Für die Breakout-Variante müssen zunächst alle Constraints als Nogoods gegeben sein. Diese ursprünglichen Constraints $c_i \in C$ werden mit Gewichten w_i versehen. Letztere werden zu Beginn mit 1 initialisiert und im weiteren Verlauf dann erhöht, wenn c_i an einem lokalen Minimum beteiligt ist. Bei der Begutachtung von Variablen wird nun versucht, $\sum_i w_i$ (die Summe der Gewichte, auch: Gesamtkosten) der verletzten Constraints zu verringern.

Die iterative Verbesserung mit der Breakout-Strategie ist nur partiell korrekt; es ist also nicht garantiert, dass das Verfahren terminiert.

2.8 Konsistenzalgorithmen

Backtracking-Algorithmen bringen zwei signifikante Probleme mit sich: Zum einen sind geeignete Heuristiken nötig, um nicht den gesamten Suchraum zu betrachten, und zum anderen kann durch wiederholtes Ändern von unproblematischen Werten (solchen, die nicht an einem Konflikt beteiligt sind) sogenanntes Flattern (*thrashing*) entstehen.

Konsistenzalgorithmen werden in der Regel vorbereitend (preprocessing) für Backtracking-Verfahren verwendet, um den Suchraum durch Fortlassen besonders problematischer Werte passend zu verkleinern und damit das Auftreten von Sackgassen-Situationen zu vermeiden. In der Literatur werden für Konsistenzalgorithmen auch die Begriffe *constraint propagation* (vgl. z. B. [Apt99]), *discrete relaxation*, *range restriction*, *domain elimination*, *filtering* (etwa der Algorithmus von Waltz, s. Kap. 2.8.1) und *full-forward look-ahead algorithms* verwendet.

2.8.1 Der Waltz-Filteralgorithmus

Der einfachste Algorithmus dieser Art ist der Filteralgorithmus von Waltz [Wal75]. Dieser entfernt aus den Domains diejenigen Werte, die nicht für eine Lösung in Frage kommen, wenn diese in jedem Fall Konflikte verursachen. Abb. 2.6 zeigt einen Ausschnitt aus dem 3-Damen-Problem, in dem x_2 den Wert 2 erhalten hat.

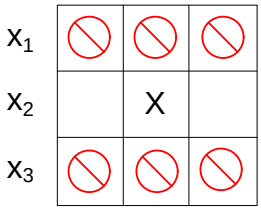

Abbildung 2.6: Mögliche Konflikte im 3-Damen-Problem

Obwohl der Wert prinzipiell erlaubt ist (da er in D_2 liegt), wird er durch den Filteralgorithmus verboten, da in diesem Fall keine konsistente Belegung für x_1 oder x_3 mehr möglich ist.

Im weiteren Verlauf wird Waltz' Algorithmus so viele Werte streichen, dass eine Domain am Ende leer ist. Tritt dieser Fall ein, so gibt es keine erlaubten Werte mehr für eine Variable, wodurch das Problem unlösbar wird. Auf diese Weise lässt sich Überbeschränkung durch den Filteralgorithmus feststellen. Analog lässt sich ableiten: Wenn alle Domains nur noch je ein Element enthalten, so ist eine eindeutige Lösung für das Problem gefunden.

Allerdings ist dieses Verfahren nicht in allen Fällen wirklich nützlich: Im 8-Damen-Problem lässt sich beispielsweise nichts eliminieren, da kein Wert für sich genommen die anderen Variablen so stark einschränkt, dass zwangsläufig ein Konflikt entsteht.

2.8.2 Der Auflösungsregel-Algorithmus

Wir gehen von einem CSP aus, in dem alle Constraints als Nogoods notiert sind. Die Auflösungsregel (*hyper-resolution rule*) [Kle89] stellt eine Möglichkeit zur Verfügung, aus mehreren Nogoods ein einzelnes neues zu konstruieren.

Variablenbelegungen, die Nogoods enthalten, stellen ebenfalls verbotene Werte-Kombinationen dar. Somit sind alle Obermengen von Nogoods wiederum Nogoods. Lässt sich nun mithilfe der Auflösungsregel ein leeres Nogood konstruieren, so ist daraus ablesbar, dass das Problem überbeschränkt ist und demzufolge keine Lösung besitzt.

2.8.3 Suche mit schwachen Zusagen

Basierend auf dem in Kapitel 2.6.1 vorgestellten Min-Conflict-Backtracking arbeitet die Suche mit schwachen Zusagen (*weak-commitment search*) [Yok00], die einen Hybrid aus Backtracking und iterativer Verbesserung darstellt. In einer Sackgasse – wenn es also keine Belegung mehr gibt, die ein konfliktfreies Weitersuchen mittels bisher erlangter partieller Lösung ermöglicht – wird jedoch nicht auf Backtracking zurückgegriffen, son-

dern die bis dort erreichte Teillösung als weiteres Nogood registriert. Das hat zur Folge, dass Variablen, die bereits alle Constraints erfüllen, nicht mehr in ihrem Wert verändert werden, demnach also kein Flattern mehr auftritt. Die fallengelassene Teillösung wird als neue, versuchsweise Initialbelegung benutzt, wodurch die bis dahin gefundenen, unproblematischen Werte übernommen werden können.

Durch eine relativ einfache Einschränkung produziert dieser Algorithmus im Vergleich zu anderen Verfahren deutlich bessere Laufzeiten. Die Breakout-Technik (siehe Kap. 2.7) erfordert im schlechtesten Fall, dass alle möglichen Werte für eine Variable betrachtet werden müssen, da insgesamt die Anzahl der Konflikte verringert werden soll.

Kommt der Min-Conflict-Algorithmus ohne Backtracking zu einer Lösung, so entstehen keine nennenswerten Laufzeitunterschiede zwischen diesem und der Suche mit schwachen Zusagen.[2] Schließlich lässt sich die Leistung des hier vorgestellten Verfahrens nochmals verbessern, indem Forward-Checking und das First-Fail-Prinzip (siehe Kap. 2.6.2) Anwendung finden.

Die Suche mit schwachen Zusagen arbeitet vollständig korrekt, da problematische Suchpfade nicht erneut betrachtet werden (s. [Yok00, S. 22]). Durch die neu entstehenden Nogoods kann der Speicherplatzbedarf signifikant steigen, was in der Praxis oft dadurch umgangen wird, dass die ältesten Nogoods nach und nach vergessen werden (wodurch andererseits die Vollständigkeit verloren geht).

2.9 Unendliche Domains und überbeschränkte Probleme

In CSP-Varianten, speziell den beschränkten Optimierungsproblemen (*constrained optimization problems*), wird nicht von diskreten und damit endlichen Domains ausgegangen. Mit der Unendlichkeit bzw. Stetigkeit der Domains geht einher, dass die bisher vorgestellten Algorithmen ungeeignet für die Behandlung derartiger CSPs sind. Russell und Norvig [RN03, S. 139ff.] erwähnen unter anderem lineare Programme als Spezialfälle von CSPs mit stetigen Domains.

Falls Constraints die Belegung der Variablen so stark einschränken, dass keine Kombination von Variablenwerten alle Constraints gleichzeitig erfüllen kann, spricht man von *überbeschränkten* (*over-constrained*, z. T. auch überspezifizierten) Problemen. Um Überbeschränkung handhaben zu können, sind spezielle Verfahren nötig, die nach einem Kompromiss zwischen Constrainterfüllung und vollständiger Lösung suchen.

[2] Yokoo vergleicht in einer empirischen Untersuchung die Performanz von Breakout, Min-Conflict-Backtracking und der Suche mit schwachen Zusagen ausführlich ([Yok00, S. 26f.]).

2.9.1 Partielle CSPs

Werden realitätsnahe Probleme als CSPs formuliert, so sind diese oft überbeschränkt. Die bis hierhin vorgestellten Algorithmen antworten auf derartige Instanzen jedoch nur, dass keine Lösung existiert, was in der Regel wenig hilfreich ist.

In diesen Fällen soll oft ein vertretbar relaxiertes Problem gelöst werden, das nur einen Ausschnitt des ursprünglichen Problems darstellt [FW92].

Definition 2.8 *Ein **partielles Constraint-Satisfaction-Problem** (PCSP) besteht aus dem ursprünglichen Problem, einer Halbordnung über relaxierten Problemen und einer Metrik [FW92]. Genauer:*

$P := (X, D, C)$ *dem ursprünglichen Problem, das in der Regel überbeschränkt ist.*

$(PS, \leq)$ *einer Halbordnung über CSPs (problem space). Die Menge PS enthält P und relaxierte CSPs Q. $\leq$ betrachtet die Lösungsmengen zweier Probleme aus PS. Ist $Q \leq Q'$, so sind alle Lösungen für Q in den Lösungen für Q' enthalten und Q ist stärker relaxiert als Q'.*

M *einer Abstandsfunktion auf PS, die die Entfernung zwischen zwei Problemen Q und Q' misst.*

(N, S) *notwendigem und hinreichendem Abstand zwischen P und einem relaxierten Problem Q. Ist $M(P, Q) < N$, so kann unter den möglichen Lösungen für Q eine akzeptable Lösung für das PCSP zu finden sein. Ist $M(P, Q) \leq S$, so ist jede Lösung für Q eine hinreichend gute Lösung für das PCSP.*

Eine Lösung für ein PCSP besteht aus einem relaxierten Problem $Q \in PS$ und einer Lösung für Q.

□

Die Abstandsfunktion dient als Maß für die Ähnlichkeit zweier CSPs aus PS. Die optimale Lösung ist gerade die notwendige Lösung, für die das relaxierte Problem Q am dichtesten am ursprünglichen Problem liegt, es also kein gemäß der Metrik näher an P liegendes Q gibt.

Man unterscheidet mehrere Klassen partieller CSPs:

- Ein *maximales* PCSP sucht nach einer Lösung, in der möglichst viele der ursprünglichen Constraints erfüllt sind.
- *Gewichtete* PCSPs sind etwas allgemeiner gefasst: Constraints werden mit Gewichten versehen und es gilt, die gewichtete Summe der erfüllten Constraints zu maximieren.

– Sind die Constraints nach Wichtigkeit geordnet, so können die weniger wichtigen Constraints bei der Relaxierung als erstes fallengelassen werden. In diesem Fall spricht man von *hierarchischen* PCSPs. Die Güte der Lösung ist durch das Gewicht des „teuersten“, entfernten Constraints bestimmt. Die Lösung ist also umso schlechter, je wichtiger dieses fallengelassene Constraint ist.

Für die algorithmische Behandlung derartiger CSP-Varianten bieten sich bekannte Techniken für Optimierungsprobleme an, etwa *branch-and-bound*-Suchalgorithmen. Freuder und Wallace haben einen verallgemeinerten Backtracking-Algorithmus vorgestellt [FW92], der sich wiederum durch z. B. Forward-Checking noch erweiteren lässt. Eine knappe Beschreibung dieser Technik findet sich auch in [Yok00, S. 44].

2.10 Weiche CSPs

Während partielle CSPs dafür ausgelegt sind, klassische CSPs in akzeptablen Grenzen zu relaxieren, bieten die von Bistarelli vorgeschlagenen *weichen* CSPs [Bis04] einen allgemeinen Ansatz, durch den sich alle gängigen Darstellungsformen der CSPs beschreiben lassen, darunter die klassischen CSPs und Fuzzy-CSPs. Die Idee dabei ist, die möglichen Belegungen für die Variablen zu qualifizieren. Diese Qualifizierung kann verwendet werden, um Gültigkeit, Güte, Verträglichkeit oder ähnliche Bewertungen einer Belegung der Variablen zu ermöglichen. Im Fall der einfachen, lokalen CSPs aus Definition 2.1 stehen nur die Bewertungen *gültig* und *ungültig* zur Verfügung. Durch die Qualifikation werden Belegungen (und damit auch Lösungen) des SCSPs deutlich differenzierter vergleichbar.

Bistarelli stützt seine Formalisierung der Soft Constraints auf algebraische Struktur eines Halbrings[3]:

Definition 2.9 *Ein Halbring* $(A, +, \times, \mathbf{0}, \mathbf{1})$ *besteht aus einer Menge* A *mit zwei Verknüpfungen* $+$ *und* $\times$*, sowie zwei ausgezeichneten Elementen* $\mathbf{0}, \mathbf{1} \in A$*.*

– *Die* ***additiv*** *genannte Operation* $+$ *ist assoziativ und kommutativ mit* $\mathbf{0}$ *als neutralem Element.*

– *Die* ***multiplikativ*** *genannte Operation* $\times$ *ist assoziativ, distributiv bezüglich* $+$ *mit* $\mathbf{1}$ *als neutralem Element und* $\mathbf{0}$ *als Nullelement.*

□

Definition 2.10 *Ein Constraint-Halbring (c-Halbring)* $(A, \sum, \times, \mathbf{0}, \mathbf{1})$ *ist ein spezieller Halbring mit den zusätzlichen Eigenschaften:*

[3] Eine ausführliche Darstellung des Frameworks mit Beweisen der geforderten Eigenschaften findet sich in [Bis04, Kap. 2]

- $\sum$ *ist auf möglicherweise unendlichen Teilmengen definiert*[4]*:* $\sum : \mathcal{P}(A) \to A$
- $\sum$ *ist idempotent:* $\forall\ a \in A.\ \sum(\{a\}) = a$
- $\sum(\emptyset) = \mathbf{0}$ *und* $\sum(A) = \mathbf{1}$
- $\sum$ *hält eine Form der Stetigkeit (flattening property) für alle Indexmengen* I *, die Teilmengen* $A_i \subseteq A$ *beschreiben:* $\sum\left(\bigcup_{i\in I} A_i\right) = \sum\left(\{\sum(A_i) | \, i \in I\}\right)$
- $\times$ *ist kommutativ*

□

Da die additive Operation im c-Halbring auf Mengen definiert ist, ist sie damit assoziativ und kommutativ. Die zusätzlichen Eigenschaften des c-Halbrings gegenüber den allgemeinen Halbringen werden nun verwendet, um eine Halbordnung zu definieren:

Definition 2.11 *Sei* $S = (A, \sum, \times, \mathbf{0}, \mathbf{1})$ *ein c-Halbring. Wir definieren eine zugehörige Halbordnung* $(A, \leq_S)$*: für zwei beliebige* $a, b \in A$ *ist* $a \leq_S b \iff \sum(\{a, b\}) = b$*.*

Bistarelli zeigt anschließend an diese Definition zunächst, dass + und $\times$ monoton über $(A, \leq_S)$ sind, d. h.:

$$\forall\ a, a', b \in A.\ a \leq_S a' \implies \begin{array}{l} a + b \leq_S a' + b \\ a \times b \leq_S a' \times b \end{array}$$

Ferner wird bewiesen, dass $\times$ *intensiv* ist, d. h. $a, b \in A \implies a \times b \leq_S a$. Zudem lässt sich zeigen, dass $(A, \leq_S)$ nicht nur eine Halbordnung, sondern sogar einen vollständigen Verband darstellt (in $(A, \leq_S)$ lässt sich die Existenz des Supremums für beliebige Teilmengen nachweisen, wodurch auch das Infimum existiert). Ist $\times$ ebenfalls idempotent, enspricht diese Operation der Infimumsoperation auf dem Verband, ist distributiv bezüglich + und bildet damit einen vollständigen, distributiven Verband. Es ist davon auszugehen, dass die Idempotenz der $\times$-Operation in den späteren Betrachtungen ebenfalls gefordert wird.

Mit diesen Eigenschaften lassen sich $\sum$ und $\times$ als Supremums- bzw. Infimumsoperation auf dem Verband auffassen. Dies führt uns zu folgender Beschreibung der Soft-Constraint-Satisfaction-Probleme.

Definition 2.12 *Ein Constraintsystem* $CS = (S, D, V)$ *besteht aus einem Constraint-Halbring* S*, einer endlichen Menge von Variablenwerten* D *(Domain) und einer endlichen, total geordneten Menge von Variablen* V*.* $(V, \leq_V)$ *beschreibt die totale Ordnung auf den Variablen.*

□

[4] $\sum$ ist auf Teilmengen von A definiert, nicht auf einzelnen Elementen. Bistarelli verwendet abweichend die Infix-Notation mit +, wenn nur zwei Elemente „addiert“ werden. Für $a, b \in A$ ist $\sum\{a, b\} = a + b$.

Definition 2.13 *Ein Constraint* (def, con) *auf einem Constraintsystem besteht aus einer Teilmenge der Variablen des Constraintsystems* $con \subseteq V$ *und einer Definitionsfunktion* $\text{def} : D^{||con||} \to A$*, die jeder möglichen Belegung der Variablen aus con eine Bewertung* $a \in A$ *zuordnet.* □

Definition 2.14 *Ein Soft-Constraint-Satisfaction-Problem* $(C(CS), con)$ *besteht aus einer Menge von Constraints bezüglich eines Constraintsystems* $C(CS)$ *und einer Teilmenge zu untersuchender Variablen* $con \subseteq V$*.* □

Neben der Bewertung von Belegungen ergibt sich mit dieser Formalisierung zudem die Möglichkeit, nicht alle Constraints zu betrachten, sondern nur einen Ausschnitt dieser. So lässt sich bei schrittweiser Hinzunahme von Constraints erkennen, welches Constraint unter Umständen für eine Überbeschränkung verantwortlich ist.

Um Constraints miteinander zu kombinieren, muss aus verschiedenen Tupeln der Form (def, con) ein neues Tupel dieser Art erzeugt werden. Das Zusammenführen der def-Funktionen geschieht durch Tupelprojektion. Dabei legen wir zugrunde, dass jede Teilmenge der Variablen sich wegen der totalen Ordnung $(V, \leq_V)$ auch als Tupel darstellen lässt, wobei die Indizes hier der Ordnung entsprechen, d. h. für zwei Elemente w_1 und w_2 ist $1 < 2$ und damit auch $w_1 \leq_S w_2$.

Definition 2.15 *Gegeben sei ein Constraintsystem* $CS = (S, D, V)$*. Wir betrachten ein* k*-Tupel von Variablenwerten* $t = (t_1, t_2, \ldots, t_k)$ *aus* D *und zwei Mengen von Variablen* $W = \{w_1, w_2, \ldots, w_k\}$ *und* $W' = (w'_1, w'_2, \ldots, w'_m)$ *mit* $W' \subseteq W \subseteq V$*.*

Die Projektion des Tupels t *von* W *nach* W' *(notiert als* $t \downarrow^W_{W'}$*) ist definiert als das Tupel* $t' = (t'_1, t'_2, \ldots, t'_m)$*, wobei* $t'_i = t_j$*, falls* $w'_i = w_j$*.*

□

Da von einem größeren auf ein kleineres Tupel projiziert wird, besteht die Tupelprojektion lediglich aus einer Auswahl von Werten. Der Laufindex j bezieht sich dabei auf die ursprüngliche Menge W und das ursprüngliche Tupel t, während i verwendet wird, um W' bzw. t' zu durchlaufen. Um das gewünschte Tupel $t' = t \downarrow^W_{W'}$ zu produzieren, werden nur die Werte aus t ausgewählt, die den Variablen zugeordnet sind, die sich sowohl in W als auch in W' befinden.

Dies führt uns zur Definition der Constraintkombination:

Definition 2.16 *Die Kombination zweier Constraints* $c_1 = (\text{def}_1, con_1)$ *und* $c_2 = (\text{def}_2, con_2)$ *zu einem neuen Constraint* $c = c_1 \otimes c_2 = (\text{def}, con)$ *ist definiert durch die Vereinigung der Variablenmengen*

$$con = con_1 \cup con_2$$

und das Produkt der Zuordnungsfunktionen bezüglich des zugehörigen Constraint-Halbrings:

$$\mathrm{def}(t) = \mathrm{def}_1\left(t \downarrow_{con_1}^{con}\right) \times \mathrm{def}_2\left(t \downarrow_{con_2}^{con}\right)$$

□

Schließlich ist zuweilen noch gewünscht, nur einen Ausschnitt aus einem kombinierten oder einfachen Constraint zu betrachten:

Definition 2.17 *Sei $c = (\mathrm{def}, con)$ ein Constraint bezüglich eines Constraintsystems $CS = (S, D, V)$ und $I \subseteq V$. Die Constraintprojektion von c auf eine Teilmenge der Variablen ergibt ein neues Constraint $c' = (\mathrm{def}', con')$ mit $con' = con \cap I$ und*

$$\mathrm{def}'(t') = \sum \left\{\mathrm{def}(t) \mid t' = t \downarrow_{con'}^{con}\right\}$$

□

Da die in der Kombination der Constraints verwendeten Verknüpfungen $\cup$ und $\times$ beide assoziativ und kommutativ sind, kann auch eine Menge von Constraints C gleichzeitig kombiniert werden. Wir verwenden dafür die Operation $\bigotimes C$.

Abb. 2.7 zeigt einen Ausschnitt aus einem SCSP, welches eine Variante des 3-Damen-Problems darstellt. Im oberen Teil der Abbildung werden zunächst die benötigten Mengen und Ordnungen angegeben. Im unteren Teil finden sich die def-Funktionen für die beiden Constraints c_1 und c_2, sowie die resultierende def-Funktion, wenn die Constraints kombiniert werden.

Wie in Kapitel 2.8.1 bereits erwähnt, ist das 3-Damen-Problem überbeschränkt und damit gibt es keine Belegung, die alle Constraints erfüllt. Vielleicht genügt aber schon eine Belegung, in der nicht mehr als 2 Konflikte bestehen? Die Menge der Qualifizierungen im Constraint-Halbring besteht aus drei Elementen. Erhält eine Belegung die Qualifizierung „2“, ist sie (bezogen auf das jeweilige Constraint) konfliktfrei; mit der Qualifizierung „0“ verursachen alle Variablen Konflikte. Der Fall zwischen diesen Extremen, indem einige Variablen einen Wert erhalten, der Konflikte mit einer anderen Variablen erzeugt, andere Variablen wiederum keine Konflikte produzieren, erhält die Qualifizierung „1“.

