Universität Hannover
Institut für Allgemeine Nachrichtentechnik
Prof. Dr.-Ing. H.-P. Kuchenbecker
Prof. Dr.-Ing. K. Jobmann

Erkennung wiederkehrender Fehlerszenarien in Kommunikationsnetzen durch Data Mining Algorithmen

Studienarbeit

Peter Tondl

29. Februar 2000

Erstprüfer	:	Prof. Dr.-Ing. K. Jobmann
Zweitprüfer	:	Prof. Dr.-Ing. H.-P. Kuchenbecker
Betreuer	:	Dipl.-Ing. Klaus-Dieter Tuchs

Inhaltsverzeichnis

Kapitel 1

Einleitung

Betreiber von Kommunikationsnetzen unterhalten landesweite Netze, deren Betriebsbereitschaft ständig überwacht und gewährleistet sein muss. Zur Bewältigung dieser Aufgabe kommen leistungsfähige Management-Systeme zum Einsatz, deren bereitgestellte Funktionen einen sicheren und wirtschaftlichen Betrieb der Netze ermöglichen.

Eine Netzmanagement-Funktion ist das Fault-Management. Aufgabe des Fault-Management ist es, im Fehlerfall die Ursache der Störung zu ermitteln und geeignete Maßnahmen zu deren Beseitigung einzuleiten. Eine große Bedeutung kommt in diesem Zusammenhang den Operateuren in den zuständigen Netzmanagement-Zentralen zu. Um Störungen im Netz zu erkennen, werden von den gestörten Elementen des Netzes an die Zentralen Alarmmeldungen übermittelt. Sind von einem Fehler mehrere Netzelemente gleichzeitig betroffen, beispielsweise durch den Ausfall einer Richtfunk-Übertragungsstrecke, kommt es zu einer Akkumulation von Fehlermeldungen. Dieser Effekt führt dazu, dass es dem Operateur auf Grund der Vielzahl gemeldeter Fehler erschwert wird, die eigentliche Ursache der Störung zu erkennen.

Für die Lösung dieses Problems kommen im Fault-Management Alarmkorrelatoren zum Einsatz. Diese fassen anhand gegebener Regeln Einzelalarme zu Alarmgruppen zusammen und ermöglichen so die Verringerung des vom Operateur zu verarbeitenden Alarmaufkommens.

Die der Regelerstellung zugrunde liegenden Muster, können zur Zeit nur manuell gefunden werden. Bedingt durch die Komplexität und Menge des zu verarbeitenden Alarmaufkommens, gestaltet sich dieser Vorgang sehr zeit- und damit kostenintensiv. Zudem unterliegen derartige Muster durch Hinzukommen neuer oder Umstrukturierung bestehender Netzkomponenten ständigen Veränderungen. Dem automatisierten, selbstständigen Finden von Mustern wird daher in Zukunft wachsende Bedeutung zukommen.

Die vorliegende Studienarbeit präsentiert eine mit Hilfe fortgeschrittener Techniken wie Data Mining entwickelte Lösung für das Problem der Musterfindung. Am Beispiel eines GSM-Mobilfunknetzes wird gezeigt, wie eine gegebene Datenbasis an Alarmmeldungen in sinnvoller Weise unterteilt und analysiert werden kann.

Die für das Verständnis der gewählten Lösung notwendigen Grundlagen aus dem Bereich Kommunikationsnetze werden im Kapitel 2 aufgezeigt. Kapitel 3 gibt einen kurzen Einblick in das Management eines Kommunikationsnetzes und erläutert die wichtigsten in dieser Arbeit gebrauchten Begriffe am Beispiel eines GSM-Mobilfunknetzes. In Kapitel 4 erfolgt eine umfangreiche Einführung in die Konzepte des Data Mining, da dieses Themengebiet für die Lösung der in dieser Arbeit behandelten Aufgabe fundamentale Bedeutung besitzt. Kapitel 5 beschäftigt sich mit der Umsetzung der gestellten Aufgabe und erläutert ausführlich die zugrunde liegenden Gedanken und Konzepte. Kapitel 6 fasst die wesentlichen Aspekte dieser Arbeit zusammen und gibt einen Ausblick über mögliche weiterführende Projekte.