In dem gezeigten Ausschnitt sind alle Belegungen aufgeführt, in denen x_1 den Wert 1 erhält – ein Wertetupel (a, b, c) entspricht hierbei der Belegung $x_1 = a; x_2 = b; x_3 = c$. Was das Spaltenconstraint c_1 anbelangt, ist nur die Belegung $(1, 1, 1)$ derart problematisch, dass hierfür die Bewertung „0“ vergeben wird. Im Diagonalconstraint erhalten die Belegungen $(1, 2, 1)$ und $(1, 2, 3)$ diese Bewertung. Da die in den beiden Constraints betrachteten Variablenmengen identisch sind, besteht die Kombination $c_1 \otimes c_2$ lediglich aus einer Infimumsbildung der Qualifizierungen für jedes Tupel (bezüglich $(A, \leq_S)$). Im Ergebnis finden sich in der Kombination der Constraints nur noch die Qualifizierungen 0

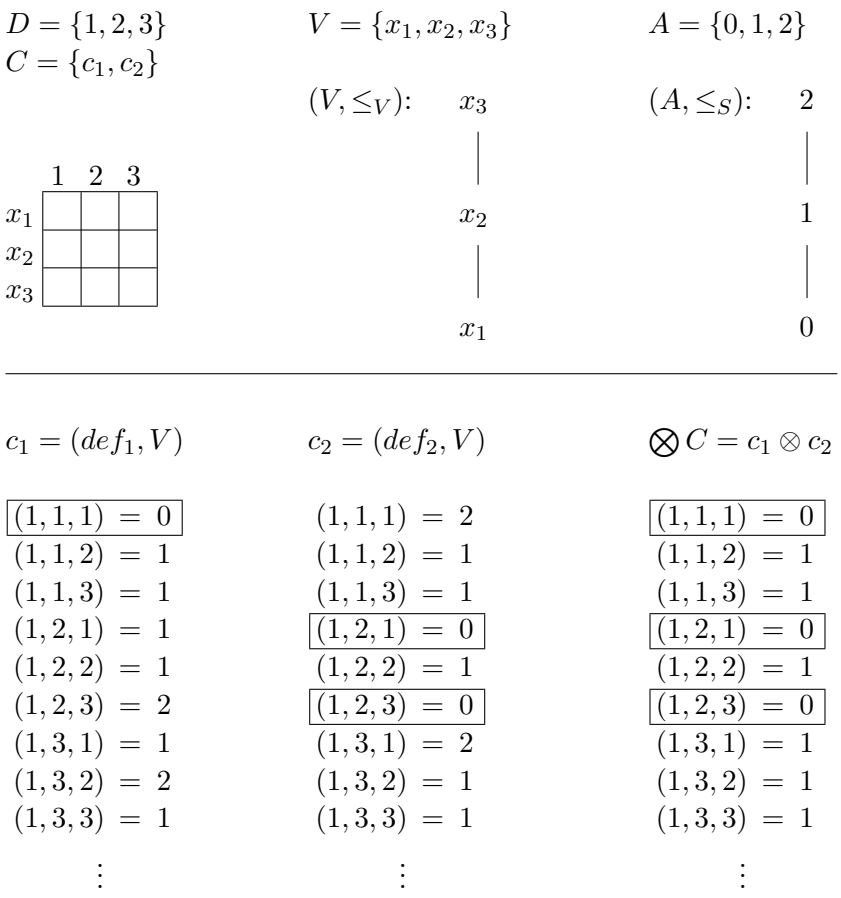

Abbildung 2.7: Ausschnitt aus einem SCSP für das 3-Damen-Problem

und 1. Aufgrund der Symmetrie des Problems wird auch in den Tupelblöcken mit $x_1 = 2$ und $x_1 = 3$ jeweils drei Belegungen die Bewertung 0 zugeordnet. Schließlich bleiben von 27 möglichen Belegungen am Ende 18 übrig, die nicht die Bewertung 0 erhalten haben, und keine dieser 18 Belegungen hat die beste Bewertung 2 erhalten. Letzteres ist bereits durch die Überbeschränkung im Sinne der klassischen CSPs bedingt.

Zu beobachten ist, dass die Belegungen mit der niederwertigsten Qualifizierung durch die Infimumsbildung auch nicht mehr in der Kombination der Constraints in Frage kommen. Auf diese Weise lassen sich leicht „K.O.“-Kriterien definieren, wenn man das kleinste Element aus $(A, \leq_S)$ als die Qualifizierung „keine Lösung“ auffasst.

Abschließend sei an dieser Stelle noch erwähnt, dass Bistarelli [Bis04] genau ausführt, wie verschiedene Formalisierungen von CSPs in entsprechende SCSPs übertragen werden

können. Der Constraint-Halbring für die klassischen CSPs ist beispielsweise:

$$(\{\text{true}, \text{false}\}, \vee, \wedge, \text{false}, \text{true})$$

Damit ist implizit auch die Ordnung $(A, \leq_S)$ gegeben: false $<_S$ true. Bei der Kombination von Constraints wird also das logische Und verwendet, da zusätzliche Bedingungen hinzukommen. Das Oder dient der Projektion.

3 Verteilte Constraintprobleme

Die Behandlung großer Constraintprobleme kann sehr aufwändig sein, da die Zeit- und Speicherplatzkomplexität im schlimmsten Fall exponentiell in der Anzahl der Constraints ist. In solchen Fällen bietet es sich an, über eine verteilte Bearbeitung der CSPs nachzudenken. Im Folgenden wird eine leicht modifizierte Variante der in [Yok00, S. 47ff.] und [YSH02] vorgestellten Formalisierung verteilter Constraintprobleme eingeführt. Die im weiteren Verlauf erklärten Algorithmen, die zumeist auf bereits vorgestellten, lokalen Algorithmen beruhen, werden auf der Grundlage dieser Beschreibung arbeiten.

Für die verteilten Constraintprobleme werden automatisierte Agenten eingeführt, die sich mit einzelnen Variablen befassen. Constraints werden dann zwischen den Agenten behandelt, sodass durch die Erfüllung dieser Interagenten-Constraints das Problem gelöst wird. Anwendungsprobleme in sogenannten Multiagentensystemen (MAS) lassen sich häufig als verteilte CSPs formulieren.

Definition 3.1 *Ein **verteiltes Constraintproblem** (distributed constraint satisfaction problem, DCSP) (A, X, D, C) besteht aus Agenten $A := \{a_1, \ldots, a_k\}$, Variablen $X := \{x_1, \ldots, x_n\}$, einer Domain $D := \{d_1, \ldots, d_m\}$ und den zwischen den Variablen zu erfüllenden Constraints C.*

□

Das Prädikat belongs(x, a) drückt aus, dass die Variable x dem Agenten a zugeordnet ist. In der Regel werden Agenten und Variablen als identisch angenommen, a_i beschäftigt sich also mit x_i, und es gilt $k = n$.

Für die Constraints gelten folgende Zusammenhänge:

- Alle Constraints sind als unäre oder binäre Nogoods notiert.
- C_x^a-Teilmengen von C enthalten die Constraints, die dem Agenten a bekannt sind und eine Aussage über die Variable x treffen. Ein Agent a kennt nur die unären Constraints $c(x)$, in denen seine Variable x vorkommt, d. h.

$$c(x) \in C_x^a \iff \text{belongs}(x, a)$$

– $C^a_{x_i,x_j}$-Teilmengen von C enthalten die Constraints, die dem Agenten a bekannt sind und von denen x_i oder x_j zu a gehört. Entsprechend kennt a nur die binären Constraints, die seine Variable einbeziehen, d. h.

$$c(x_i, x_j) \in C^a_{x_i,x_j} \iff \text{belongs}(x_i, a) \vee \text{belongs}(x_j, a)$$

Die unären Nogoods der Form $(x_i = d_j)$, schließen für x_i mögliche Werte d_j aus dem Universum D aus. Binäre Nogoods $(x_i = d_j; x'_i = d'_j)$ verbieten wiederum simultane Zuweisungen an verschiedene Variablen.

Für die in den nächsten Abschnitten beschriebenen Algorithmen nimmt Yokoo [Yok00, S. 47] ein einfaches Kommunikationsmodell an:

– Agenten kommunizieren miteinander, indem sie Nachrichten verschicken.

– Ein Agent kann einen anderen Agenten benachrichtigen, wenn er die Identifikationsmerkmale dieses Agenten kennt (z. B. IP-Adresse).

– Die Sendeverzögerung sei zufällig, aber endlich.

– Für alle Paare von Agenten entspricht die Empfangsreihenfolge der Sendereihenfolge. Zwei Nachrichten kommen also in der gleichen Reihenfolge beim Empfänger an, in der sie verschickt wurden.

Satz 3.2 *Jedes lokale CSP (X, D, C) lässt sich in ein äquivalentes DCSP (A, X, D', C') umwandeln.*

Beweisskizze Da DCSPs formal nur auf unären und binären Nogoods definiert sind, muss ein CSP, das Constraints mit Stelligkeit > 2 enthält, zunächst in ein binäres CSP transformiert werden (siehe auch Kap. 2.2).

Die Variablen können direkt übernommen werden; die Agenten werden entsprechend erzeugt, d. h. für jede Variable x_i wird ein Agent a_i eingeführt, der für x_i zuständig ist. Dadurch ist auch das belongs-Prädikat gegeben.

Die unären und binären Constraints werden ebenfalls direkt übernommen und in die zugehörigen Mengen $C^{a_i}_{x_i}$, $C^{a_i}_{x_i,x_j}$ und $C^{a_j}_{x_i,x_j}$ aufgenommen.

Es bleibt zu zeigen, dass auch die Übertragung der Domains gelingt. Dazu wird zunächst $D' = \bigcup_i D_i \in D$ konstruiert. Die Einschränkungen für die einzelnen D_i werden in weiteren, unären Constraints kodiert: $\forall\, d_j \in D \setminus D_i.\ (x_i = d_j)$ und ebenfalls in $C^{a_i}_{x_i}$ registriert.

□

Häufig werden verteilte Constraintprobleme durch Netzwerke ähnlich den Constraintnetzen repräsentiert, wobei die Knoten für Agenten (und damit meist auch wieder für Variablen) stehen und die Kanten binäre Constraints anzeigen. Diese Darstellung legt den Begriff der *Nachbarn* nahe:

Definition 3.3 *Zwei Agenten heißen* ***benachbart****, wenn sie im zugehörigen Constraintnetz durch eine Kante verbunden sind, demnach also ein binäres Constraint teilen. Das Prädikat* neighbours(x_i) *liefert die Menge aller Nachbarn von* x_i*.*

□

3.1 Das verteilte 4-Damen-Problem

DCSPs finden unter anderem Anwendung im Erkennungsproblem (*recognition problem*), bei dem nach einer konsistenten Interpretation von Rohdaten gesucht wird und in der Ressourcenzuordnung (*allocation problem*), wo es gilt, für eine parallele Abarbeitung von Aufgaben einen Plan zu finden. Um die Unterschiede zur lokalen Variante zu illustrieren, sehen wir uns die verteilte Version des 4-Damen-Problems an.

Wie in Kapitel 2.5.1 besteht das 4-Damen-Problem zunächst aus den zeilenweise vergebenen Variablen $X := \{x_1, \ldots, x_4\}$. Hinzu kommen die Agenten $A := \{a_1, \ldots, a_4\}$, denen jeweils die entsprechende Variable zugeordnet ist: belongs $= \{(x_1, a_1), \ldots, (x_4, a_4)\}$.

Ferner gilt für alle a_i, a_j mit $i < j$, dass $C^{a_i}_{x_i,x_j}$ die Diagonal- und Spaltenconstraints enthält, z. B.:

$$\begin{aligned} C^{a_1}_{x_1,x_2} = \quad & \{(x_1 = 1; x_2 = 2), (x_1 = 2; x_2 = 1), (x_1 = 2; x_2 = 3), \\ & (x_1 = 3; x_2 = 2), (x_1 = 3; x_2 = 4), (x_1 = 4; x_2 = 3)\} \\ \cup \quad & \{(x_1 = 1; x_2 = 1), (x_1 = 2; x_2 = 2), (x_1 = 3; x_2 = 3), (x_1 = 4; x_2 = 4)\} \end{aligned}$$

3.2 Einfache Algorithmen

Neben einer zentralisierten Methode, bei der ein Agent die Rolle des Leiters übernimmt und nach initialem Sammeln von Informationen das Problem traditionell löst, ist eine synchrone Backtracking-Variante denkbar. Dafür vereinbaren die Agenten zu Beginn eine Instanzierungs- oder Belegungsreihenfolge, die der Einfachheit halber den Indizes der Variablen bzw. Agenten entspricht; es sind aber auch andere Anordnungen möglich. Der erste Agent wählt einen gültigen Wert für seine Variable und sendet sie an a_2. Empfängt Agent x_i (für $i > 1$) nun eine partielle Lösung, so versucht er eine konsistente Zuweisung für seine Variable zu finden. Gelingt dies, wird die erweiterte Belegung an x_{i+1} geschickt. Andernfalls sendet a_i eine Backtracking-Nachricht an a_{i-1}.

Zweifelsohne haben diese beiden Ansätze deutliche Nachteile und stellen keine ernstzunehmenden Verfahren für die Behandlung von DCSPs dar. Bei der zentralisierten Methode häufen sich Daten beim Leiter an und beide Algorithmen kranken an fehlender Parallelität.

3.3 Asynchrones Backtracking

Wie einleitend in diesem Kapitel bemerkt, werden Variablen und Agenten synonym verwendet. Vorbereitend für dieses Verfahren werden streng monotone Prioritäten vergeben, sodass für das 4-Damen-Problem der in Abb. 3.1 gezeigte Prioritätsgraph entstehen könnte. Die Kanten zeigen stets von einem Agenten höherer Priorität zu einem mit geringerer Priorität. (In diesem Fall bedeutet ein größerer Index an einer Variable eine kleinere Priorität.)

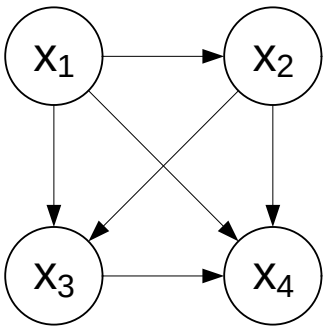

Abbildung 3.1: Asynchrones Backtracking: Prioritätsgraph für das 4-Damen-Problem

Die Agenten agieren in zwei Rollen: zum einen als Wertesender und zum anderen als Constraint-Auswerter. Ein Wertesender schickt einen Vorschlag für seinen eigenen Wert an alle Agenten mit geringerer Priorität. Dies geschieht bei Yokoo [Yok00, S. 58ff.] mittels sog. *ok?*-Nachrichten.

Die Agenten, die eine *ok?*-Nachricht empfangen, prüfen, ob für diesen neuen Wert, zusammen mit allen anderen bekannten Werten, eine konsistente Belegung für ihren eigenen Wert möglich ist. Sie übernehmen damit die Rolle des Constraint-Auswerters. Ist der aktuelle Wert des Empfängers verträglich mit den bekannten, anderen Werten, so schweigt der Agent. Lässt sich ein entstehender Konflikt durch Änderung eines eigenen Werts auflösen, aktualisiert der Agent seine Belegung und schickt entsprechende *ok?*-Nachrichten.

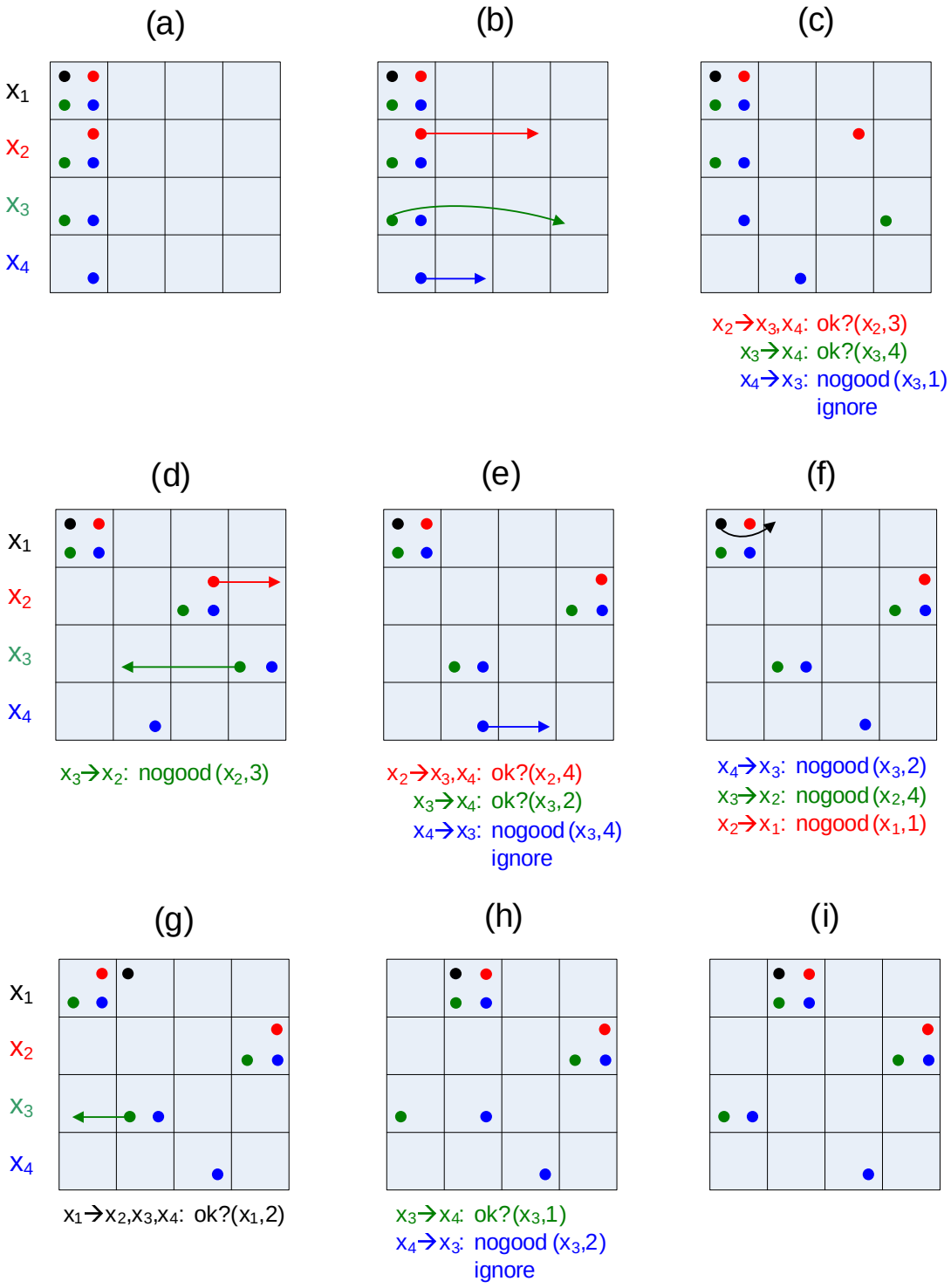

Abbildung 3.2: Asynchrones Backtracking am 4-Damen-Problem

Bleibt der Konflikt unauflösbar, wird eine *nogood*-Nachricht verschickt. Damit wird der höherpriorisierte Agent, der den Konflikt mitverursacht, davon in Kenntnis gesetzt, dass der von diesem gewählte Wert problematisch ist. Die Nachricht enthält ein minimales Nogood (s. Kap. 2.2), das den Konflikt beschreibt.

In Abb. 3.2 wird *ein möglicher* Beispiellauf für das 4-Damen-Problem vorgestellt.[1] Bedingt durch die mögliche Parallelität sind auch verschiedene andere Abläufe denkbar. Die Grafik unterscheidet dabei zwischen den tatsächlichen Positionen auf dem Spielbrett und den von den Agenten geringerer Priorität „geglaubten“ Positionen. Ein Beispiel: In Abb. 3.2 (c) glauben x_3 und x_4, dass sich x_2 noch in der ersten Spalte befindet. Ihr Wissen ist durch den grünen (x_3) und den blauen (x_4) Punkt in der Zeile von x_2 dargestellt. Die Abweichung entsteht dadurch, dass x_2 seinen Wert bereits geändert hat, die Information über die modifizierte Belegung jedoch noch nicht bei x_3 und x_4 angekommen ist. Der tatsächliche Wert von x_2 ist also durch den der Agentenfarbe entsprechenden Punkt in der eigenen Zeile ausgedrückt (zweite Zeile, dritte Spalte, roter Punkt für x_2). Insbesondere wissen Agenten nichts über die Werte von Agenten mit einer kleineren Priorität als ihrer eigenen.

Abb. 3.2 (a) zeigt die Ausgangsbelegung. In (b) ändern die Agenten für x_2, x_3 und x_4 ihre Werte und senden diese in (c) an die entsprechenden Nachbarn (*ok?*-Nachrichten). Gleichzeitig stellt x_4 fest, dass er durch $x_3 = 1$ behindert wird und schickt ein entsprechendes Nogood an x_3, was von diesem jedoch ignoriert wird, da er seinen Wert bereits geändert hat.

In (d) reagiert x_2 auf das Nogood, das dieser von x_3 erhält und wechselt in die vierte Spalte, während x_3 seinen Wert auf 2 ändert. Schritt (e) beschreibt das Versenden der aktualisierten Werte von x_2 und x_3 und das Absetzen einer weiteren Nogood-Nachricht durch x_4, die abermals ignoriert wird. Angekommen bei der Belegung in Zustand (f) verkündet zunächst x_4, dass er nicht mit $x_3 = 2$ zufrieden ist. Da x_3 jedoch keinen konsistenten Wert finden kann, (1 wegen x_1, 2 wegen x_4, 3 und 4 wegen x_2), gibt er das Backtracking an x_2 weiter. Da für x_2 in Schritt (d) bereits die dritte Spalte verboten wurde und durch x_1 die Werte 1 und 2 ausgeschlossen sind, sendet er eine Nogood-Nachricht an x_1. Für x_1 ist 2 der nächstbeste Wert, was er den anderen Agenten in (g) mitteilt. Als die Nachricht x_3 erreicht, wechselt dieser in die erste Spalte. Der aktualisierte Wert wird verschickt (Schritt (h)) und schließlich ist im letzten Bild eine Lösung gefunden.

Der hier vorgestellte Algorithmus ist beweisbar vollständig, wenn zusätzliche Techniken benutzt werden, um festzustellen, ob eine Lösung gefunden ist. Überbeschränkung (siehe Kap. 2.9) lässt sich erkennen, falls im Verlauf der Berechnungen ein leeres Nogood erzeugt wird. Die Laufzeit ist proportional zur Anzahl der erzeugten Nogoods, von denen maximal $|D_i|$ viele entstehen können.[Yok00, S. 64ff.]

[1] Falls die vorliegende Version keine farbige Abbildung enthält, verwenden Sie einfach die folgende Kodierung: Die möglichen vier Punkte in jeder Zeile sind so angeordnet, dass der Punkt oben links zu x_1 gehört, der oben rechts zu x_2 und analog der untere linke Punkt zu x_3, der Punkt unten rechts zu x_4.

3.4 Asynchrone Suche mit schwachen Zusagen

Basierend auf asynchronem Backtracking lässt sich eine verteilte Version der Suche mit schwachen Zusagen (Kap. 2.8.3) entwickeln [Yok00, S. 69ff.]. Die grundlegende Idee besteht dabei darin, dass die erschöpfende Suche unterhalb eines Agenten hoher Priorität entfällt, falls dieser einen Konflikt verursacht.

Ausgehend von einer initialen Priorisierung der Agenten wird diese dynamisch geändert, falls ein Agent einen Konflikt mit einem höherrangigen Agenten findet. Der den Konflikt entdeckende Teilnehmer erhält nun die höchste Priorität und kann dadurch Wertänderungen bei vormals wichtigeren Agenten bewirken.

Folgende Änderungen werden gegenüber dem asynchronem Backtracking eingeführt:

- Die Min-Conflict-Heuristik sorgt dafür, dass der verträglichste Wert mit der momentan bekannten Belegung der anderen Variablen gewählt wird.
- Es werden (schwach) monotone Prioritätswerte $\text{prio}(x_i) \in \mathbb{N}_0$ hinzugefügt. Dabei bedeuten größere Werte auch eine höhere Priorität. Für die Vorrangbestimmung zweier Agenten gleicher Priorität ist eine entsprechende Regel nötig (z. B. lexikographisch).
- Initial gilt: $\forall\ i.\ \text{prio}(x_i) := 0$.
- Gibt es keinen konsistenten Wert für x_i, wird die Priorität in Bezug auf die Nachbarn entsprechend erhöht. $\text{maxprio}(x_i)$ bezeichnet dabei die derzeit höchste Priorität aller Nachbarn von x_i.

$$\nexists \text{ konsistenter Wert für } x_i \rightsquigarrow \text{prio}(x_i) := \text{maxprio}(x_i) + 1$$

- *ok?*-Nachrichten, mit denen neue Werte übermittelt werden, werden um die eigene Priorität ergänzt und an *alle* Nachbarn gesendet, nicht nur an Nachbarn mit geringerer Priorität.
- Jeder Agent merkt sich die von ihm gesendeten Nogoods.

Findet ein Agent nun keinen verträglichen Wert, so berechnet er die zu versendenden Nogoods und gleicht diese Constraints mit der Liste bereits gesendeter Nogoods ab. Nur falls alle Nogoods „frisch“ sind, fährt er wie folgt fort: Zunächst werden die Constraints an alle Nachbarn geschickt. Dann erhöht der Agent seine eigene Priorität auf

$$max\{\text{prio}(x_j) \mid x_j \in \text{neighbours}(x_i)\} + 1\ ,$$

wählt anschließend den verträglichsten Wert (gemäß Min-Conflict-Heuristik) und verschickt diesen neuen Wert mithilfe von *ok?*-Nachrichten an alle seine Nachbarn.

Es ist möglich, dass während der Bearbeitung des Problems ein leeres Nogood von einem der beteiligten Agenten generiert wird. In diesem Fall informiert der Agent seine Nachbarn über dieses Ereignis und terminiert. Auf diese Weise kann die asynchrone Suche mit schwachen Zusagen Überbeschränkung feststellen.

In einem empirischen Laufzeittest vergleicht Yokoo das asynchrone Backtracking (mit und ohne Min-Conflict-Heuristik) mit dem hier beschriebenen Algorithmus. Die teils signifikanten Laufzeitunterschiede zeigt Tabelle 3.1 auf.

Algorithmus		n		
		60	90	120
Asynchrones Backtracking	Quote	13%	0%	0%
	Zyklen	917.4	—	—
Asynchrones Backtracking mit Min-Conflict-Heuristik	Quote	12%	2%	0%
	Zyklen	937.8	994.5	—
Asynchrone Suche mit schwachen Zusagen	Quote	100%	100%	100%
	Zyklen	59.4	70.1	106.4

Tabelle 3.1: Laufzeitvergleich nach [Yok00, S. 76, Tab. 4.2]

Die Rahmenbedingungen für die Tests definieren die Zyklen (cycles), die im Wesentlichen einzelnen Bearbeitungsschritten entsprechen. Nach 1000 Zyklen wird die Berechnung abgebrochen, falls bis dahin kein Ergebnis vorliegt. Anhand zufällig generierter, aber nicht überbeschränkter Beispielinstanzen für das Graphfärbbarkeitsproblem (s. Kap. 2.5.2) wurden die aufgeführten Algorithmen getestet, wobei n der Anzahl der Knoten in dem einzufärbenden Graphen entspricht. Die Anzahl der Kanten im Graph ist festgelegt auf $m = 2n$ und es stehen $k = 3$ Farben zur Verfügung. Die Quote (ratio) bezeichnet den Anteil gelöster Instanzen für die angegebene Problemgröße. Für 60, 90 und 120 Knoten wurden je zehn verschiedene Probleme generiert und je zehn unterschiedliche Initialbelegungen gewählt, sodass für jedes n pro Algorithmus 100 Testläufe durchgeführt wurden.

Es wird deutlich, dass die asynchrone Suche mit schwachen Zusagen weitaus effizienter und zuverlässiger arbeitet als das asynchrone Backtracking, was vornehmlich auf die flexible Rangordnung der Agenten zurückzuführen ist.

3.5 Verteiltes Breakout

Wie die iterative Verbesserung für lokale CSPs (beschrieben in Kap. 2.7) versucht auch der verteilte Breakout-Algorithmus [Yok00, S. 81ff.], Schritt für Schritt die aktuelle Wertebelegung zu verbessern. Um die potenzielle Parallelität auszunutzen, muss jedoch die globale Überprüfung auf lokale Minima entfallen.

Das verteilte Breakout besteht im Wesentlichen daraus, Vorschläge für den eigenen Wert

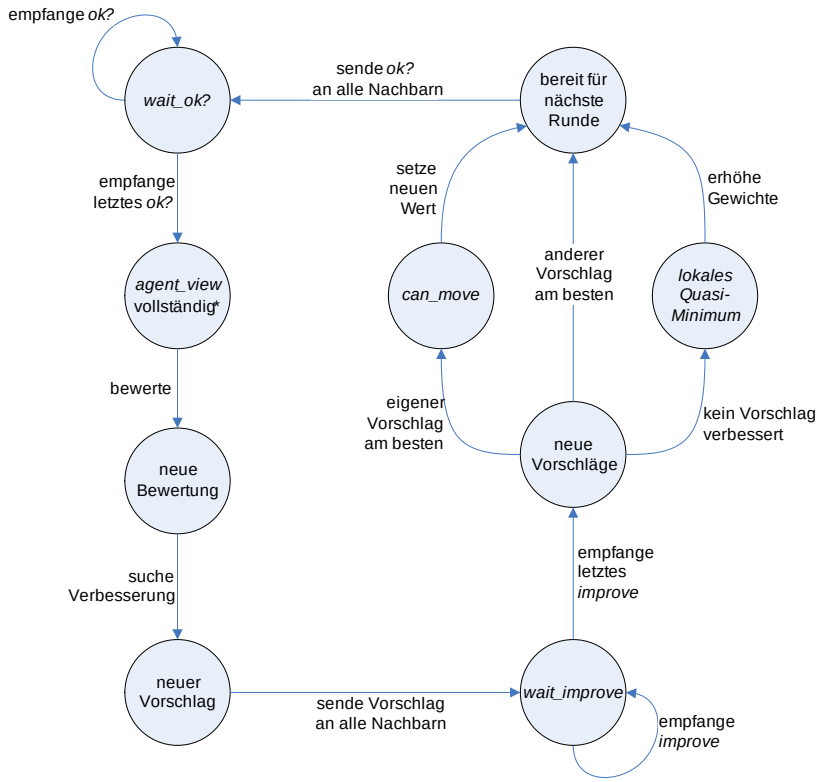

Abbildung 3.3: Verteiltes Breakout als Automat

zu versenden und diejenige Wertänderung unter allen aktuellen Kandidaten anzunehmen, die am meisten zur Lösung des Problems beiträgt (d. h. die Anzahl verletzter Constraints maximal verringert). Auch in der verteilten Variante wird von einer fehlerbehafteten Initialbelegung der Variablen ausgegangen, die durch geeignete Änderungen der Belegung verbessert werden soll.

Jeder Agent verwaltet eine Liste seiner Nachbarn (s. Def. 3.3). Zusätzlich sei die Distanz zwischen zwei Agenten distance(x_i, x_j) definiert als die Anzahl im Constraintnetz besuchter Knoten auf dem kürzesten Weg zwischen x_i und x_j. Agenten, die nicht benachbart sind, können simultane Wertänderungen vornehmen.

Wie beim asynchronen Backtracking werden die eigenen Werte mittels *ok?*-Nachrichten versandt, hinzu kommen nach dem Einsammeln der Werte aller Nachbarn die *improve*-Nachrichten. Diese enthalten einen neuen eigenen Wert, zusammen mit u. a. der dadurch möglichen Verbesserung (Anzahl zusätzlich erfüllter Constraints).[2]

[2]Die *improve*-Nachrichten enthalten noch weitere Informationen, auf die an dieser Stelle jedoch nicht näher eingegangen werden soll. Einer der mitgesendeten Werte stellt die Terminierung sicher, wenn eine Lösung gefunden wurde.

Da es (wie eingangs erwähnt) nicht möglich ist, ein lokales Minimum festzustellen, wird als Hilfskonstrukt das *lokale Quasi-Minimum* (*quasi-local-minimum*) eingeführt. Dieses besteht, wenn x_i ein Constraint verletzt und alle möglichen Verbesserungen durch x_i bzw. alle $x_j \in \text{neighbours}(x_i)$ gleich Null ist. Ein Teil des Constraintnetzes befindet sich demnach in einer Sackgasse. Liegt ein echtes lokales Minimum vor, ist auch das Vorhandensein eines lokalen Quasi-Minimums gegeben – die Umkehrung gilt entsprechend nicht, da nicht das gesamte Problem beobachtet wird. Eine mögliche Erweiterung des Algorithmus (broadcasting) sieht vor, dass Nachrichten nicht nur an Nachbarn, sondern an alle anderen Agenten verschickt werden.

Abb. 3.3 stellt den verteilten Breakout-Algorithmus in einer automatenähnlichen Form dar. Zentral sind dabei die beiden Zustände *wait_ok?* und *wait_improve*. In diesen wartet ein Agent jeweils alle Nachrichten des entsprechenden Typs ab, bevor er diese untersucht.

Sind alle *ok?*-Nachrichten eingetroffen, ist die Sicht auf die Werte der Nachbarn (*agent_view*) vollständig[3]. Diese momentane Belegung um x_i wird nun bewertet (in der Regel gemäß der Anzahl erfüllter oder verletzter Constraints). Danach sendet der Agent einen Verbesserungsvorschlag an alle seine Nachbarn und wartet die entsprechenden Nachrichten der anderen Agenten in *wait_improve* ab.

Nach dem Empfang der letzten derartigen Nachricht werden die gesammelten Vorschläge untersucht (Zustand „neue Vorschläge“). In dem Fall, dass ein Nachbar den besten Vorschlag gemacht hat, sendet der Agent seinen eigenen Wert erneut an die anderen Teilnehmer und beginnt die nächste Runde. Ist die Verbesserung mit dem eigenen Vorschlag am größten, so setzt der Agent seinen Wert auf den in der *improve*-Nachricht angegebenen neuen Wert und verkündet diesen den anderen Agenten. Schließlich bleibt noch das Szenario des lokalen Quasi-Mimimums: Hier werden die Gewichte für die am aktuellen Minimum beteiligten Constraints erhöht. Nachdem *ok?*-Nachrichten abgesetzt wurden, begibt sich der Agent zurück in den Zustand *wait_ok?*.

Empfängt ein Agent beispielsweise eine *improve*-Nachricht, ohne sich im Zustand *wait_improve* zu befinden, wird diese aufbewahrt und zu gegebener Zeit verarbeitet.

Der verteilte Breakout-Algorithmus kann seine Stärken vor allem bei sehr schwierigen Probleminstanzen in der Phasenübergangsregion ausspielen. Durch die höhere Parallelität verglichen mit der asynchronen Suche mit schwachen Zusagen ist der entstehende Kommunikationsoverhead vernachlässigbar. In einfachen Instanzen überzeugt jedoch eher die asynchrone Suche, die dem verteilten Breakout zudem die totale Korrektheit voraus hat.

[3] Vollständig bedeutet hier: für alle $x \in \text{neighbours}(x_i)$.

3.6 Erweiterungen

Die hier vorgestellten Algorithmen lassen sich um zwei interessante Aspekte erweitern bzw. entsprechend abändern.

Zum einen kann es für den Fall eines überbeschränkten DCSPs sinnvoll sein, eine partielles, verteiltes Constraintproblem zu lösen. Hierfür einigen sich die Agenten auf eine Lösung für ein relaxiertes Problem, wobei sie die angegebene Constraint-Wichtigkeit berücksichtigen.

Eine zweite Erweiterung lässt einen Agenten mehr als eine Variable handhaben. Dazu sind unter anderem folgende Ansätze denkbar, wenn auch nur sehr bedingt geeignet:

- Die Agenten versuchen, alle Lösungen für ihre lokalen Probleme zu finden und repräsentieren diese nach außen wiederum durch eine einzige Variable, wobei die Domain für diese Variable gerade die gefundenen Lösungen enthält. Dieser Ansatz ist offensichtlich ungeeignet, falls die lokalen Probleme komplex werden.
- Es werden virtuelle, lokale Agenten eingeführt, um die Variablen transparent zugänglich zu machen. Dadurch entsteht unter Umständen jedoch enormer Kommunikationsoverhead.
- Die Agenten erhalten wiederum Prioritäten und versuchen, eine mit den Lösungen höherpriorisierter Agenten verträgliche, lokale Lösung zu konstruieren. Gehört die Teillösung des höherrangigen Agenten jedoch nicht zu einer globalen Lösung, ist eine erschöpfende Suche auf den unteren Ebenen notwendig.

Konkret stellt Yokoo [Yok00, S. 103ff.] eine effiziente Erweiterung der asynchronen Suche mit schwachen Zusagen auf mehreren lokalen Variablen vor. Testergebnisse zeigen unter anderem, dass dieser aufgestockte Algorithmus ohne Schwierigkeiten Probleminstanzen behandeln kann, bei denen ein Agent 20 Variablen bearbeitet.

4 Constraintprobleme im Kontext der Sicherheit

Nachdem in den vorherigen Kapiteln die Grundlagen von CSPs erläutert wurden, sollen nun Ansätze vorgestellt werden, die sich auf CSPs stützen, um sicherheitsrelevante Probleme zu lösen. Daneben wird ein Algorithmus zur Behandlung von CSPs vorgestellt, mit dessen Hilfe beliebige, klassische CSPs (mit harten Constraints) unter Berücksichtigung von Sicherheitsaspekten gelöst werden können. Schwerpunkt dieses verteilten Algorithmus ist die Einhaltung von Vertraulichkeitsanforderungen bei der Lösung des Problems.

4.1 Terminvereinbarung mit Wahrung der Privatsphäre

In einer verteilten Umgebung sollen verschiedene Agenten mit individuellen Terminkalendern einen geeigneten Zeitpunkt und Ort für ein gemeinsames Treffen finden. Da jedoch nicht jeder Agent seinen bestehenden Kalender offenlegen möchte, wird nach einem Verfahren gesucht, das es gestattet, in angemessener Zeit einen solchen Termin zu vereinbaren, ohne die Privatsphäre der teilnehmenden Agenten über die Maßen zu beeinträchtigen.

In diesem Abschnitt beschreiben wir einen Ansatz, der in den Arbeiten von Freuder, Minca und Wallace [FMW01], Wallace und Freuder [WF02] und Wallace, Freuder und Minca [WFM04] dargelegt wird.[1]

4.1.1 Formulierung als CSP

Freuder, Minca und Wallace [FMW01] beschreiben ein Framework, das mit Hilfe von CSPs die Aushandlung eines derartigen Kompromisses ermöglicht. Jeder Agent bearbeitet dabei sein eigenes CSP, um eine konsistente Belegung für seinen Terminplan zu finden. Zusätzlich gibt es die globale Einschränkung, dass alle Agenten letztlich zu einem gemeinsamen Termin finden müssen, also alle individuellen CSPs insgesamt eine gewisse Konsistenzeigenschaft erfüllen sollen.

Im Detail soll das wie folgt charakterisierte Problem gelöst werden.

[1] In den zitierten Arbeiten wird dieses Szenario als *distributed meeting scheduling with privacy/efficiency tradeoffs* bezeichnet.

– Zur Eingabe des Problems gehört eine Menge von Zeitfenstern (time slots) Z. Ein Zeitfenster $z_i \in Z$ bezeichnet dabei die Startzeit eines Termins, die in den genannten Arbeiten den vollen Stunden zwischen 9 und 18 Uhr einschließlich entsprechen. Die Dauer eines Treffens beträgt stets eine Stunde. Betrachtet wird jeweils eine vollständige Woche (sieben Tage) mit den erwähnten zehn Zeitfenstern pro Tag. Insgesamt stehen damit 70 Zeitfenster zur Verfügung.

– Ebenfalls muss angegeben werden, welche Orte $o_j \in O$ für die Treffen zur Verfügung stehen und wie weit diese zeitlich voneinander entfernt sind. Die o_j werden dazu als Knoten eines Constraintnetzwerks (siehe Abb. 4.1) aufgefasst.

– Ein Termin oder ein Terminvorschlag wird durch ein Tupel (z_i, o_j) repräsentiert und steht für ein Treffen am Ort o_j im Zeitfenster z_i.

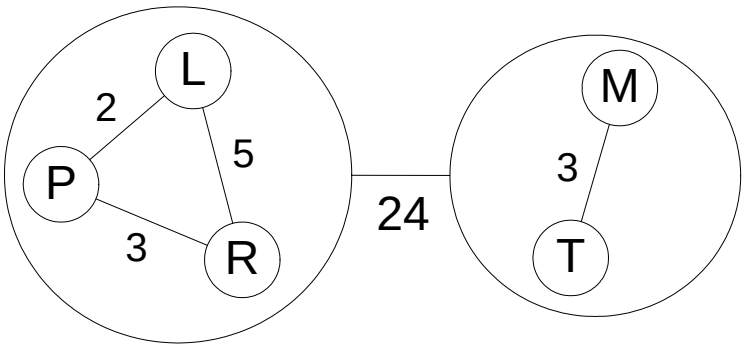

Abbildung 4.1: Constraint-Graph für die Reisezeitconstraints

Die Orte im Constraintgraph sind **L**ondon, **P**aris, **R**om, **M**oskau und **T**bilisi. Die Kante mit der Beschriftung „24“ bedeutet, dass die Reisezeit von jedem Ort der Menge $\{L, P, R\}$ zu einem beliebigen Ort der Menge $\{M, T\}$ (und umgekehrt) gerade 24 Stunden beträgt. Alle übrigen Kanten geben die Reisezeit zwischen den jeweiligen Knoten in Stunden an.

Bei der Suche nach einem geeigneten Termin schlagen die Agenten reihum (round-robin) einen für sie passenden Termin vor und warten die Antworten der anderen Agenten ab. Kann einer der angesprochenen Agenten zum vorgeschlagenen Termin nicht an einem Treffen teilnehmen, so unterbreitet der nächste Agent einen weiteren Vorschlag.

Ein Agent (a_1 in Abb. 4.2) sendet einen mit seinem eigenen Terminplan verträglichen Vorschlag (z_i, o_j) für ein Treffen an alle anderen Agenten. Diese antworten dann wie folgt:

OK falls der Vorschlag keine Konflikte mit dem eigenen Terminplan verursacht.

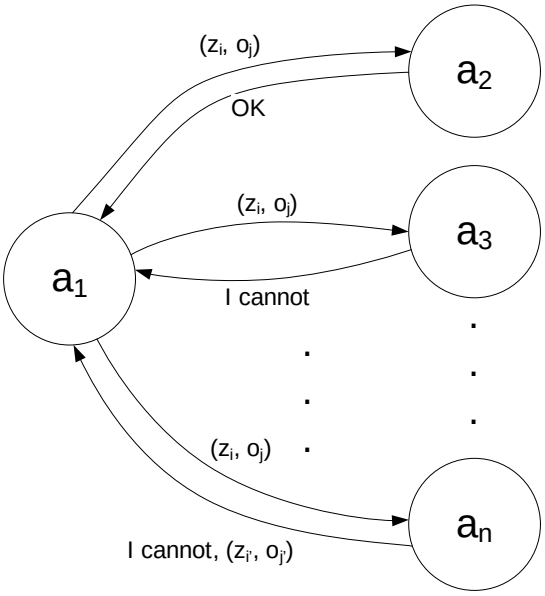

Abbildung 4.2: Ablaufskizze für distributed meeting scheduling

I cannot falls der Vorschlag nicht verträglich mit dem eigenen Terminplan ist, etwa weil dieses Zeitfenster bereits belegt ist oder der vorgeschlagene Termin nicht wahrgenommen werden kann, weil der Agent zu dieser Zeit zwischen zwei Orten unterwegs ist.

I cannot, $(z_{i'}, o_{j'})$ im Wesentlichen wie der Fall *I cannot*, jedoch wird hier zusätzlich ein den Konflikt verursachender Termin aus dem eigenen Kalender mitgesendet. Die Reisezeiten werden dabei berücksichtigt. Der angesprochene Agent befindet sich also im Zeitfenster $z_{i'}$ an Ort $o_{j'}$ und kann deshalb den Termin nicht wahrnehmen. Hier kann auch eine Menge von Terminen mitgesendet werden. Diese Mehrinformation wirkt sich (im Testfeld) jedoch nicht wesentlich auf die Effizienz des Verfahrens aus, wodurch der zusätzliche Verlust der Privatsphäre nicht gerechtfertigt ist.

An dieser Stelle sei darauf hingewiesen, dass die Agenten stets ehrlich antworten. Anders gesagt: Wenn der Terminplan ein Treffen zulässt, antwortet der Agent mit *OK*.

Der Agent a_1 wertet nun die eingehenden Antworten aus.[2] Dabei markiert dieser den Vorschlag im einfachsten Fall lediglich als unpassend, um in der nächsten Runde einen anderen Termin vorzuschlagen. Alternativ kann a_1 unter Umständen Schlussfolgerungen ziehen, wenn er Konflikte verursachende Termine mitgeteilt bekommt und dadurch weitere Kandidaten für Terminvorschläge ausschließen, die nicht für einen Termin bei einem

[2] Um den Ablauf möglichst anschaulich darzustellen, beschränken wir uns auf a_1. Tatsächlich kann natürlich jeder Agent in der Rolle des Vorschlagenden auftreten.

anderen Agenten in Frage kommen.

Für die Definition der CSPs der einzelnen Agenten benötigen wir zunächst noch eine Hilfsfunktion travel, die verwendet wird, um das Reisezeitconstraint einzuhalten:

$$\text{travel}((z_i, o_j), (z_{i'}, o_{j'})) = \begin{cases} true & \text{falls gemäß Reisezeitconstraints die Reise von } o_j \text{ nach } o_{j'} \text{ in der Zeit } |z_{i'} - z_i - 1| \text{ möglich} \\ true & \text{falls } o_j \text{ oder } o_{j'} \text{ undefiniert} \\ false & \text{sonst} \end{cases}$$

Damit lässt sich das CSP wie folgt charakterisieren:

- Die Zeitfenster bilden die Variablen. Jeder Agent verwaltet in seinem Problem demnach (Anzahl Tage) × (Anzahl Zeitslots pro Tag) Variablen – 70 Variablen im betrachteten Szenario. Damit entspricht die Variablenmenge X gerade der oben definierten Menge Z der Zeitfenster. Der Wert einer Variablen ist undefiniert, solange noch kein Termin für dieses Zeitfenster feststeht.
- Die Domains sehen für alle Agenten und Variablen gleich aus. Sie entsprechen den für die Treffen möglichen Orte: $D = \{L, P, R, M, T\}$. Dadurch lassen sich die Termine der Form (z_i, o_j) auf die Variablen abbilden: $[\![z_i]\!] = o_j$.
- Als Constraint betrachten wir

  $$\forall\ z, z' \in X \text{ mit } z \neq z'. \quad travel((z, [\![z]\!]), (z', [\![z']\!])) = true$$

  Dadurch ist gewährleistet, dass keine zwei Termine in den Kalender eingetragen werden, die für den Agenten zu dicht aufeinander folgen, um den entsprechenden Reiseweg zurückzulegen.

Da in dem beschriebenen Szenario nur die verhandelnden Agenten untereinander Termine vereinbaren können – sie also insbesondere keine Termine mit Personen haben können, die *nicht* zu den verhandelnden Agenten gehören – ist die globale Konsistenzeigenschaft einfach zu beschreiben: Es muss einen Termin geben, der in allen Kalendern vorkommt.

Wie bereits im einführenden Kapitel bemerkt, wird ein CSP mit zunehmender Anzahl von Constraints nur bedingt schwerer lösbar. Gerade sehr stark beschränkte Probleme haben oft nur sehr wenige Lösungen und durch die große Anzahl verbotener Werte lassen sich diese in der Regel auch in kurzer Zeit finden.[3] Obwohl in der oben gegebenen

[3] Im Allgemeinen gilt diese Folgerung nicht, denn die Problemklasse FewP [All86] enthält gerade die Probleme mit wenigen Lösungen. Dadurch, dass jedoch beim Lösen von Constraintproblemen mittels Heuristiken gewisse Teile des Suchbaums gar nicht betrachtet werden, ist die Folgerung in diesem Fall berechtigt.

Definition für das Terminvereinbarungsproblem nur ein Constraint aufgeführt ist, steigt die tatsächliche Beschränkung mit der Anzahl der im Kalender vermerkten Termine. Daraus lässt sich ableiten, dass das Problem leichter zu lösen ist, wenn die Anzahl initialer Meetings wächst. Diese Folgerung ist konsistent mit der empirischen Betrachtung, auf die im nächsten Abschnitt eingegangen wird. Allerdings sollte erwähnt werden, dass in den durchgeführten Experimenten stets eine Lösung garantiert ist, im Allgemeinen jedoch durchaus der Fall eintreten kann, dass keine Lösung existiert.

4.1.2 Vorteile durch Inferenzen

Wie bereits erwähnt, steht im Vordergrund, dass eine effiziente Lösung des Problems unter Berücksichtigung der Privatsphäre der Agenten bzw. der Geheimhaltung ihrer Kalender erzielt werden soll. Um nun die eigentliche Frage dieses Ansatzes zu beantworten – nämlich inwieweit sich die Privatsphäre der Agenten und eine möglichst schnelle Terminvereinbarung gegenseitig ausschließen – müssen wir uns zunächst ansehen, wie sich eingebüßte Privatsphäre und Effizienz messen lassen.

Die Autoren schlagen vor, die Effizienz abhängig von der Anzahl der unterbreiteten Terminvorschläge aller Agenten zu betrachten. Dies entspricht den oben vorgestellten Runden (Vorschlag und Antworten). Der Effizienzwert steigt, je *weniger* Vorschläge unterbreitet werden müssen. Demnach ist die Effizienz umgekehrt proportional zur Anzahl der Runden. Lässt sich also durch eine Veränderung des Antwortverhaltens eine geringere Rundenzahl bewirken, so arbeitet das Verfahren effizienter. Was die Privatsphäre anbelangt, unterscheiden wir zwei Typen von Antworten: zum einen das einfache Akzeptieren bzw. Ablehnen und zum anderen Antworten mit einer Menge von Konflikt-Terminen. Im ersten Fall zählt eine solche Antwort als eine eingebüßte Privatsphären-Einheit. Sind die Antworten ausführlicher, so zählen einerseits die direkt mitgeteilten Termine, zusätzlich aber auch die mittels Inferenzen sicher ausgeschlossenen Termine.

Die von den Agenten benutzten Inferenzen sind sehr einfach gehalten und schließen lediglich unwahrnehmbare Termine aus. Für einen Agenten a, der auf einen Vorschlag mit „I cannot, $((\text{Mo., 12 Uhr}), B)$“ antwortet, kommen am Montag um 10 Uhr, 11 Uhr, 13 Uhr und 14 Uhr lediglich Termine in Berlin in Frage, da er zu den genannten Uhrzeiten keinesfalls in einer anderen Stadt sein kann (nach London sind es drei und bis Paris fünf Stunden, die Orte in Osteuropa sind sogar einen ganzen Tag entfernt). Es lassen sich somit für a die oben genannten Zeitfenster für alle $o \in O \setminus \{B\}$ ausschließen – insgesamt kommen dadurch 16 Termine nicht mehr für ein Treffen in Frage. Andererseits sind die isolierten vier Zeitfenster in Berlin möglicherweise sehr gut geeignet für den nächsten Vorschlag. Dieser Effizienzgewinn hat jedoch seinen Preis, denn jeder auf diese Weise ausgeschlossene Termin zählt ebenfalls als eine eingebüßte Privatsphären-Einheit für den Agenten a.

Im empirischen Teil ihrer Arbeit [FMW01] zeigen die Autoren folgende Zusammenhänge auf:

Nur Ja-/Nein-Antworten Der Verlust an Privatsphäre entspricht hier gerade der Anzahl eliminierter Meetings, was gleichbedeutend mit der Anzahl von Nein-Antworten ist. Je mehr Termine zu Beginn schon in den Kalendern der Agenten zu finden sind, desto mehr Verhandlungsrunden werden auch geführt. Der Nutzen dieser einfachen Variante besteht hauptsächlich in der Möglichkeit, die anderen Varianten mit dieser vergleichen zu können und dadurch mögliche Verbesserungen oder Verschlechterungen zu erkennen.

Mitteilung eines Konflikts bzw. aller Konflikte In diesem Fall wird dem Anfrager ein Konflikt-Termin (bzw. alle Konflikt-Termine) mitgeteilt. Die Preisgabe von mehr Informationen schlägt sich jedoch nur sehr moderat in einem Effizienzgewinn nieder und das Mitteilen aller Konflikte bringt nur bei vielen initialen Terminen einen kleinen Vorteil gegenüber der Ein-Konflikt-Variante. Demgegenüber stehen hohe Einbußen bei der Privatsphäre, wobei der Alle-Konflikte-Fall erwartungsgemäß noch etwas schlechter abschneidet. Von den initial in den Kalendern befindlichen Meetings wurden insgesamt etwa 27% (Ein-Konflikt-Fall) bzw. 67% (Alle-Konflikte-Fall) identifiziert.

Mitteilen mit Inferenzen Zusätzlich zur Mitteilung etwaiger Konflikte versuchen ein oder mehrere Agenten in dieser Variante die oben beschriebenen Schlussfolgerungen zu ziehen. Auf diese Weise lassen sich zwischen fünf und zehn mal so viele Kandidaten, die nicht für einen gemeinsamen Termin in Frage kommen, eliminieren. Der Anteil identifizierter Termine (in den Kalendern der jeweils anderen Agenten) sinkt dadurch im Mittel um etwa 9% (Ein-Konflikt-Fall) bzw. 25% (Alle-Konflikte-Fall). Dies ist auf die gesunkene Rundenzahl zurückzuführen.

Vergleicht man das zweite und dritte Szenario, so ist zu beobachten, dass ein Effizienzzuwachs auch eine bessere Wahrung der Privatsphäre bewirken kann.

4.1.3 Agendas und Schließen aus Möglichkeiten

In einer weitergehenden Arbeit [WF02] wird zunächst das Maß für die eingebüßte Privatsphäre revidiert. Dabei liegt die Idee zugrunde, dass die Einbuße umso größer ist, je höher der Anteil der bekannt gewordenen Informationen an der Gesamtmenge von Informationen ist. Anders gesagt: 49 von 50 Terminen zu identifizieren ist gravierender als 49 von 490 Terminen zu ermitteln. Desweiteren werden einige mögliche Angriffe konkretisiert, falls einer der Agenten gar nicht an einem Termin interessiert ist, sondern lediglich etwas über einen oder mehrere andere Agenten in Erfahrung bringen möchte. Ein Angreifer verfügt im skizzierten Szenario über eine Agenda, die eine der folgenden Fragen beinhalten könnte:

- Ist Agent a_4 am Mittwochnachmittag (zwischen 12 und 18 Uhr) in Rom?

– Ist Agent a_2 zu einem bestimmten Zeitpunkt definitiv nicht in Berlin?

– Kann ich die Agenten a_3 und a_7 treffen, ohne dass Agent a_2 dem beiwohnen kann?

– Treffen sich die Agenten a_1 und a_2?

Ein Angreifer versucht also, anhand der gegebenen Antworten Schlussfolgerungen zu ziehen, die ihm letztendlich mehr über den Terminkalender eines anderen Agenten verraten als dieser Agent ausdrücklich bekannt gegeben hat. Dabei ist stets zu beachten, dass auch ein solcher Teilnehmer die Regel befolgen muss, auf eine Frage stets die Wahrheit zu sagen. Unklar ist jedoch, ob ein Agent beispielsweise einen Kalender verwendet, der nicht seine tatsächlichen Termine enthält.

Im Zuge dieses erweiterten Verfahrens wird nun versucht, eine Menge ursprünglich relevanter Möglichkeiten (original-relevant-set) auf eine Menge „wahrscheinlicher“ Möglichkeiten (resultant-set) zu reduzieren. Dabei sind jedoch keine Wahrscheinlichkeiten im strengen mathematischen Sinn gemeint. Zu Beginn erscheinen aus Sicht eines Agenten zunächst alle Belegungen der Kalender der anderen Agenten möglich, da noch nichts über die Kalender oder Termine der anderen Agenten bekannt ist. Im Verlauf des Verfahrens machen die Agenten jedoch Aussagen, die gewisse Termine wahrscheinlich machen und andere Termine mit Sicherheit ausschließen.

Die für eine bestimmte Agenda relevanten Fakten befinden sich wiederum in einer Menge (effective-set), die dem Agenten jedoch nicht bekannt ist. Relevante Fakten sind solche Informationen, die einen gewissen Rückschluss auf eine zu gewinnende Information aus der Agenda zulassen. Die Menge effective-set enthält für die Suche nach einem bereits im Kalender befindlichen Termin gerade das entsprechende Tupel aus Zeitfenster und Ort. Diese Überlegungen fließen auch in die geänderte Definition[4] der eingebüßten Privatsphäre ein ($||M||$ bezeichnet dabei die Kardinalität der Menge M):

$$\text{Einbuße} \;=\; ||\text{original-relevant-set}|| - ||\text{resultant-set}||$$

Für einen Angreifer ist erstrebenswert, eine Menge resultant-set zu produzieren, die gerade die Elemente aus effective-set enthält. In diesem Fall hat er alle für die Agenda relevanten Informationen ableiten können und der Privatsphären-Verlust ist damit maximal. Für alle Agenten, die die eigentliche Terminfindung zum Ziel haben, stehen stets zwei Agendas fest: Meetings anderer Agenten zu identifizieren und herauszufinden, wann diese Zeit für ein gemeinsames Treffen haben.

Um eine derartige Agenda zu verfolgen, hält ein Agent dafür sogenannte Schatten-CSPs (shadow CSPs) neben dem eigenen, mit deren Hilfe er versucht, die Terminkalender der anderen Agenten zu simulieren. Die Schatten-CSPs haben keine reguläre Lösung und

[4] Die Autoren verwenden eine $\log_2$-Betrachtung der Kardinalitäten, um Informationen und damit auch die eingebüßte Privatsphäre in Bits messen zu können.

die möglichen Werte aus den Domains haben Möglichkeits-Charakter. Genauer gesagt, werden für einen anderen Agenten Schatten-CSPs verwaltet, die mögliche *existierende* Termine (possible-has-meeting) und mögliche Termine für ein *gemeinsames Treffen* (possible-can-meet) enthalten. Prinzipiell werden noch mögliche Ursachen für Absagen verwaltet, also solche Termine, die den anderen Agenten zur Absage bewogen haben (possible-causes). Das Ziel der Bearbeitung dieser Schatten-CSPs ist es, von den entstehenden Möglichkeiten letztendlich auf sichere Fakten zu schließen. Dieses Verfahren bezeichnen die Autoren als *possibilistic reasoning* [WFM04].

4.2 Analyse von Sicherheitsprotokollen mittels weicher Constraints

Das von Bistarelli vorgeschlagene Framework für die Bearbeitung weicher Constraints in den Soft-Constraint-Satisfaction-Problemen (SCSPs) (siehe auch Kap. 2.10) eignet sich auch zur Behandlung von sicherheitsrelevanten Fragestellungen. Ein Ansatz beschäftigt sich mit der Verifikation von Sicherheitsprotokollen hinsichtlich Vertraulichkeits- und Authentizitätsanforderungen.

Für ein Sicherheitsprotokoll – etwa das Needham-Schroeder- oder das Kerberos-Protokoll – soll untersucht werden, inwieweit Informationen nur denjenigen Teilnehmern zugänglich sind, für die sie auch bestimmt sind. Details des begutachteten Protokolls finden sich im Anhang A. Das hier vorgestellte Verfahren ist in [BB05] skizziert und detailliert ausgeführt in [BB01] und [BB04]. Grundlegende Motivation für die Soft-Constraint-Betrachtung ist nach Bella und Bistarelli vor allem, dass übliche Analysemethoden nur Boolesche Antworten geben: Ein Protokoll wahrt Vertraulichkeit oder eben nicht. Mit dem vorgeschlagenen Analyseframework auf Basis der weichen Constraints wird eine differenziertere Betrachtung möglich.

4.2.1 Grundlagen für die SCSPs

Wir betrachten eine Menge von Sicherheits-Labels

$$L = \{public, traded_k, \ldots, traded_2, traded_1, private, unknown\}$$

mit der Ordnung

$$public \leq_L traded_k \leq_L \ldots \leq_L traded_2 \leq_L traded_1 \leq_L private \leq_L unknown$$

sowie zwei Operationen auf dem Verband $(L, \leq_L)$:

- $+_{sec}$: entspricht der Supremumsoperation $\sqcup$

– $\times_{sec}$: entspricht der Infimumsoperation $\sqcap$

public bedeutet dabei, dass diese Information öffentlich zugänglich ist, *private* hingegen, dass nur ein bestimmter Agent, für den die Information mit *private* markiert ist, diese kennt. Eine mit *unknown* markierte Nachricht ist dem jeweiligen Agenten nicht bekannt. Die Zwischenstufen $traded_i$ symbolisieren den Nachrichtenversand über das Netzwerk. Schickt ein Teilnehmer eine Nachricht mit der Vertraulichkeitsmarkierung $traded_i$, so trifft diese in der Regel beim Empfänger mit einer schlechteren Vertraulichkeit $traded_j < traded_i$ ein. Um allgemeine Aussagen mittels *traded*-Markierungen zu treffen, werden folgende Doppelbenennungen verwendet:

$$\begin{aligned} unknown &= traded_{-1} \\ private &= traded_0 \\ public &= traded_{k+1} \end{aligned}$$

Im Folgenden betrachten wir den Constraint-Halbring $S_{sec} = (L, +_{sec}, \times_{sec}, public, unknown)$. Ein Netzwerk, in dem ein Protokoll untersucht werden soll, stellen wir durch das Constraintsystem $CS_n = (S_{sec}, D, V)$ dar, wobei V die Teilnehmer (Agenten) des Netzwerks beschreibt[5]. D enthält alle atomaren, abgeleiteten und zusammengesetzten Nachrichten, sowie eine leere Nachricht $\{\!|\}$. Mittels Konkatenierung und Verschlüsselung können aus ursprünglich atomaren Nachrichten neue, zusammengesetzte Nachrichten entstehen. Aufteilung und Entschlüsselung werden verwendet, um aus zusammengesetzten Nachrichten deren Komponenten zurückzugewinnen. Die genaue Darstellung dieser (De-)Kompositionsregeln soll an dieser Stelle jedoch ausbleiben. Diese Nachrichten spiegeln im Wesentlichen das Wissen der Agenten wider.

Ziel der Protokollanalyse ist es festzustellen, ob für bestimmte Agenten und Nachrichten eine Zusicherung bezüglich Vertraulichkeit bzw. Authentifizierung gegeben werden kann. Anders formuliert: Ist in einem bestimmten Szenario die durch eine Sicherheitspolitik vorgegebene l-Vertraulichkeit ($l \in L$) gewahrt? Oder für den Fall der Authentifizierung: Basiert die Authentifizierung eines Teilnehmers gegenüber einem anderen Teilnehmer auf einer l-vertraulichen Nachricht? Durch die Politik (in Form eines SCSP) werden also die Anforderungen an das Protokoll vorgegeben, die es dann in einem bestimmten Szenario zu überprüfen gilt.

Dazu werden noch einige Berechnungsregeln für die Sicherheitsmarkierungen festgelegt, die vorschreiben, welche Markierung eine Nachricht erhält, wenn sie aus bestehenden Nachrichten aufgebaut (Verschlüsselung und Konkatenierung) oder extrahiert (Entschlüsselung und Aufteilung) wird. Mit Hilfe dieser Regeln lässt sich später feststellen, ob ein Constraint – damit also ein Ausschnitt des Wissens eines Agenten – aus einem anderen Constraint (beispielsweise früheres Wissen) abgeleitet werden kann.

[5] Die notwendige Ordnung auf den Variablen ist erforderlich, aber für unsere Betrachtung nicht erheblich.

4.2.2 Das initiale SCSP

Für jeden Agenten $A \in V$ wird ein unäres Constraint aufgestellt, das das Wissen von A kategorisiert:[6]

- Alle Agentennamen (und Zeitstempel, falls das Protokoll diese verwendet) erhalten das Label *public* und sind somit für A (und alle anderen Agenten) sichtbar. Falls eine Nachricht für einen Agenten mit *public* markiert ist, ist sie auch für alle anderen Agenten so markiert.
- Alle initialen Geheimnisse wie symmetrische Schlüssel K_A, geheime Teile asymmetrischer Schlüsselpaare K_A^{-1}, sowie zu Beginn der Protokollausführung *bereits erzeugte* Nonces erhalten das Label *private*. Sie sind nur A selbst bekannt.
- Alle anderen Geheimnisse, sowie später im Verlauf einer Protokollsitzung erzeugte Nonces oder Schlüssel erhalten das Label *unknown*. Dies sind alle Elemente aus D, die nicht in die ersten beiden Kategorien fallen.

Die auf diese Weise für die einzelnen Agenten erzeugten Constraints bilden die Menge $C_i(CS_n)$. Damit erhalten wir das initiale SCSP $P_i = (C_i(CS_n), V)$ für ein Protokoll. Dies entspricht im Wesentlichen einem Netzwerk, in dem das Protokoll ausgeführt werden kann, aber noch keine Session stattgefunden hat.[7]

4.2.3 Das Politik-SCSP (policy SCSP)

Mit einem Sicherheitsprotokoll assoziieren die Autoren auch eine Sicherheitspolitik, die – sehr allgemein gesagt – vorgibt, welche Informationsflüsse gestattet sind. Dementsprechend wird auch ein SCSP entworfen, das die zu Grunde liegende Politik beschreibt. Es spezifiziert die zwischen Paaren von Agenten auszutauschenden Nachrichten im Verlauf einer Session. Dazu wird das initiale SCSP erweitert: die Menge von Constraints $C_i(CS_n)$ bezüglich des das Netzwerk repräsentierenden Constraintsystems wird wie nachfolgend beschrieben ergänzt.

Jeder Protokollschritt besteht aus dem Senden einer Nachricht von einem Teilnehmer A an einen Teilnehmer B und wird durch höchstens zwei binäre Constraints beschrieben:

(R1) Kreiert A ein neues Geheimnis n, um die Nachricht m zu konstruieren? Dann wird ein neues Constraint eingefügt, das n für A das Label *private* zuweist.

[6] Variablen repräsentieren hier die Teilnehmer und werden ausnahmsweise mit Großbuchstaben bezeichnet, Nachrichten mit Kleinbuchstaben.

[7] Der Index i steht hier nur abkürzend für „initial“ und stellt keinen Bezug zu einem bestimmten Agenten her.

(R2) Für jede versendete Nachricht m wird ein binäres Constraint zwischen A und B hinzugefügt, das dem Tupel $(\{\!\}, m)$ eine Markierung zuweist. Diese wird gegebenenfalls durch eine der Operation entsprechenden Regel berechnet oder setzt lediglich eine bestehende $traded_i$-Markierung von m für den Empfänger auf $traded_{i+1}$ herab.

Alle übrigen, neuen Tupel, die durch die Hinzunahme von n bzw. m zur Domain D entstehen, erhalten das Label *unknown*.[8] Das resultierende SCSP bezeichnen wir als Politik-SCSP $P_p = (C_p(CS_n), V)$ – es beschreibt ein Netzwerk, in dem beliebig viele Sessions des Protokolls stattfinden können, ohne dass ein Teilnehmer etwas Bösartiges unternimmt. Das Politik-SCSP spezifiziert also gerade die beabsichtigte Ausführung des Protokolls.

4.2.4 Das Zuschreibungs-SCSP (imputable SCSP)

Wo das Politik-SCSP das initiale SCSP um die gemäß der Protokollausführung erlaubten bzw. vorgesehenen Schritte erweitert, wird das Zuschreibungs-SCSP benötigt, um das tatsächliche Ausführungsverhalten des Protokolls zu erfassen. Insbesondere werden in dieser Variante des initialen SCSP abgefangene Nachrichten berücksichtigt. Eine Instanz des Zuschreibungs-SCSP beschreibt also den Zustand einer konkreten Ausführung (Session) des Protokolls. Diese Erweiterung des initialen SCSP verwendet wiederum die Regeln R1 und R2 (für das Politik-SCSP), sowie zwei weitere Regeln. R2 wird dabei nur dann angewandt, wenn die Nachricht ihren vorhergesehenen Empfänger auch tatsächlich erreicht.

(R3) Falls C eine Nachricht m von A an B abfängt, füge ein binäres Constraint zwischen A und C hinzu, das dem Tupel $(\{\!\}, m)$ eine Markierung zuweist, die ggf. die Markierung gemäß der Ordnung herabsetzt (falls sie noch nicht *public* ist). Alle anderen Tupel erhalten die Markierung *unknown*.

(R4) Falls C aus einer mitgehörten Nachricht m mittels kryptographischer Analyse die ursprüngliche Nachricht n rekonstruieren kann, erhält diese für C die Markierung *private*, alle anderen neuen Tupel wiederum *unknown*.

Das Zuschreibungs-SCSP hat damit die Form $P_z = (C_z(CS_n), V)$, wobei $C_z(CS_n)$ wie bereits beim Politik-SCSP durch Erweiterung der Constraintmenge des initialen SCSP $C_i(CS)$ gemäß der angegebenen Regeln entsteht.

[8] In den weichen Constraintproblemen werden für ein Constraint (def, con) allen möglichen Wertebelegungen (Tupeln) über der Variablenmenge *con* Markierungen zugeordnet. Siehe dazu auch Kapitel 2.10.

4.2.5 Angriffe auf Vertraulichkeit und Authentifizierung

Unter *l-Vertraulichkeit* verstehen die Autoren, dass es einem Angreifer in einem durch ein P_z beschriebenen Szenario – bezogen auf ein Politik-SCSP P_p – nicht gelingt, die Vertraulichkeitsstufe $l \in L$ einer Nachricht m herabzusetzen. Ist $l = unknown$, so besteht dafür die beste Vertraulichkeit und analog für $l = public$ die schlechteste.

Damit lässt sich nun ein Vertraulichkeitsangriff präzisieren: Gelingt es einem Teilnehmer, die Vertraulichkeitsstufe l bezüglich einer Nachricht m in einem P_z auf l' herabzusetzen – l' ist damit gemäß der Ordnung $(L, \leq_L)$ kleiner als l – dann besteht ein Vertraulichkeitsangriff in P_z bezogen auf die Politik P_p. Betrachtet man zwei Angriffe, so ist derjenige als gravierender zu betrachten, für den die resultierende Vertraulichkeitsstufe l' kleiner ist. Dies entspricht einem höheren Vertraulichkeitsverlust. Alternativ kann auch die Anzahl „verlorener“ Vertraulichkeitsstufen ein Kriterium sein.

l-Authentifizierung von B bei A in einem Zuschreibungs-SCSP P_z besteht genau dann, wenn es eine Nachricht m gibt, die sowohl A als auch B kennen (die also für beide eine Markierung $< unknown$ besitzt, für A jedoch nicht schlechter als mit l markiert ist. Zudem muss B in m „erwähnt“ werden. Das Original-Prädikat speaksabout(m, B) ist nicht genauer spezifiziert und für das entsprechende Protokoll geeignet zu definieren.

Die Autoren stellen exemplarisch Lowes Angriff auf das Needham-Schroeder-Protkoll dar (siehe auch Anhang A). Durch den Angriff erhält der böswillige Teilnehmer I, der wechselweise die Rolle von A und B einnimmt, für bestimmte Nachrichten „öffentlichere“ Vertraulichkeitsstufen als *unknown* und gelangt damit in den Besitz der beiden Nonces N_a und N_b, die für ihn laut Policy-SCSP *unknown*-vertraulich sein sollten, mit dem Angriff jedoch nur noch $traded_4$-vertraulich sind [BB04]. In [BB01] wird dabei nur auf die Vertaulichkeit eingegangen, [BB05] berücksichtigt zudem die Authentifizierung. In [BB04] ist außerdem noch ein Angriff auf das Kerberos-Protokoll mit Hilfe des vorgestellten Frameworks modelliert.

4.3 Kaskaden-Verwundbarkeit in Netzwerken mit mandatorischer Zugriffskontrolle

Das klassische Modell der mandatorischen Zugriffskontrolle geht zurück auf eine Arbeit von Bell und LaPadula [BL73]. In einem System mit derartiger Zugriffskontrolle gibt es Objekte, auf die von Subjekten, den Teilnehmern, zugegriffen wird. Objekte erhalten eine Klassifizierung. Üblicherweise folgen diese der totalen Ordnung:

$$\text{Open/Unclassified} < \text{Confidential/Classified} < \text{Secret} < \text{Top Secret}.$$

Analog erhalten die Subjekte eine Freigabe, die ebenfalls einer der genannten Stufen entspricht. Um zu verhindern, dass Informationen aus Objekten mit hoher Geheimhal-

tungsstufe Unbefugten zugänglich werden, wird der Informationsfluss bei jedem Zugriff kontrolliert. Lesezugriffe sind nur erlaubt, falls das zugegriffene Objekt eine geringere Klassifizierung hat als der zugreifende Teilnehmer. Dadurch fließt die Information entsprechend der Ordnung aufwärts. Auf der anderen Seite kann ein Teilnehmer nur in Objekte mit höherer Klassifizierung als seiner eigenen schreiben. Damit ist auch hier der Informationsfluss aufwärts gerichtet. Wenn ein Teilnehmer auf ein Objekt zugreift, dessen Klassifizierung genau seiner Freigabe entspricht, dann sind beide Zugriffe gestattet. Das Modell kann um weitere Stufen erweitert werden, wobei die lineare Ordnung eingehalten werden muss. Entsprechend der englischen Bezeichnung für diese Art der Zugriffskontrolle werden die Systeme häufig multi-level secure Systems genannt.[9]

Für ein MLS-System lassen sich bestimmte Eigenschaften zusichern, die für einige Zertifizierungen etwa gemäß der Common Criteria notwendig sind. Unglücklicherweise ist ein Netzwerk aus MLS-Systemen in der Regel nicht mehr sicher bezüglich der zugesicherten Informationsfluss-Eigenschaften für ein einzelnes MLS-System. Lässt sich die Struktur eines Netzwerks ausnutzen, um Informationsfluss entgegen der eigentlich durch die Systeme erzwungenen Richtung zu bewirken, sprechen wir von einer Kaskaden-Verwundbarkeit (*cascade vulnerability*).

Bistarelli et al. [BFO04] haben mit Hilfe weicher Constraints ein Analysewerkzeug vorgeschlagen, das solche Verwundbarkeiten erkennt und eine geeignete Umstrukturierung des Netzwerks vorschlägt. Letztere besteht aus dem Kappen von Verbindungen zwischen verschiedenen Systemen, wodurch jedoch möglichst wenige Einschränkungen entstehen sollen.

4.3.1 Modellierung des Netzwerks

Zunächst werden zwei totale Ordnungen definiert:

- $(L, \leq_L)$ enthält die Klassifizierungsstufen (*labels*) gemäß der oben vorgestellten Hierarchie. Die kleinste Stufe (in der Regel *Open*) wird zusätzlich mit $\mathbf{0}_L$ bezeichnet, die höchste Stufe (meist *Top Secret*) mit $\mathbf{1}_L$.
- $(A, \leq_A)$ beschreibt die Zusicherungsstufen (*assurance levels*) für die Sicherheit eines Systems s aus der Menge aller betrachteten Systeme S gemäß bestimmter Kriterien. Analog verwenden wir die Bezeichnungen $\mathbf{0}_A$ und $\mathbf{1}_A$. Für zwei solcher Stufen $x, y \in A$ mit $x \leq_A y$ gilt: y ist nicht weniger sicher als x und entsprechend: Wenn es einem Angreifer gelingt, ein System mit der Zusicherungsstufe y zu kompromittieren, dann kann dieser auch ein System mit der Zusicherungsstufe x brechen. Gemäß der Zusicherungsstufe eines Systems $s \in S$ liefert $\text{accred} : S \to A$

[9] Wir betrachten an dieser Stelle eine einfache, sehr restriktive Variante dieser Systeme. In erweiterten Modellen können besonders vertrauenswürdige Teilnehmer beispielsweise Objekte deklassifizieren bzw. deren Klassifizierung verringern. Dies ist in unserer Betrachtung nicht vorgesehen.

gerade die entsprechende Stufe, die den minimal nötigen Einbruchsaufwand beschreibt.

Ein realistisches Maß für die Sicherheit der klassifizierten Objekte ist der nötige Aufwand, um Informationen der Klassifizierung l auf $l' < l$ herabzusetzen. Dies nennen die Autoren *akzeptables Risiko*. Eine entsprechende Funktion risk : $L \times L \to A$ liefert den entsprechenden Aufwand in Form einer Zusicherungsstufe, die überwunden werden muss. Ist $l < l'$, findet erwünschter Informationsfluss in Richtung einer niedrigeren Klassifizierungsstufe statt, und risk liefert in diesem Fall stets $\mathbf{0}_A$.

Die bewerteten Systeme – für die eine Zusicherung vorliegt – müssen stets höher klassifiziert sein als das akzeptable Risiko einer Kompromittierung der dort abgelegten Daten. Formal: Für alle Tupel (l, l') von Datenklassifizierungen in s gelte daher: $\text{risk}(l, l') \leq_A \text{accred}(s)$.

Für die Modellierung des Netzwerks betrachten wir je eine Ebene eines MLS-Systems als einen Systemknoten (*system node*). Verarbeitet ein System a beispielsweise Informationen mit den Klassifizierungen Secret und Confidential, stellen wir es mit zwei Systemknoten S_a und C_a dar. Wir verwenden die Systemknoten als Knoten in einem Graph und repräsentieren die Verbindungen zwischen den Systemknoten (und damit Verbindungen sowohl innerhalb von als auch zwischen Systemen) durch Kanten. Kantengewichte an einer Kante zwischen l_s und $l'_{s'}$ stellen den benötigten Aufwand dar, um Informationen direkt von der Stufe l im System s auf die Stufe l' im System s' zu kopieren. Der Politik nach erlaubte Flüsse (aufwärts oder auf gleicher Klassifizierungsstufe) erhalten Gewicht 0, alle anderen Flüsse Gewicht $\text{accred}(s)$, wobei s das System ist, aus dem kopiert wird. Das Prädikat $\text{effort}(l_s, l'_{s'})$ liefert das höchste Kantengewicht entlang aller Pfade von l_s nach $l'_{s'}$.

Ein solches MLS-Netzwerk ist nun frei von Kaskaden-Verwundbarkeiten, wenn gilt:

$$\forall\, l, l' \in L,\ s, s' \in S. \quad \text{risk}(l, l') \leq_A \text{effort}(l_s, l'_{s'})$$

Damit gibt es keinen Pfad, der es ermöglicht, die fragliche Information mit geringerem Aufwand zu kopieren, als er für die direkte Herunterklassifizierung der Information nötig wäre. Die Topologie des Netzwerks stellt also kein größeres Risiko dar als die Aufbewahrung der Information in einem bestimmten System.

4.3.2 Formalisierung über weiche Constraintprobleme

Das Constraintsystem $CS_{casc} = (S_{casc}, D, V)$ verwendet den Constraint-Halbring $S_{casc} = (\mathbb{N}, min, max, \infty, 0)$.

Die $2n$ Variablen V sind je zwei *Systemknoten-Variablen* S_i^s und S_i^d (kurz $S_i^?$), die den

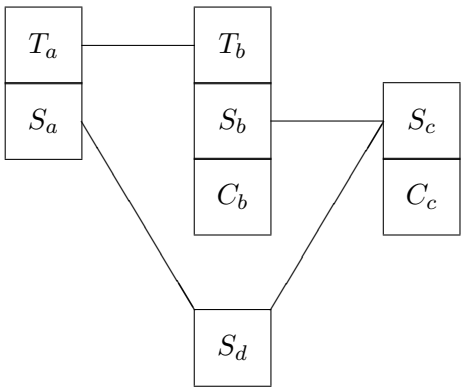

Abbildung 4.3: Beispiel für ein MLS-Netzwerk

i-ten Knoten eines Pfads durch das Netzwerk beschreiben. Das Fragezeichen steht stellvertretend für s oder d, was die Rolle des Systems im Pfad ausdrückt: sender (Absender) oder destination (Empfänger). Die Autoren unterscheiden *Flüsse* zwischen verschiedenen Klassifizierungsstufen innerhalb ein und desselben Systems und *Synchronisierungen*, d. h. Flüsse zwischen verschiedenen Systemen. Diese Unterscheidung bedingt, dass ein Pfad durch das Netzwerk gegeben ist durch eine Folge der Länge $2n$: $(S_1^s, S_1^d, S_2^s, S_2^d, \ldots, S_n^s, S_n^d)$. Knotenpaare S_i^s und S_i^d (mit $1 \leq i \leq n$) beschreiben Flüsse; Paare S_i^d und S_{i+1}^s (mit $1 \leq i < n$) beschreiben Synchronisierungen.

Die Domain D der Variablen ist gegeben durch eine Aufzählung aller im Netzwerk vorhandenen Systemknoten. Im Beispiel aus Abbildung 4.3 sind dies $D = \{T_a, S_a, T_b, S_b, C_b, S_c, C_c, S_d\}$. Die Domain wird zusätzlich um weitere Elemente ergänzt, falls Flüsse der Länge k mit $2 < k < n$ betrachtet werden. In diesem Fall werden die verbleibenden $n-k$ Systemknoten-Variablen mit künstlichen Systemknoten $*_1^s, *_1^d, \ldots, *_{n-k}^s, *_{n-k}^d$ belegt.

Für den Fluss innerhalb eines Systems wird unterschieden zwischen gemäß $(L, \leq_L)$ erlaubten Flüssen ($Flow_{permitted}$); nicht erlaubten, aber möglichen Flüssen, falls ein System kompromittiert wird ($Flow_{risk}$) und unmöglichen Flüssen ($Flow_{invalid}$). Entsprechend werden zwischen den Systemknoten-Variablen Constraints $C_{flow}(C_{casc})$ definiert:

$$c(S_i^s, S_i^d) = \begin{cases} accred(S_i) & ([\![S_i^s]\!], [\![S_i^d]\!]) \in Flow_{risk} \\ 0 & ([\![S_i^s]\!], [\![S_i^d]\!]) \in Flow_{permitted} \text{ oder} \\ & ([\![S_i^s]\!], [\![S_i^d]\!]) \in \{(*_1^s, *_1^d), \ldots, (*_{n-k}^s, *_{n-k}^d)\} \\ \infty & \text{sonst} \end{cases}$$

Zusätzlich gilt ein alldifferent-Constraint c_{unique} auf der Menge $\{S_i^s \mid S_i^s \in V\}$. Da zwei Variablen S_i^s und S_i^d jeweils das gleiche System beschreiben, ist mit diesem Constraint gewährleistet, dass Systemknoten-Variablen mit verschiedenen Indizes i auch tatsächlich verschiedene Systeme beschreiben. Ein analoges Constraint wird ggf. für die künstlichen Systemknoten aufgestellt.

Für Synchronisierungen (Flüsse zwischen verschiedenen Systemen) werden ebenfalls Constraints $C_{syn}(CS_{casc})$ aufgestellt, die lediglich die entsprechend der Netzwerk-Topologie bestehenden Verbindungen zwischen den Systemknoten formalisieren ($1 \leq i < n$):

$$c(S_i^d, S_{i+1}^s) = \begin{cases} 0 & \llbracket S_i^d \rrbracket \text{ und } \llbracket S_{i+1}^s \rrbracket \text{ im Netzwerk verbunden sind oder} \\ & (\llbracket S_i^d \rrbracket, \llbracket S_{i+1}^s \rrbracket) \in \{(*_1^s, *_1^d), \ldots, (*_{n-k}^s, *_{n-k}^d)\} \\ & \cup \{(\#, *_1^s) \mid \# \text{ nicht künstlich}\} \\ \infty & \text{sonst} \end{cases}$$

Damit haben wir ein Aufwands-SCSP $P_E = (C_E(CS_{casc}), V)$ mit $C_E = C_{flow} \cup C_{syn} \cup \{c_{unique}\}$ (*effort SCSP*) definiert, dessen Lösung alle möglichen Pfade durch das System liefert, wenn wir die Belegungen betrachten, deren Markierung kleiner als ∞ ist.

Um nun das Risko der Herunterklassifizierung vom Anfang bis zum Ende eines Pfads fassen zu können, definieren wir noch eine Menge von Risiko-Constraints $C_{risk} = \{r(S_1^s, S_i^d) \mid 2 \leq i \leq n\}$. Jedes einzelne dieser $r(S_1^s, S_i^d)$ gibt dabei das akzeptable Risiko zwischen der Klassifizierungsstufe in S_1^s und der in S_i^d an. Das Hilfsprädikat risk$'$ liefert dabei das akzeptable Risiko zwischen den Klassifizierungsstufen der gegebenen Systemknoten.

$$r(S_1^s, S_i^d) = \begin{cases} 0 & \llbracket S_i^d \rrbracket \text{ ist künstlicher Systemknoten} \\ \text{risk}'(\llbracket S_1^s \rrbracket, \llbracket S_i^d \rrbracket) & \textit{sonst} \end{cases}$$

Die Lösung des entsprechenden Risiko-SCSPs $P_R = (C_{risk}(CS_{casc}), V)$ liefert damit die akzeptablen Risiken.

Ein kaskadierender Pfad ist nun eine Belegung der Systemknotenvariablen, für die gilt: $\bigotimes C_{risk} > \bigotimes C_{effort}$. Damit ist eine direkte Herunterklassifizierung einer bestimmten Information aufwändiger als der durch die Belegung der Systemknoten-Variablen gewählte Weg durch das Netzwerk.

4.4 Workflow-Management mit dynamischer Teilnehmerbasis

Ein Workflow-Management-System (WFMS) wird verwendet, um einzelne Aufgaben (*tasks*) zu Arbeitsabläufen (*workflows*) zu strukturieren und zusammenzufassen. Allgemein gesprochen gibt es eine Menge von Tasks T und eine nicht zwingend lineare Abarbeitungsreihenfolge der Tasks. Ein sicheres WFMS sorgt nicht nur dafür, dass alle Tasks geeignete Nutzer zugewiesen bekommen, sondern berücksichtigt auch eventuelle Beschränkungen bei der Zuordnung von Nutzern zu Rollen, welche dann wiederum Tasks zugeordnet werden.

Das von Moodahi et al. [MGLM04, MGM05] vorgeschlagene, dynamische WFMS-Modell verwendet dabei ein rollenbasiertes Konzept, das in der entsprechenden Sicherheitspolitik keine direkte Zuordnung von Nutzern zu Tasks vorsieht, sondern lediglich bestimmt, welche Rollen für welche Tasks notwendig sind. Die dynamische Komponente dieses Ansatzes

besteht darin, dass im Gegensatz zu üblichen WFMS die Teilnehmerbasis nicht im Voraus bekannt ist (in etablierten Workflow-Systemen ist diese Zuordnung von Nutzern zu Rollen fest). Somit können prinzipiell jederzeit neue Nutzer hinzutreten und das System verlassen, wodurch die Idee auf offene Systeme wie etwa das Internet anwendbar wird. Allerdings erfordert dieser Umgang mit den Teilnehmern auch eine adäquate Überprüfung der entsprechenden Berechtigungen, die in den vorgestellten Papieren durch Credentials[10] realisiert ist. Eine Vielzahl von Beschränkungen begrenzt die tatsächlich möglichen Zuweisungen von Nutzern zu Rollen und im Endeffekt zu Tasks. Ferner berücksichtigt das Modell einen einfachen Kostenbegriff: Die von den Nutzern vorgelegten Zertifikate weisen auch die Kosten aus, die anfallen, wenn dieser Nutzer einen bestimmten Task ausführt.

Die wesentliche Neuerung gegenüber etablierten Workflow-Systemen ist die Zuteilung von Rollen an eine dynamische Nutzerbasis im Gegensatz zur bisher festen Zuordnung von Nutzern zu Rollen.

In der parallel zur vorliegenden entstandenen Diplomarbeit von Matthias Niggemeier [Nig06] wird dieses Workflow-Management-System ausführlich untersucht, weshalb an dieser Stelle auf eine detaillierte Beschreibung verzichtet werden soll.

4.4.1 Zentrale Komponenten und ihre CSPs

Das Modell verwendet drei Komponenten:

- Einen Credentials Collector (CC), bei dem Teilnehmer ihre Credentials vorlegen. Diese Komponente prüft auch ständig die Gültigkeit registrierter Credentials und meldet etwaige abgelaufene Berechtigungen. Zusätzlich kennt der CC auch die Kosten für einen Teilnehmer, die anfallen, wenn dieser einen Task bearbeitet.
- Einen Role Manager (RM), der versucht, eine kostenminimale Zuordnung von Nutzern zu Rollen in einer gegebenen Workflow-Instanz (im Folgenden *Session*) zu finden.
- Pro Workflow-Definition wird ein Task Manager (TM) benötigt, der für die enthaltenen Tasks jeder Session eine Zuordnung von Nutzern zu diesen Tasks vornimmt und dazu mit dem RM kommuniziert. Dies entspricht einer Betrachtung aus der Sicht des in Abb. 4.4 dargestellten *Systems*, das für eine Workflow-Instanz die Nutzer- / Rollen- / Task-Zuordnung benötigt.

Abb. 4.4 zeigt die zwischen den Komponenten ausgetauschten Nachrichten im Überblick. Einige Bezeichnungen sind weniger technisch gewählt als in der Originalliteratur, um das

[10] Ein Credential ist eine kryptographisch gesicherte Bestätigung einer Eigenschaft, z. B. einer Berechtigung.

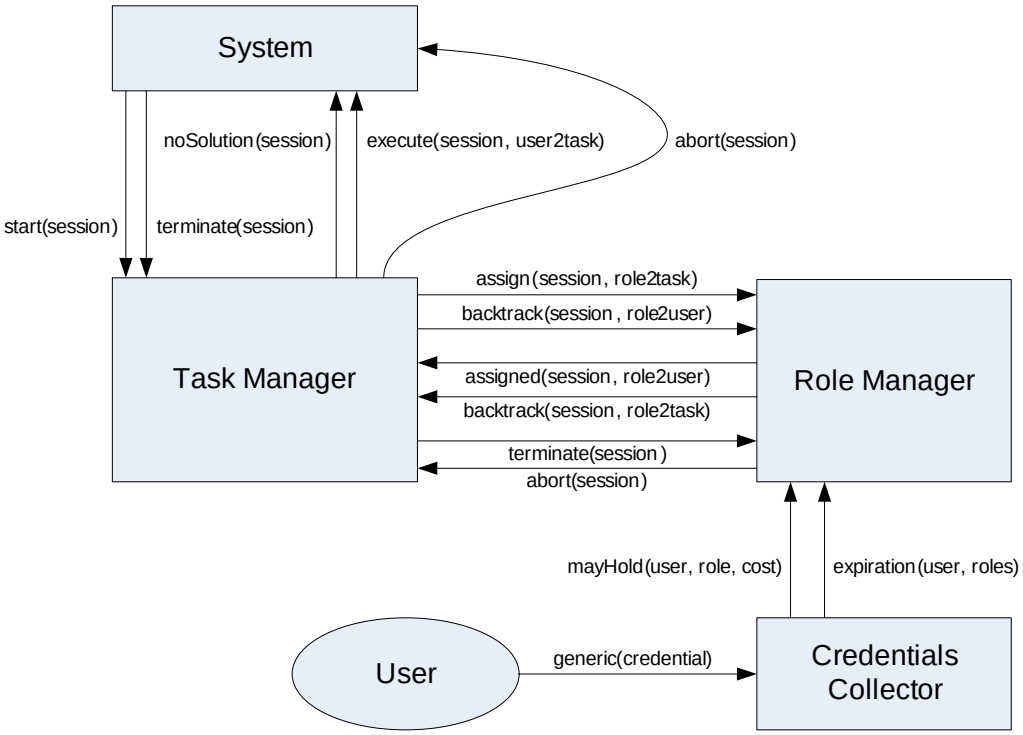

Abbildung 4.4: Überblick über ausgetauschte Nachrichten im sicheren Workflow-System

Verständnis zu erleichtern. Nachfolgend werden die Arbeitsweisen der drei Komponenten skizziert. Wird dabei auf Nachrichten in der erwähnten Abbildung verwiesen, sind diese serifenlos notiert, z. B. generic(credential).

Credentials Collector (CC)

Ein Teilnehmer, der Tasks im fraglichen System ausführen möchte, meldet sich mit einer von den Autoren *generisches Credential* genannten Nachricht beim CC an. Diese Nachricht (in Abb. 4.4 generic(credential)) enthält eine Menge von Rollen, die der Nutzer gerne annehmen möchte und eine Adresse zu einem *credential repository* (Speicherort für die Credentials des Nutzers). Abhängig von einer in der Sicherheitspolitik des Systems spezifizierten *role-credentials*-Teilpolitik weiß der CC, welche solcher *spezifischen* Credentials ein Teilnehmer vorweisen muss, um eine bestimmte Rolle bei Bedarf zugewiesen zu bekommen. Neben diesen teilnehmerspezifischen Speicherorten kennt der CC zusätzlich noch Speicherorte für negative Credentials (*negated credentials*), mit denen er die von den Nutzern vorgelegten Credentials abgleicht. Wenn ein solches Negativccredential vorliegt, verleiht das etwaige zugehörige Erlaubniscredential, das am Speicherort des jeweiligen Teilnehmers zu finden ist, keine Berechtigung mehr. Weiterhin hält der CC

auch eine Datenbank *credentials_validity*, in der die registrierten Credentials mit ihren Ablaufdaten vermerkt sind. Falls ein Credential abgelaufen ist und am entsprechenden Speicherort kein erneuertes Credential zu finden ist, sendet der CC eine entsprechende Nachricht expiration(user, roles) an den Role Manager, in der er den entsprechenden Benutzer und die betroffenen Rollen nennt. Die Kosten einer Nutzer-Rollen-Zuordnung entnimmt der CC aus dem generischen Credential, das der Nutzer selbst vorgelegt hat.

Informationen über hinzugekommene Berechtigungen sendet der CC umgehend an den RM (mayHold(user, roles, cost)). Analog informiert der CC den RM auch über abgelaufene bzw. durch ein negatives Credential entzogene Berechtigungen (expiration(user,roles)).

Der Credentials Collector löst keine Constraintprobleme.

Role Manager (RM)

Die Zuständigkeit des Role Managers konzentriert sich darauf, unter der Einhaltung von Constraints eine Zuweisung von Teilnehmern zu den in einer bestimmten Session benötigten Rollen zu finden. Erhält der RM vom Task Manager (TM) eine Nachricht assign(session, role2task), dann berechnet ersterer eine kostenoptimale Zuweisung von Nutzern an die Rollen aus der gegeben Zuweisung von Rollen auf Tasks role2task. Die dabei zu berücksichtigenden Constraints zur Pflichtentrennung (*separation of duty*) können zwischen verschiedenen Rollen, Rollen und Teilnehmern, sowie verschiedenen Teilnehmern bestehen. Mithilfe einer Datenbank *workflow_assignments* wird verhindert, dass ein Nutzer in mehreren, parallelen Workflows aktiv ist. Die vom CC an den RM gesendeten Informationen verwendet der RM, um Mengen S_{R_i} zu verwalten, die spezifische Credentials $(user, R_i, cost)$ entsprechend der mayHold-Nachrichten enthalten. Aus diesen Mengen werden diejenigen Tupel entfernt, für die entsprechende *expiration*-Nachrichten empfangen wurden. Die spezifischen Credentials besagen, dass der Teilnehmer *user* die Rolle R_i annehmen darf und dabei Kosten *cost* verursacht.

Das entsprechende CSP $P_{RM} = (X_{RM}, D_{RM}, C_{RM})$ ist zu lösen, wenn der RM eine Nachricht assign(session, role2task) vom TM empfängt. Zunächst werden die Mengen S_{R_i} gemäß der statischen Constraints des RM bereinigt. Die resultierenden Credentials werden in Mengen S'_{R_i} abgelegt. Sollten also statische Pflichtentrennungs-Constraints verbieten, dass beispielsweise ein bestimmer Nutzer zu irgendeiner Zeit eine gegebene Rolle annimmt, so wäre ein entsprechendes Credential an dieser Stelle in S'_{R_i} nicht mehr enthalten. Das CSP hat dann folgende Form:

- Die Variablen X_{RM} sind gegeben durch die in role2task enthaltenen Rollen $R_1, \ldots, R_n$.
- Für jede der Rollen-Variablen R_i enthält die Domain $D_i \in D$ gerade die Teilnehmer u, für die in der bereinigten Menge S'_{R_i} ein entsprechendes Credential der Form (u, R_i, c) vorliegt: $D_i = \{u \mid (u, R_i, c) \in S'_{R_i}\}$

– Prädikate bilden die Menge der dynamischen Constraints C_{RM}. Dabei werden unter anderem Regeln für dynamische Pflichtentrennung berücksichtigt (vgl. dazu [MGLM04, Tab. 2]).

Ist ein Nutzer für eine Rolle in einer Session gefunden, wird dieser Nutzer in der Datenbank *workflow_assignments* als aktiv vermerkt, aus der er bei Terminierung oder Abbruch der Session wieder entfernt wird. Damit wird ausgeschlossen, dass ein Nutzer in mehreren Workflows gleichzeitig aktiv ist.

Task Manager (TM)

Ein Task Manager wird benötigt, um die Sessions einer Workflow-Definition zu verwalten. Seine Aufgabe besteht in der Zuordnung autorisierter Benutzer zu Tasks einer Session. Der TM wahrt Beschränkungen zwischen verschiedenen Tasks, Tasks und Benutzern, sowie Tasks und Rollen. Dazu fragt der TM beim RM mit einer Nachricht assign(session, role2task) nach einer Zuordnung von Nutzern zu den Rollen aus der Rollen-zu-Tasks-Zuweisung role2task, die der TM aus der Lösung seines CSPs gewinnt. Der TM hält eine Datenbank *instances_assignments*, in der Tupel der Form $(user, task, session)$ abgelegt sind und die die tatsächliche Zuordnung von Nutzern zu Tasks in einer Session festhält, sobald diese gestartet wurde. Im Fall eines Abbruchs oder der regulären Terminierung einer Session werden die entsprechenden Einträge wieder aus der Datenbank entfernt.

Das von einem TM zu lösende Constraintproblem $P_{TM} = (X_{TM}, D_{TM}, C_{TM})$ bezüglich einer Workflow-Definition W hat folgende Gestalt:

– Die Variablen X_{TM} sind die Tasks $T_1, \ldots, T_m$ der verwalteten Workflow-Definition.

– Die Domains D_i für jeden der Tasks T_i enthalten die Rollen, die gemäß der Politik des TM für den jeweiligen Task nötig sind. Dabei werden Rollenhierarchien berücksichtigt. Rollen, die die benötigte Rolle enthalten, sind also ebenfalls in diesen Domains zu finden.

– Eine Reihe von Prädikaten (siehe dazu [MGLM04, Tab. 3 und 4]) steht zur Verfügung, um die erwähnten Beschränkungen zu formalisieren.

Die Lösung des CSPs ist die Grundlage für die entsprechende execute(session, user2task)-Antwort an das System.

4.4.2 Nachrichtenaustausch für die Nutzer-/Rollen-/Task-Zuordnung einer Session

Wir beschreiben knapp Arbeitsschritte und Reihenfolge für die Nutzer-/Rollen-/Task-Zuordnung für eine Session:

F1. System → TM: start(session)
Das System fordert eine Nutzer-/Rollen-/Task-Zuordnung für die angegebene Session an.

F2. TM bestimmt die role2task-Zuordnung. Gibt es keine konsistente Belegung in seinem CSP, sendet er eine noSolution(session)-Nachricht zurück an das System. In diesem Fall ist für die gegebene Session die TM-Politik zu restriktiv.

F3. TM → RM: assign(session, role2task)
Für eine gefundene Rollen-zu-Task-Zuordnung fragt der TM beim RM nach einer Zuweisung von Nutzern zu den Rollen aus role2task.

F4. Der RM versucht eine kostenminimale Belegung für die gegebenen Rollen zu finden. Lässt sich keine solche Belegung finden, setzt er eine backtrack(session, role2task)-Nachricht an den TM ab (weiter bei Schritt F2).

F5. Der RM registriert die Nutzer-zu-Rollen-Zuweisung in *workflow_assignment* und sendet die gefundene Zuordnung an den TM:
RM → TM: assigned(session, user2role)

F6. Der TM prüft die Nutzer-zu-Task-Zuweisung, die aus seiner role2task-Zuweisung und der user2role-Zuordnung des RM resultiert. Bestehen noch Konflikte mit den Constraints des TM, sendet er eine Backtrack-Nachricht an den RM:
TM → RM: backtrack(session, user2role) – danach weiter bei Schritt F5.
Sind die role2task- und user2role-Zuordnungen akzeptabel, wird die Session als gestartet registriert und das System benachrichtigt, dass die Session gestartet werden kann:
TM → System: execute(session, user2task).

Meldet sich ein neuer Benutzer beim CC an, werden die entsprechenden spezifischen Credentials eingesammelt und entsprechende mayHold(user, role, cost)-Nachrichten an den RM gesendet.

Für den Fall, dass ein Credential abläuft, ist Folgendes vorgesehen:

C1. CC → RM: expiration(user, roles)
Der CC informiert den RM, dass der Teilnehmer user die Rollen aus der Menge roles nicht mehr annehmen darf, da das entsprechende Credential abgelaufen ist.

C2. Der RM entfernt den Teilnehmer aus allen Domains der betroffenen Rollen-Variablen und sucht nach einer neuen passenden Belegung. Kann diese gefunden werden, teilt der RM die neue (ebenfalls auf geringe Kosten optimierte) Zuordnung user2role dem TM mit. Auf diese Weise angestoßen, setzt der TM die Abarbeitung ab Schritt F5 fort.

Gelingt keine neue Nutzer-Rollen-Zuordnung, sendet der RM eine abort(session)-Nachricht an den TM. In diesem Fall ist keine der Sicherheitspolitik entsprechende Ausführung des Workflows mehr möglich. Die Datenbanken im RM (*workflow_assignments*) und TM (*instances_assignments*) werden aktualisiert und auch der TM sendet eine abort(session)-Nachricht an das System. Damit wird die aktuelle Ausführung des Workflows abgebrochen.

Schließlich ist noch zu betrachten, was geschieht, wenn eine Session regulär terminiert:

T1. System → TM: termination(session)

T2. Der TM markiert session als beendet und entfernt die relevanten Einträge aus der Datenbank *instance_assignments*.

T3. TM → RM: termination(session)

T4. Der RM entfernt die für session relevanten Einträge aus der Datenbank *workflow_assignments*.

4.4.3 Relaxierende Betrachtungen

Moodahi et al. beschreiben in [MGM05], dass alternativ zur allgemein geltenden NP-Härte des Gesamtproblems in Spezialfällen Approximierungen möglich sind. Werden die Constraints gänzlich fortgelassen, lässt sich das betrachte Problem der Zuweisung von Nutzern zu Rollen zu Tasks auf das sogenannte generische Zuordnungsproblem (*generic assignment problem*) reduzieren, für das die NP-Härte bewiesen wurde.

Einige Relaxierungen der ursprünglichen Problemstellung erlauben den Einsatz spezieller Verfahren, etwa einfachen Polynomialzeit-Algorithmen. Andere Relaxierungsvarianten lassen eine Reduzierung auf das Max-Flow-Problem zu, für das ebenfalls effiziente Algorithmen existieren (etwa Edmonds-Karp oder Ford-Fulkerson). Da diese Varianten jedoch sämtlich auf spezielle Lösungsverfahren setzen und damit nicht mehr auf Constraintprobleme aufbauen, soll eine genauere Beschreibung hier entfallen.

4.5 Verteiltes Forward-Checking

Um die beispielhafte Betrachtung der Anwendung von CSPs im Kontext von Sicherheitsfragestellungen abzuschließen, verlagern wir den Fokus. Während die bisher in diesem Kapitel vorgestellten Ansätze sicherheitsrelevante Fragestellungen als klassisches oder weiches CSP formulieren, haben Brito und Meseguer [BM03] einen Algorithmus beschrieben, der es erlaubt, CSPs unter Wahrung von Vertraulichkeitsinteressen zu lösen.

Dieser *Distributed Forward Checking* genannte Algorithmus vereint Ideen aus dem asynchronen Backtracking-Algorithmus (ABT) von Yokoo (siehe Kap. 3.3) und dem Forward Checking (siehe Kap. 2.6.2). Dabei stellt das Verfahren eine Erweiterung des asynchronen Backtracking dar und verwendet lediglich die Idee der vorwärtsgerichteten[11] Domainfilterung aus dem Forward-Checking-Verfahren.

Die Hauptmotivation für das verteilte Forward-Checking liegt in dem Wunsch nach Geheimhaltung von sowohl angenommenen Variablenwerten als auch der jeweils geltenden Constraints. Informationen über Wertänderungen einer Variablen werden im ABT-Algorithmus aus zwei Gründen ausgetauscht: Zum einen soll die Konsistenz der aktuellen Belegung gewährleistet werden (bezogen auf Agenten mit höherer Priorität), zum anderen sorgt die Aktualisierung der Sichten auf die anderen Agenten (nach dem Empfangen eines neuen Werts) dafür, dass gegebenenfalls verzögert eintreffende Backtracking-Nachrichten verworfen werden können. Um die Variablenwerte geheimzuhalten, verwenden Brito und Meseguer sogenannte Sequenznummern (*sequence numbers*), die als Pseudonyme für die eigentlich zur Verfügung stehenden Werte Anwendung finden. Dazu erzeugen die Agenten für jeden zu kommunizierenden Wert eine eindeutige Sequenznummer, die ihnen selbst – jedoch keinem anderen Agenten – die zweifelsfreie Zuordnung zu einem „echten“ Wert ermöglicht. Auch in einem zwischen zwei Agenten x_i und x_j geltenden Constraint C_{ij}, finden sich ausschließlich Sequenznummern als Stellvertreter für die Variablenwerte. Für die Beschreibung des Verfahrens verwenden wir Agenten und Variablen synonym und notieren alle Constraints als binäre Nogoods (ganz ähnlich wie in Kapitel 3 zu verteilten Constraintproblemen).

4.5.1 Bekanntheitsmodelle für die Constraints

Für die Verwaltung der Constraints werden zwei alternative Modelle angeboten, unter anderem, um den Effekt der eingeführten Geheimhaltung besser beobachten zu können. Die Autoren unterscheiden zwischen *Totally Known Constraints* (TKC) und *Partially Known Constraints* (PKC). Im TKC-Modell kennen beide Agenten x_i und x_j das gesamte Constraint C_{ij}, während ihnen im PKC-Modell jeweils nur Ausschnitte daraus bekannt sind.

Wir betrachten zunächst das TKC-Modell. Hierfür sei der Geltungsbereich (*scope*) eines Constraints C_{ij} gegeben durch $\text{var}(C_{ij}) = \{x_i, x_j\}$; dieser ist beiden Agenten x_i und x_j bekannt. Die eigentliche Relation, die durch das Constraint beschrieben wird, ist durch $\text{rel}(C_{ij})$ repräsentiert und ist nur einem der beteiligten Agenten bekannt. Diese Funktion lässt sich vermutlich als Menge der gültigen Wertekombinationen von x_i und x_j auffassen.

[11] Die Idee des Forward Checking ist, während der Lösung des CSPs problematische Werte aus den Domains zu entfernen. Damit erübrigt sich das Preprocessing wie etwa beim Filteralgorithmus von Waltz (siehe Kap. 2.8.1). Durch die vorausschauende Manipulation der Domains wird Backtracking vermieden.

Im PKC-Modell werden statt einem Constraint C_{ij} zwei Constraints verwaltet: Agent x_i kennt das Constraint $C_{i(j)}$ mit $\text{var}(C_{i(j)}) = \{x_i, (x_j)\}$ und Agent x_j kennt das Constraint $C_{(i)j}$ mit $\text{var}(C_{(i)j}) = \{(x_i), x_j\}$. Die umklammerten Indizes bzw. die zugehörigen Variablen sollen dabei bedeuten, dass „ein Agent nur wenig über die andere Variable des Constraints weiß“ [BM03, S. 802]. Die zugehörigen Relationen sind unschärfer gefasst als die eigentlich durch das Constraint ausgedrückte Relation, d. h. sie enthalten nun auch möglicherweise verbotene Wertekombinationen. Formal gilt: $\text{rel}(C_{ij}) \subseteq \text{rel}(C_{i(j)})$ und entsprechend $\text{rel}(C_{ij}) \subseteq \text{rel}(C_{(i)j})$. Der Schnitt dieser beiden rel-Mengen darf ausschließlich die tatsächlich erlaubten Kombinationen enthalten: $\text{rel}(C_{ij}) = \text{rel}(C_{i(j)}) \cap \text{rel}(C_{(i)j})$.

4.5.2 Die TKC-Variante

Im einfacheren Fall, in dem ein Constraint nicht aufgesplittet wird, ist das Constraint C_{ij} lediglich dem höher priorisierten Agenten aus $\text{var}(C_{ij})$ bekannt. Wie beim ABT-Algorithmus ist auch hier eine totale Ordnung auf den Agenten erforderlich, die die Prioritäten der Agenten festlegt. Es werden zwei Arten von Nachrichten ausgetauscht:

Info Ändert ein Agent den Wert seiner Variablen x_i, so teilt er dem niedriger priorisierten Agenten x_j diesen neuen Wert in Form der entsprechenden Sequenznummer mit, zusammen mit einer Menge damit verträglicher Werte für x_j, die er aus $\text{rel}(C_{ij})$ und seinem eigenen neuen Wert ermittelt. Zudem ergänzt der empfangende Agent x_j seine Menge bekannter Nogoods: Alle durch x_i für x_j ausgeschlossenen Werte werden mit dem neuen Wert für x_i als binäre Nogoods eingetragen. Da x_i diese Werte ausgeschlossen hat, verursachen Kombinationen dieser Werte mit dem neuen, gerade empfangenen Wert Konflikte. Hier kommt der Forward-Checking-Charakter zur Geltung.

Back Für den Fall, dass Backtracking durchgeführt werden muss, sendet ein niedrig priorisierter Agent x_j eine Back-Nachricht an den höher priorisierten Agenten x_i aus dem verletzten Constraint C_{ij}. Diese Nachricht enthält gerade das gefundene Nogood, also eine verbotene Kombination von Variablenwerten.

Die TKC-Variante arbeitet vollständig korrekt, ebenso wie der ABT-Algorithmus.

4.5.3 Die PKC-Variante

Wie oben beschrieben wird ein Constraint C_{ij} durch zwei entsprechende Constraints $C_{i(j)}$ und $C_{(i)j}$ ersetzt. Der Algorithmus für diese Variante durchläuft eine Schleife mit zwei Phasen:

(I) Aus den Constraints wird ein DAG[12] konstruiert, um eine passende totale Ordnung

[12] Ein DAG ist ein gerichter, kreisfreier Graph (directed acyclic graph).

auf den Variablen zu finden. Anschließend wird eine Lösung für die Constraints der Form $C_{i(j)}$ gesucht. Lässt sich keine passende Belegung finden, ist das Problem überbeschränkt und der Algorithmus bricht ab.

(II) Die Lösung aus Phase I wird auf Verträglichkeit mit den Constraints der Form $C_{(i)j}$ geprüft. Ist die Belegung konsistent, handelt es sich dabei um eine Lösung; andernfalls werden entsprechende Nogoods generiert und Phase I wird erneut durchgeführt.

Da Constraints immer nur von höherpriorisierten Agenten geprüft werden (um anschließend ggf. einen neuen Wert und kompatible Domains zu versenden) und $C_{(i)j}$ nur x_j bekannt ist, muss j eine höhere Priorität als x_i haben. Daher wird zu Beginn der zweiten Phase der konstruierte DAG umgekehrt, d. h. die gerichteten Kanten werden in umgekehrter Richtung interpretiert. Nimmt ein nun höher priorisierter Agent x_j bezüglich des Constraints $C_{(i)j}$ einen neuen Wert an, so verschickt er Info-Nachrichten an x_i. Dieser Agent prüft wiederum das Constraint $C_{i(j)}$ und sendet bei Unverträglichkeit der Werte eine Back-Nachricht an x_j, welcher die übermittelten Nogoods registriert. Sind schließlich alle Info- und Back-Nachrichten ausgetauscht und wurde noch keine Lösung gefunden, wird der DAG wieder in seine ursprüngliche Form zurückversetzt und Phase I beginnt erneut. Die in Phase II generierten Nogoods wirken sich nun in Phase I aus.

Die Schleife wird (neben dem in Phase I genannten Fall) verlassen, falls eine Lösung gefunden oder das leere Nogood generiert wurde. Da ein leeres Nogood-Constraint nie erfüllt werden kann, besteht in diesem Fall Überbeschränkung.

Die Autoren sprechen von erhöhtem Kommunikations- und Berechnungsaufwand, geben dazu jedoch keine Details an. Weiterhin ist unklar, wie resistent die Pseudonymisierung der Variablenwerte gegenüber einfachen Inferenzattacken ist. Erwirbt ein Agent Kenntnisse über die allgemeine Struktur des Problems, wären unter Umständen Informationen über die tatsächlichen Werte ableitbar.

4.6 Verwandte Ansätze

Die in diesem Kapitel bisher beschriebenen Verfahren nutzen Constraintprobleme auf verschiedene Weisen, um damit spezielle Probleme der Sicherheit in Rechensystemen zu lösen. Bevor wir im nächsten Kapitel versuchen werden, eine allgemeine Beschreibung der Gruppe von Sicherheitsproblemen zu finden, die sich mit CSPs lösen lassen, geben wir noch einen knappen Überblick über verwandte Ansätze.

Sicheres verteiltes Constraintlösen Ein weiteres Verfahren, mit dem sich CSPs unter Einhaltung von Vertraulichkeitsanforderungen lösen lassen, stellen Yokoo et al. [YSH02] vor. In ihrem *Secure DisCSP* genannten Algorithmus verwenden sie zentrale, steuernde

Komponenten: Es gibt eine Komponente, die die Suche nach einer konsistenten Belegung steuert (Suchsteuerer, *search controller*); mindestens zwei Entschlüsselungskomponenten (*decryptors*), die ausschließlich gemeinsam Nachrichten entschlüsseln können; einen Wertauswähler (*value selector*) für jede im Problem vorkommende Variable. Das Problem wird vornehmlich durch den search controller und die Wertauswähler gelöst. Zu Beginn senden die jeweils eine Variable behandelnden Agenten ihre unären und binären Constraints in verschlüsselter Form an die Wertauswähler, den Suchsteuerer und die Entschlüsselungskomponenten. Finden die zentralen Komponenten eine Lösung, senden diese die konsistente Belegung an die Agenten. Für die beteiligten Agenten gleicht damit die Problemlösung einer Orakelanfrage. Die Verwendung asymmetrischer Kryptographie ist (bezogen auf die Vertraulichkeitsanforderungen) der einfacheren Wertverheimlichung mittels Sequenznummern zweifellos überlegen. Die zusätzliche Aufteilung der Entschlüsselung auf mehrere Wertentschlüssler verspricht zudem eine höhere Sicherheit für aus dem System selbst stammende Angriffe. Wie auch beim verteilten Forward-Checking bringen die eingeführten Sicherungsmaßnahmen höheren Kommunkations- und Berechnungsaufwand mit sich, verglichen mit einfachen, verteilten Algorithmen.

Rollenbasierte Zugriffskontrolle (RBAC) mit Constraints Ahn und Sandhu [AS01] entwickeln auf der Grundlage des aus dem Jahre 1996 stammenden Standards für rollenbasierte Zugriffskontrolle eine Erweiterung, die Constraints – also Beschränkungen – berücksichtigt. Mit Hilfe dieser Constraints soll unter anderem dynamische Pflichtentrennung realisiert werden, wenn ein Benutzer eine Rolle in Anspruch nehmen möchte. Allerdings formuliert dieser Ansatz keine zu lösenden Constraintprobleme, sondern erweitert ein bestehendes Modell. Eine Realisierung mit Hilfe von CSPs würde hier vermutlich auf ein ähnliches Konzept wie bei der Nutzer-/Rollen-/Task-Zuordnung im Workflow-Management-System aufbauen (siehe auch Kapitel 4.4). Barker und Stuckey [BS03] zeigen, wie sich einige RBAC-Modelle als Constraint-Logic-Programme darstellen lassen, was eine Bereicherung der bisher gängigen Techniken (etwa über einfache, logische Programme) darstellt. Auch Zugriffskontrollpolitiken werden auf CLPs abgebildet, um dort bestimmte Überprüfungen vornehmen zu können. Hier besteht keine direkte Verbindung zu CSPs.

Constraints im Flexible Authorization Framework Mit dem Flexible Authorization Framework (FAF) [JSSS01] steht ein auf logischer Programmierung mit Strata basierendes Zugriffskontrollsystem zur Verfügung. Im FAF spezifiziert ein Administrator die Sicherheitspolitik über ein lokal stratifiziertes, logisches Programm, das eine eigens entwickelte Sprache verwendet. Damit lassen sich funktionale Berechtigungen (etwa lesen, schreiben und ausführen) von Benutzern bezüglich Rollen und Benutzern oder Rollen bezüglich Objekten spezifizieren. Die Hinzunahme von Qualifizierungen ermöglicht die Spezifikation von sowohl Erlaubnissen als auch Verboten; dabei stellt das Framework ebenfalls Mechanismen zur Konfliktlösung bereit, falls zwei gegensätzliche Privilegien gleichzeitig auftreten. Chen et al. [CWJ04] stellen nun eine Erweiterung dieses Fram-

works zum sogenannten dynFAF vor, um einige identifizierte Mängel zu beseitigen. Unter anderem erlaubt dynFAF die Einbeziehung von kontrolliertem Nichtdeterminismus: Beispielsweise kann es egal sein, welche von zwei Rollen ein User annimmt, solange er nur *eine* von beiden annimmt. Zudem sollte es möglich sein, bestimmte Privilegien auch zurückziehen zu können. Durch die Spezifikation von atomaren, binären Konflikten (die gewisse funktionale Operationen verbieten) und komplexere, über Regeln abgeleitete Konflikte werden Constraints direkt in die Politik integriert. dynFAF vereint damit Ideen der Constraint-Logik-Programmierung mit einem bestehenden, auf logischen Programmen basierenden Zugriffskontrollsystem.

Beschränkte Optimierungsprobleme in einer serviceorientierten Architektur Swart et al. stellen ein formales Modell einer Service-orientierten Architektur (SOA) zusammen mit Metriken (und Kosten) für Performanz, Verfügbarkeit und verschiedene Sicherheitseigenschaften vor [SAFH05]. Da SOA-Werkzeuge häufig in Businessprozessen Anwendung finden, wird hier nach einem Kompromiss zwischen den erstrebenswerten Eigenschaften und einer günstigen Lösung gesucht. Darauf basierend wird ein Constrained-Optimization-Problem formuliert, für dass die Gesamtkosten minimiert werden sollen. Die Autoren verwenden dazu die Optimization Programming Language (OPL). Als Ergebnis der Berechnung steht eine Menge gemäß der gegebenen Kriterien optimaler Zuweisungen logischer Komponenten an physikalische Ressourcen.

Anonymitätsschutz bei der Auswertung von GPS-Daten Mit der zunehmenden Verbreitung des Global Positioning System (GPS), insbesondere in Form von satellitengestützter Navigationssoftware in Fahrzeugen, stellt sich die Frage, inwiefern die GPS-Koordinaten verwendet werden können, um Informationen aus dem jeweiligen Umfeld des Empfängers zu aggregieren. Autohersteller und Verkehrsüberwachungseinrichtungen versuchen, die Positionsdaten auszuwerten, um beispielsweise verlässliche Angaben zu Staus und Verkehrsnutzung zu machen. Hoh und Gruteser [HG05] haben zum Schutze der Privatsphäre der GPS-Nutzer eine Idee ausgeführt, die bei Treffpunkten von zwei Teilnehmern deren „Spuren“ vertauscht, sodass keine zuverlässige Zuordnung von GPS-Daten auf Nutzer mehr möglich ist. Dieses Szenario wird als Constrained-Optimization-Problem formuliert und versucht, die Privatsphäre der Teilnehmer zu wahren und die Positionsdaten maximal zu verzerren (*perturbation*), ohne den Nutzwert dieser für den jeweiligen Anwendungszweck zu stark einzuschränken. Der Ansatz verwendet schließlich Heuristiken für einen effizienten Algorithmus, verwendet also zumindest nicht ausschließlich bereits verfügbare CSP- bzw. COP-Lösungsmethoden.

Analyse kryptographischer Protokolle Neben der ausführlich in Kapitel 4.2 vorgestellten Technik zur Analyse von Sicherheitsprotokollen mittels weicher Constraints gibt es eine Idee von Corin et al. [CMAFE03], das neue *guessing attacks* (Angriffe basierend auf schlechten Passwörtern und mitgehörten Nachrichten) in Protokollen finden kann und

dabei einen Constraint-Löser verwendet.

5 Charakterisierung und Klassifizierung

Nachdem wir nun einige Anwendungen von CSPs zur Lösung von sicherheitsrelevanten Problemen betrachtet haben, stellt sich die Frage, was die dargestellten Ansätze gemein haben. Es wäre sicher wünschenswert, über Kriterien zu verfügen, die beim Betrachten eines Problems aus dem Bereich der Sicherheit Aufschluss darüber geben, ob sich diese konkrete Fragestellung als Constraintproblem formulieren lässt.

5.1 Vergleich der dargelegten Anwendungen

In Tabelle 5.1 sind die inhaltlichen Aspekte der vorgestellten Ansätze übersichtsartig zusammengefasst. Jede Zeile stellt die wichtigsten Merkmale der in den genannten Abschnitten und Kapiteln vorgestellten Ideen heraus; betrachtet werden die verwendeten Variablen und ihre Domains mitsamt der aufgestellten Constraints. Ergänzend dazu zeigt die ganz ähnliche Tabelle 5.2 strukturelle Eigenschaften der Kennmerkmale in den verschiedenen Arbeiten. Wir versuchen nun zunächst, etwaige Gemeinsamkeiten in diesen Merkmalen aufzudecken.[1]

Variablen Für die Variablen wird in jedem CSP eine Zuweisung gesucht. Hier sind es Zeitfenster, Rollen, Tasks, Agenten und Systemknoten. Die verschiedenen Interpretationen der Variablen haben eine eher feste Struktur und eine Unveränderlichkeit gemeinsam. Besonders ausgeprägt sind die strukturierenden Merkmale bei den Zeitfenstern: zehn Termine am Tag an sieben Tagen in der Woche. Der Task Manager aus dem Workflow-Management-Ansatz verwendet Tasks aus einer Workflow-Definition, die wiederum aus der Strukturierung eines Arbeitsablaufs hervorgegangen ist. Der zugehörige Role Manager muss eine Belegung für die in einem Workflow benötigten Rollen finden, welche gerade deshalb verwendet werden, weil sie sich im Gegensatz zur tatsächlichen Nutzerbasis in einer Organisation eher kaum ändern. Bei der Analyse von Sicherheitsprotokollen mittels weicher Constraints scheint es zunächst so, als ob die Agenten eine dynamische Größe darstellen könnten, doch tatsächlich wird immer nur ein spezielles Szenario der Protokollausführung betrachtet, in dem die teilnehmenden Agenten wiederum unveränderlich sind. Für die Entdeckung etwaiger Kaskaden-Verwundbarkeiten (letzte Zeile in Tab. 5.1) werden Systemknoten-Variablen verwendet, deren Struktur einzig von der Struktur des untersuchten Netzwerks abhängt. Dieses ist ebenfalls unveränderlich und legt mit der

[1] Verständlicherweise erheben beide Tabellen keinen Anspruch auf Vollständigkeit.

Szenario	**Variablen**	**Domains**	**Constraints**	**Bemerkungen**
Terminvereinbarung (Kap. 4.1)	Zeitfenster	Orte	Reisezeitconstraint	jeder Agent verwaltet CSPs für alle anderen Agenten
WFMS Task Manager (Kap. 4.4)	Tasks aus dem Workflow	gemäß Workflow-Definition benötigte Rollen	Verbote, Erlaubnisse, Pflichten	Schwierigkeit der Synchronisation ohne große Blockierung
WFMS Role Manager (Kap. 4.4)	für den Workflow benötigte Rollen	gemäß Credentials berechtigte User	z. B. Pflichtentrennung	
Protokollanalyse mit SCSPs (Kap. 4.2)	Agenten	atomare und abgeleitete Nachrichten	definieren Bekanntheitsgrad einer Nachricht für einen Agenten	Politik explizit ausgedrückt; „Lösung“ ist Agentenwissen in bestimmtem Szenario
Kaskaden-Verwundbarkeit in MLS-Netzwerken (Kap. 4.3)	Systemknoten-Variablen	Schichten in MLS-Systemen (Teilsysteme)	gemäß Politik erlaubte / verbotene Flüsse und durch bestimmte Flüsse verursachte Risiken	Auffinden bedrohlicher Pfade

Tabelle 5.1: Übersicht der betrachteten Anwendungen von Constraintproblemen

Szenario	**Variablen**	**Domains**	**Constraints**
Terminvereinbarung (Kap. 4.1)	starke Strukturierung, große Anzahl	sehr kleine Menge	Reisezeitconstraint
WFMS Task Manager (Kap. 4.4)	topologisch sortiert	—	einfache Regeln
Protokollanalyse mit SCSPs (Kap. 4.2)	in der Regel geringe Anzahl	werden mittels Entailment-Regeln erzeugt	—
Kaskaden-Verwundbarkeit in MLS-Netzwerken (Kap. 4.3)	Anzahl gegeben durch doppelte Anzahl Teilsysteme	in der Regel kleine Anzahl	—

Tabelle 5.2: Auffällige Strukturmerkmale in den betrachteten Anwendungen von Constraintproblemen

zusammengenommenen Anzahl der Schichten aller Systeme die maximale Länge eines Pfades fest.

Domains Auch die Inhalte der verwendeten Domains sind im Grunde statisch und werden generiert, bevor mit der Lösung des CSPs begonnen wird. Eine Ausnahme bildet hier der Role Manager aus dem Workflow-Ansatz. Dort verändern sich die Domains abhängig von den durch die vorhandenen Credentials gegebenen Berechtigungen, die über den Credentials Collector aktualisiert werden. Allerdings ist hier anzumerken, dass eine Änderung der Domains in den meisten Fällen wohl einen Neustart der Lösungsfindung verursachen wird. Inwiefern hier also eine dynamische Komponente einen Zusatznutzen nach sich zieht, bleibt fraglich. Alle betrachteten Domains stellen zudem diskrete und tatsächlich endliche Definitionsbereiche für die Variablen dar. Obwohl prinzipiell beispielsweise bei der Protokollanalyse unendlich viele Nachrichten konstruiert werden können, ist die Anzahl der möglichen Werte in der Domain tatsächlich abhängig vom betrachteten Ausführungsszenario und damit wieder endlich groß. Eine Veränderung, die die Domains während der Bearbeitung eines CSPs erfahren, ist möglicherweise eine Reduktion, da durch gegebenenfalls angewandte Konsistenzverfahren unverträgliche Werte entfernt werden können. So bleiben die Domains im Endeffekt unverändert oder werden eingeschränkt, nicht jedoch erweitert.

Constraints Hier muss zunächst unterschieden werden zwischen den Constraints der klassischen CSPs in den Ansätzen der ersten bis dritten Zeile in Tab. 5.1 und den weichen Constraints aus den letzten beiden Zeilen. Bei den klassischen Constraints finden wir einfach gehaltene Constraints, von denen das Reisezeitconstraint besonders kompakt ist. Die beiden Workflow-Ansätze erhalten ihre Constraints im Wesentlichen aus Listen erlaubter und verbotener Kombinationen von beispielsweise Rollen und Benutzern. Was die weichen Constraints betrifft, finden wir dort in beiden Fällen einen akkumulierenden Charakter vor: Bei der Protokollanalyse werden die monoton zunehmenden Bekanntheitsgrade der Nachrichten beobachtet, während bei der MLS-Netzwerk-Untersuchung die entsprechenden Risiken aufgesammelt werden, um letztendlich den nummerischen Vergleich zwischen zwei Werten zu ermöglichen.

5.2 Beobachtungen

Was macht nun ein CSP aus? Alle Constraintprobleme sind primär Suchprobleme. Durch die möglichen Belegungen der Variablen wird ein Zustands- bzw. Suchraum aufgespannt, in dem sich mögliche Lösungen des Problems verbergen. Aus dieser Struktur heraus ergibt sich auch die Eignung von Backtracking-Verfahren (zusammen mit filternden Konsistenzansätzen) für die Behandlung der Probleme. Der vorangegangene Abschnitt hat Hinweise darauf gegeben, dass das Problem im Grunde schon scharf umrissen sein muss

und insbesondere kaum Platz für Dynamik in den Variablen und Domains ist, ohne die Performanz der Lösungsfindung erheblich zu beeinträchtigen.

Der aufgespannte Zustandsraum muss also schon „ausgefaltet“ vorliegen, d. h. der Zustandsraum darf sich nicht erst dynamisch entwickeln, wenn die Lösung des CSPs beginnt. Am Beispiel: Bei der Protokollanalyse mittels SCSPs ist in den Variablen, Domains und Constraints des Politik-SCSPs bereits kodiert, welche Schritte für das betrachtete Protokoll erforderlich sind. Ein Zuschreibungs-SCSP auf der anderen Seite stellt ein durch einen eventuell vom Protokollverlauf abweichenden Nachrichtenaustausch entstehendes Szenario dar, das auf Protokollkonformität geprüft werden soll: Hier werden also effektiv zwei statische Szenarien miteinander verglichen. Insbesondere werden nicht direkt alle möglichen Protokollanwendungen betrachtet.

Generell ist zu bemerken, dass die einzige Dynamik bei der Bearbeitung eines CSPs ausschließlich aus den Constraints stammt. Da diese auch noch zur Bearbeitungszeit beispielsweise Anfragen an Datenbanken stellen oder komplexe Auswertungen von z. B. Politik-Richtlinien vornehmen können, lässt sich ein statisch als CSP modelliertes Szenario eventuell dadurch um die gegebenenfalls benötigte dynamische Komponente anreichern. In der Formalisierung der klassischen Constraintprobleme (wie in Kapitel 2) lässt die Beliebigkeit der Constraints dort genug Spielraum, um ein Constraint angemessen zu gestalten. Weiche Constraints wiederum werden durch eine Funktion repräsentiert, die einer Belegung von Variablen eine Bewertung zuordnet. Auch diese Funktion kann dabei durchaus nicht direkt im CSP vorhandene Informationen einfließen lassen, etwa wieder aus einer Datenbank.

5.3 Eignung für Sicherheitsprobleme

In Tabelle 5.3 sind die Sicherheitsaspekte aus den in Kapitel 4 vorgestellten Beispielen hervorgehoben. Zu erkennen ist eine Schwerpunktsetzung auf Probleme, die sich mit der Vertraulichkeit von Informationen oder Nachrichten beschäftigen. Als möglicher Konflikt zu den verfolgten Sicherheitszielen tritt die weniger effiziente Lösungsfindung in Erscheinung, der jedoch kein besonderes Gewicht zukommt, da die Einbringung von Sicherheitsaspekten in bis dahin „unsichere“ Systeme generell Performanzeinbußen nach sich zieht. Da der Forschungszweig, der sich mit der Verbindung von CSPs mit Sicherheit in Rechensystemen beschäftigt, jedoch erst wenige Jahre alt ist, sind hieraus keine verlässlichen Schlussfolgerungen in Bezug auf die Eignung von Problemen mit bestimmten Schutzzielen gegenüber anderen ableitbar.

Für die lediglich skizzierten Verfahren zeigt Tabelle 5.4 eine gleichartige Übersicht. Mit der Loslösung von streng CSP-basierten Ideen und der Hinwendung zu allgemeinen Ideen mit Constraints, schwerpunktmäßig der Constraint-Logikprogrammierung (siehe dazu auch [FA97]), rückt eher die Erweiterung bestehender Konzepte um genauere Ausgestaltungsmöglichkeiten und differenziertere Betrachtungen in den Vordergrund.

Szenario	**Sicherheitsaspekte**	**mögliche Konflikte**
Terminvereinbarung (Kap. 4.1)	Vertraulichkeit	Effiziente Lösungsfindung
WFMS (Kap. 4.4)	Zugriffskontrolle	—
Protokollanalyse mit SCSPs (Kap. 4.2)	Protokollsicherheit, Vertraulichkeit, Authentizität	—
Kaskaden-Verwundbarkeit in MLS-Netzwerken (Kap. 4.3)	Vertraulichkeit	—
Verteiltes Forward-Checking (Kap. 4.5)	Vertraulichkeit	Effiziente Lösungsfindung

Tabelle 5.3: Sicherheitsaspekte in den betrachteten Anwendungen von Constraintproblemen

Szenario	**Sicherheitsaspekte**	**mögliche Konflikte**
Sicheres verteiltes Constraintlösen	Vertraulichkeit	Effiziente Lösungsfindung
RBAC mit Constraints	Erweiterung der Ausdrucksmöglichkeiten des RBAC-Konzepts	—
dynFAF	Erweiterung der Ausdrucksmöglichkeiten des FAF	—
Beschränkte Optimierungsprobleme für SOA	Kostenminimale Zuweisung von logischen Einheiten an physikalische Komponenten	—
Anonymitätsschutz bei GPS-Datenauswertung	Anonymität / Privatsphäre	detaillierte Auswertung aggregierter Daten
Analyse kryptographischer Protokolle	Protokollsicherheit, Resistenz gegen guessing attacks	—

Tabelle 5.4: Sicherheitsaspekte in den verwandten Ansätzen

So steht an dieser Stelle die Vermutung, dass die Eignung von Sicherheitsproblemen für die Behandlung mit CSPs nur auf die strukturellen Merkmale des Problems, nicht aber bestimmte, verfolgte Ziele zurückzuführen ist. Zudem scheint es unumgänglich, einen erheblichen Aufwand in die Kodierung des CSP zu investieren. Dies schmälert nicht die Leistungsfähigkeit und Relevanz von Constraintproblemen für den Themenkomplex der Sicherheit, verlangt jedoch ein gewisses Geschick vom jeweiligen Anwender bezüglich der Codierung des Problems.

6 Fazit und Ausblick

Constraintprobleme sind ursprünglich als Lösungsverfahren für geometrische Probleme entstanden und wurden seit den späten 1970er Jahren zu einer leistungsfähigen Methode weiterentwickelt, deren Hauptaugenmerk auf einer möglichst intuitiven Beschreibung des Problems und nicht der Lösung liegt. Mit Hilfe neuerer Techniken wie partieller Constraintlösung oder weicher Constraints stehen vielfältige Möglichkeiten offen, auch komplizierte Probleme zu behandeln, die nicht von den klassischen CSP-Formulierungen erschlagen werden. Das Anwendungsfeld für derartige Probleme ist klar abgesteckt: Suchprobleme mit teils sehr großen Suchräumen, die prinzipiell schwer zu begutachten sind. Durch gute Heuristiken werden jedoch auch die Probleme mit besonders ausladenden Suchräumen in adäquater Zeit lösbar und zumindest für die klassischen CSPs gibt es auch Algorithmen, die die verteilte Behandlung erlauben.

Die Idee, Constraintprobleme in Szenarien anzuwenden, die sichere Rechensysteme bzw. Verfahren in diesen beschreiben, ist noch relativ jung und demnach gibt es erst wenige Anwendungen dafür. Dennoch: Die vorgestellten Ansätze zeigen, wie Probleme, deren Schwierigkeit hauptsächlich in der Größe des Suchraums begründet liegt, erfolgreich bewältigt werden können, wenn sie präzise als Eingabe für klassische oder weiche Constraintprobleme formuliert werden. Es ist in dieser Arbeit jedoch nicht gelungen, einen praktikablen Kriteriensatz zu finden, der für die Eignungsbeurteilung eines Sicherheitsproblems für die Behandlung mit CSPs verwendet werden kann. Was die Strukturmerkmale der zu lösenden Probleme betrifft, konnten einige kennzeichnende Eigenschaften identifziert werden, die eine grobe Richtung bei der notwendigen Analyse vorgeben können. Hier ist noch Raum für weitere Untersuchungen. Zudem lässt das wachsende Interesse an diesem Thema auch hoffen, dass weitere CSP-Formulierungen für bekannte und neue Probleme mit Sicherheitsbezug in den Blickpunkt rücken werden.

Mögliche Ansatzpunkte für weitere Betrachtungen liegen daher sicher vornehmlich bei der Charakterisierung geeigneter Sicherheitsprobleme. Ein im Rahmen dieser Arbeit unternommener Versuch, für das *take-grant*-Zugriffsmodell [LS77] die Frage nach der Erreichbarkeit einer bestimmten Privilegien-Konstellation mittels eines CSPs zu entscheiden, ist an der Dynamik des Prozesses gescheitert. Dabei sollte ausgehend von einer bestimmten Konstellation geprüft werden, ob ein bestimmter Teilnehmer jemals ein gewisses Recht erhalten kann. Möglicherweise lässt sich im Bereich der Zugriffskontrolle dennoch eine Anwendung finden, etwa bei der kontrollierten Anfrageauswertung [Bur06]. Durch den Charakter der Constraints ergibt sich eine Eignung für Probleme, bei denen Kompromisse (trade-offs) im Mittelpunkt stehen, die speziell über weiche Constraints

sogar eine gewisse Gewichtung erfahren können.

Insgesamt gesehen stellt die Verwendung von CSPs ein leistungsfähiges Mittel der Problembehandlung bereit, dessen Handhabung jedoch einige Übung in der Formulierung der Problemstellung erfordert. Die Anwendung auf Szenarien mit Sicherheitsbezug ist durchaus denkbar und zum Teil auch praktikabel, wie etwa im Fall der Protokollanalyse mit weichen Constraints. Eine spezielle „Verwandschaft“ der beiden Gebiete, die die zu bewältigenden Aufgaben deutlich vereinfacht, ist jedoch anscheinend nicht gegeben.

A Lowes Angriff auf das Needham-Schroeder-Protokoll

Das Needham-Schroeder-Protokoll [NS78] wird verwendet, um zwei Nutzer gegenseitig zu authentifizieren. Das Protokoll sieht die Verwendung von secret-key- oder public-key-Kryptographie vor, wobei wir uns hier nur auf den public-key-Fall beschränken. Wenn wir davon ausgehen, dass zwei Teilnehmer gegenseitig ihre öffentlichen Schlüssel kennen, dann werden die folgenden Nachrichten ausgetauscht (dabei umschließen geschweifte Klammern eine Nachricht und der Index K_A an der schließenden Klammer steht für Verschlüsselung mit dem öffentlichen Schlüssel von A):

$$A \to B: \quad \{N_A, A\}_{K_B} \tag{A.1}$$
$$B \to A: \quad \{N_A, N_B\}_{K_A} \tag{A.2}$$
$$A \to B: \quad \{N_B\}_{K_A} \tag{A.3}$$

Zunächst (in Nachricht A.1) sendet A eine Nonce[1] N_A, zusammen mit seiner Identität „A“ an B und verschlüsselt diese Nachricht mit dem öffentlichen Schlüssel von B. B kann diese Nachricht mit dem zugehörigen geheimen Schlüssel entschlüsseln und sendet N_A zusammen mit einer weiteren, von ihm erzeugten Nonce N_B zurück an A. Da nur B über den privaten Schlüssel zu K_B verfügt, kann nur dieser die Nachricht entschlüsseln. A erkennt an der zurückgesendeten Nonce N_A, dass B die Nachricht entschlüsselt hat und sendet im letzten Schritt (A.3) die von B erzeugte Nonce zurück. Somit sind beide Teilnehmer davon überzeugt, dass ihr jeweiliger Kommunikationspartner der ist, den sie erwarten und authentifizieren sich in nachfolgenden Nachrichten durch die ausgetauschten Nonces.

Lowe [Low95] hat jedoch festgestellt, dass mit einem Mittelsmann, der jeweils vorgibt, einer der beiden Teilnehmer zu sein, das Protokoll nicht den beabsichtigten Effekt liefert. Wir bezeichnen den Mittelsmann als $I(A)$, wenn er in der jeweiligen Nachricht A imitiert.

$$A \to I(B): \quad \{N_A, A\}_{K_I} \tag{A.4}$$
$$I(A) \to B: \quad \{N_A, A\}_{K_B} \tag{A.5}$$
$$B \to A: \quad \{N_A, N_B\}_{K_A} \quad \text{(abgefangen)} \tag{A.6}$$
$$I(B) \to A: \quad \{N_A, N_B\}_{K_A} \tag{A.7}$$
$$A \to I(B): \quad \{N_B\}_{K_I} \tag{A.8}$$
$$I(A) \to B: \quad \{N_B\}_{K_B} \tag{A.9}$$

[1] Abkürzung für *number used once*, eine einmalig verwendete Zahl.

Kommunizieren A oder B mit I, so glauben sie stets, mit dem richtigen Teilnehmer „verbunden“ zu sein. Nachricht A.6 wird von I abgefangen und unverändert weitergeleitet – mit dem Unterschied, dass A glaubt, diese Nachricht von B bekommen zu haben, sie aber tatsächlich von I erhält. Nach Schritt A.9 kennt I beide Nonces N_A und N_B und kann diese in späteren Nachrichten an A oder B verwenden, um sich zu authentifizieren. Beheben lässt sich dieses Problem dadurch, dass B in Schritt A.3 zusätzlich seine Identität mitsendet:

$$A \rightarrow B : \qquad \{N_B, B\}_{K_A} \tag{A.10}$$

Dadurch weiß A, dass er seine Nachrichten direkt an B senden und auch mit dem Schlüssel von B verschlüsseln soll, wodurch der Mittelsmann-Angriff nicht mehr funktioniert, denn I kann die für B bestimmten Nachrichten somit nicht mehr entschlüsseln.

Literaturverzeichnis

[All86] Allender, Eric: The Complexity of Sparse Sets in P. In: Selman, Alan L. (Hrsg.): *Structure in Complexity Theory, Proceedings of the Conference hold at the University of California, Berkeley, California, June 2-5, 1986* Bd. 223, Springer, 1986 (Lecture Notes in Computer Science). – ISBN 3–540–16486–3, S. 1–11 42

[And01] Anderson, Ross J.: *Security Engineering — A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001 1

[Apt99] Apt, Krzysztof R.: The Rough Guide to Constraint Propagation. In: Jaffar, Joxan (Hrsg.): *CP* Bd. 1713, Springer, 1999 (Lecture Notes in Computer Science). – ISBN 3–540–66626–5, S. 1–23 16

[AS01] Ahn, Gail-Joon ; Shin, Michael E.: Role-Based Authorization Constraints Specification Using Object Constraint Language. In: *WETICE*, 2001, 157–162 64

[BB98] Bacchus, Fahiem ; Beek, Peter van: On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In: Mostow, Jack (Hrsg.) ; Rich, Charles (Hrsg.): *AAAI/IAAI*, AAAI Press / The MIT Press, 1998. – ISBN 0–262–51098–7, S. 310–318 7

[BB01] Bella, Giampaolo ; Bistarelli, Stefano: Soft Constraints for Security Protocol Analysis: Confidentiality. In: Ramakrishnan, I. V. (Hrsg.): *PADL* Bd. 1990, Springer, 2001 (Lecture Notes in Computer Science). – ISBN 3–540–41768–0, 108–122 46, 50

[BB04] Bella, Giampaolo ; Bistarelli, Stefano: Soft Constraint Programming to Analysing Security Protocols. In: *TPLP* 4 (2004), Nr. 5-6, S. 545–572 46, 50

[BB05] Bella, Giampaolo ; Bistarelli, Stefano: Information Assurance for security protocols. In: *Computers & Security* 24 (2005), Nr. 4, 322–333. http://dx.doi.org/10.1016/j.cose.2004.10.004 46, 50

[BFO04] Bistarelli, Stefano ; Foley, Simon N. ; O'Sullivan, Barry: Detecting and Eliminating the Cascade Vulnerability Problem from Multilevel Security Networks Using Soft Constraints. In: McGuinness, Deborah L. (Hrsg.)

; Ferguson, George (Hrsg.): *AAAI*, AAAI Press / The MIT Press, 2004. – ISBN 0–262–51183–5, S. 808–813 51

[Bis04] Bistarelli, Stefano: *Lecture Notes in Computer Science*. Bd. 2962: *Semirings for Soft Constraint Solving and Programming*. Springer, 2004. – ISBN 3–540–21181–0 20, 24

[BL73] Bell, David E. ; LaPadula, Len J.: Secure Computer Systems: Mathematical Foundations and Model / The MITRE Corp. 1973 (M74-244). – Forschungsbericht. – Bedford MA 50

[BM03] Brito, Ismel ; Meseguer, Pedro: Distributed Forward Checking. In: Rossi, Francesca (Hrsg.): *ICCP: International Conference on Constraint Programming (CP), LNCS 2833* Bd. 2833, Springer, 2003 (Lecture Notes in Computer Science). – ISBN 3–540–20202–1, S. 801–806 60, 62

[BS03] Barker, Steve ; Stuckey, Peter J.: Flexible access control policy specification with constraint logic programming. In: *ACM Trans. Inf. Syst. Secur* 6 (2003), Nr. 4, 501–546. http://doi.acm.org/10.1145/950194 64

[Bur06] Burgard, Dominique M.: *Lösung von Constraintproblemen für sichere Anfrageauswertungen mit Inferenzkontrolle*, Universität Dortmund, Diplomarbeit, 2006 vii, viii, 75

[CMAFE03] Corin, Ricardo ; Malladi, Sreekanth ; Alves-Foss, Jim ; Etalle, Sandro: Guess what? Here is a new tool that finds some new guessing attacks (Extended Abstract). In: Gorrieri, R. (Hrsg.) ; Lucchi, R. (Hrsg.): *IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*. Warsaw, Poland : Dipartamento di Scienze dell'Informazione Universita di Bologna, Italy, April 2003, 62–71 65

[CWJ04] Chen, Shiping ; Wijesekera, Duminda ; Jajodia, Sushil: Incorporating Dynamic Constraints in the Flexible Authorization Framework. In: Samarati, Pierangela (Hrsg.) ; Ryan, Peter Y. A. (Hrsg.) ; Gollmann, Dieter (Hrsg.) ; Molva, Refik (Hrsg.): *ESORICS* Bd. 3193, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–22987–6, S. 1–16 64

[Dec92] Dechter, Rina: Constraint Networks. In: Shapiro, Stuart C. (Hrsg.): *Encyclopedia of Artificial Intelligence* Bd. 1, Addison-Wesley Publishing Company, 1992, S. 276–285. – Second Edition 5

[Eck06] Eckert, Claudia: *IT-Sicherheit. Konzepte – Verfahren – Protokolle*. vierte. München : Oldenbourg, 2006. – ISBN 3–486–57851–0 1

[FA97] Frühwirth, Thom ; Abdennadher, Slim: *Constraint-Programmierung*. Berlin : Springer, 1997 71

[FMW01] Freuder, Eugene C. ; Minca, Marius ; Wallace, Richard J.: Privacy/Efficiency Tradeoffs in Distributed Meeting Scheduling by Constraint-Based Agents. In: *Proc. Workshop on Distributed Constraint Reasoning* (2001). ISBN 1–55860–777–3 39, 43

[Fre78] Freuder, Eugene C.: Synthesizing Constraint Expressions. In: *Communications of the ACM* 21 (1978), Nr. 11, S. 958–966. – and M.I.T., A.I.MEMO 370, Cambridge, 1976 9

[Fre82] Freuder, Eugene C.: A Sufficient Condition for Backtrack-Free Search. In: *Journal of the ACM* 29 (1982), Nr. 1, S. 24–32 8, 9

[FW92] Freuder, Eugene C. ; Wallace, Richard J.: Partial constraint satisfaction. In: *Artificial Intelligence* 58 (1992), S. 21–70 19, 20

[HE80] Haralick, Robert M. ; Elliott, Gordon L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems. In: *Artifcial Intelligence* 14 (1980), Nr. 3, S. 263–313 14

[HG05] Hoh, Baik ; Gruteser, Marco: Protecting location privacy through path confusion. In: *First International Conf. on Security and Privacy for Emerging Areas in Communications Networks*. N.Y. : IEEE Comp. Society, September 2005, S. 194–205 65

[JSSS01] Jajodia, Sushil ; Samarati, Pierangela ; Sapino, Maria L. ; Subrahmanian, V. S.: Flexible support for multiple access control policies. In: *ACM Transactions on Database Systems* 26 (2001), Nr. 2, 214–260. http://portal.acm.org/citation.cfm?id=383891.383894 64

[Kle89] Kleer, Johan de: A Comparison of ATMS and CSP Techniques. In: Sridharan, N. S. (Hrsg.): *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Detroit, MI, USA : Morgan Kaufmann, August 1989. – ISBN 1–55860–094–9, S. 290–296 17

[Low95] Lowe, Gavin: An Attack on the Needham-Schroeder Public Key Authentication Protocol. In: *Information Processing Letters* 56 (1995), November, Nr. 3, S. 131–136 77

[LS77] Lipton, Richard J. ; Snyder, Lawrence: A Linear Time Algorithm for Deciding Subject Security. In: *Journal of the ACM* 24 (1977), Juli, Nr. 3, S. 455–464 75

[Mac87] Mackworth, Alan K.: Constraint Satisfaction. In: Shapiro, Stuart C. (Hrsg.): *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, 1987, S. 205–211 5

[MGLM04] Moodahi, Ilanit ; Gudes, Ehud ; Lavee, Oz ; Meisels, Amnon: A Secure Workflow Model Based on Distributed Constrained Role and Task Assignment for the Internet. In: Lopez, Javier (Hrsg.) ; Qing, Sihan (Hrsg.) ; Okamoto, Eiji (Hrsg.): *ICICS* Bd. 3269, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–23563–9, 171–186 54, 58

[MGM05] Moodahi, Ilanit ; Gudes, Ehud ; Meisels, Amnon: A Three Tier Architecture for Dynamic User/Role/Task Assignment – Design and Theoretical Issues. In: *N.N.*, 2005, S. 0–0 54, 60

[MJPL92] Minton, Steven ; Johnston, Mark D. ; Philips, Andrew B. ; Laird, Philip: Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. In: *Artifcial Intelligence* 58 (1992), Nr. 1-3, S. 161–205 13

[Nig06] Niggemeier, Matthias: *Lösung von Constraintproblemen für ein sicheres Workflow Management System*, Universität Dortmund, Diplomarbeit, 2006 vii, viii, 55

[NS78] Needham, Roger M. ; Schroeder, Michael D.: Using Encryption for Authentication in Large Networks of Computers. In: *Commun. ACM* 21 (1978), Nr. 12, S. 993–999 77

[RN03] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. second. Englewood Cliffs, New Jersey : Prentice Hall, 2003. – ISBN 0–137–90395–2 18

[RPD90] Rossi, Francesca ; Petrie, Charles J. ; Dhar, Vasant: On the Equivalence of Constraint Satisfaction Problems. In: Aiello, Luigia C. (Hrsg.): *ECAI*, Pitman, 1990. – ISBN 0–273–08822–X, S. 550–556 9

[SAFH05] Swart, Garret ; Aziz, Benjamin ; Foley, Simon N. ; Herbert, John: Trading Off Security in a Service Oriented Architecture. In: Jajodia, Sushil (Hrsg.) ; Wijesekera, Duminda (Hrsg.): *DBSec* Bd. 3654, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–28138–X, 295–309 65

[Sch96] Schneier, Bruce: *Applied cryptography: protocols, algorithms, and source code in C*. second. New York : John Wiley & Sons, 1996. – ISBN 0–471–12845–7 1

[SS77] Stallman, Richard M. ; Sussman, Gerald J.: Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. In: *Artificial Intelligence* 9 (1977), Nr. 2, S. 135–196 15

[Tsa93] Tsang, Edwad: *Foundations of Constraint Satisfaction*. New York (NY), USA : Academic Press, 1993 (Computation in Cognitive Science) 6

[Wal75] Waltz, David: Understanding lines drawings of scenes with shadows. In: Winston, Patrick H. (Hrsg.): *The Psychology of Computer Vision*. McGraw-Hill, 1975 (Computer Science Series), S. 19–91 16

[WF02] Wallace, Richard J. ; Freuder, Eugene C.: Constraint-based Multi-agent Meeting Scheduling: Effects of Agent Heterogeneity on Performance and Privacy Loss. In: *Proceedings Workshop on Distributed Constraint Reasoning* (2002), S. 176–182 39, 44

[WFM04] Wallace, Richard J. ; Freuder, Eugene C. ; Minca, Marius: Possibilistic Reasoning and Privacy/Efficiency Tradeoffs in Multi-agent Systems. In: Monroy, Raul (Hrsg.) ; Arroyo-Figueroa, Gustavo (Hrsg.) ; Sucar, Luis E. (Hrsg.) ; Azuela, Juan Humberto S. (Hrsg.): *MICAI* Bd. 2972, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–21459–3, S. 380–389 39, 46

[Yok00] Yokoo, Makoto: *Distributed Constraint Satisfaction*. Berlin : Springer, 2000 xiii, 5, 9, 13, 17, 18, 20, 27, 28, 30, 32, 33, 34, 37

[YSH02] Yokoo, Makoto ; Suzuki, Koutarou ; Hirayama, Katsutoshi: Secure Distributed Constraint Satisfaction: Reaching Agreement without Revealing Private Information. In: Hentenryck, Pascal V. (Hrsg.): *Constraint Processing* Bd. 2470, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–44120–4, 387–401 27, 63