Kapitel 2

Kommunikationsnetze

2.1 Grundsätzliche Betrachtungen

Kommunikation gewinnt für die wirtschaftliche und gesellschaftliche Entwicklung moderner Industriestaaten eine immer größere Bedeutung. Information ist neben Kapital und Arbeit zu einem zentralen Produktionsfaktor geworden. Der Erhaltung und Steigerung der Leistungsfähigkeit von Kommunikationsanlagen kommt daher fundamentale Bedeutung zu. Ziel dieser Studie ist es, ein Hilfsmittel für die Unterstützung des Fault-Managements (nähere Erläuterungen dazu finden sich im Kapitel 3) eines Kommunikationsnetzes zu entwickeln. Das für das Verständnis der gewählten Lösung notwendige Grundlagenwissen im Bereich Kommunikationsnetze soll in diesem Kapitel am Beispiel eines GSM-Mobilfunknetzes aufgezeigt werden.

Die Ausführungen der folgenden Abschnitte stützen sich auf [Sieg99].

2.2 Grundlegender Aufbau von GSM-Netzen

Mobilfunknetze besitzen eine regelmäßige Struktur. Jede *Basisstation* formt eine oder mehrere Zellen. Durch Überlappung in den Randbereichen werden große, zusammenhängende Funknetze gebildet. Den mobilen Endgeräten steht innerhalb jeder Zelle eine begrenzte Anzahl an Kommunikationskanälen zur Verfügung. Aktuelle Standards dieser zellularen Netze sind das in dieser Arbeit nicht näher behandelte *Digital Enhanced Cordless Telecommunication* (DECT), sowie das *Global System for Mobile Communication* (GSM). Ein GSM-Mobilfunknetz gliedert sich in mehrere Teilsysteme, denen spezielle Aufgaben zugeordnet sind.

2.2.1 Mobile Station

Der Rolle des Endgerätes im Festnetz entspricht die *Mobile Station* (MS) im Mobilfunknetz. Dabei handelt es sich zumeist um tragbare oder in Fahrzeugen fest eingebaute Telefone, Interfacekarten in TK-Anlagen und Notebooks oder kombinierte Geräte zur Fax-, Daten- und Sprachübertragung („Communicator“).

2.2.2 Base Station Sub-System

Die zum *Base Station Sub-System* (BSS) gehörende *Base Transceiver Station* (BTS) arbeitet im Frequenzbereich von 900 MHz bzw. 1800 MHz (Europa, Asien) sowie 1900 MHz (Nordamerika). Die Kanalanordnung geschieht im Frequenz- und Zeitvielfach. Eine bestimmte Anzahl von BTS wird von einem *Base Station Controller* (BSC) verwaltet, welcher seinerseits zu einer *Mobile-Services Switching Center* (MSC) genannten Vermittlungsstelle gehört.

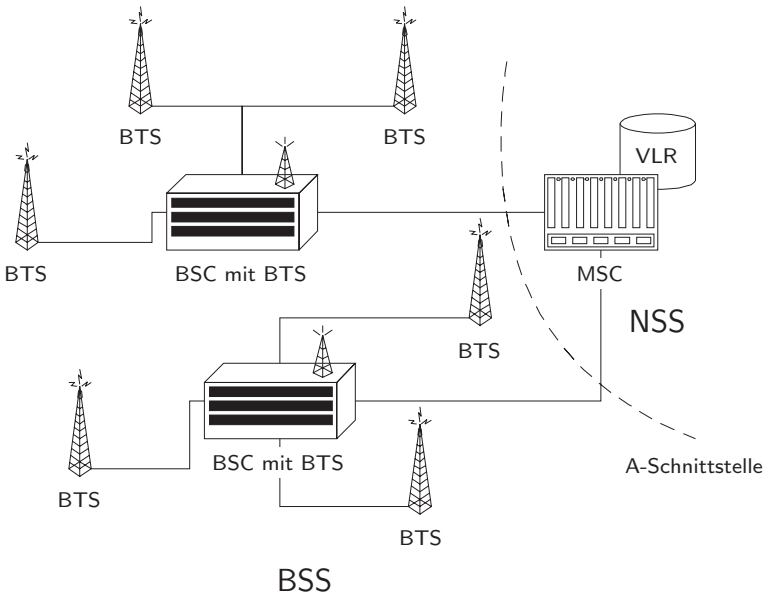

Abbildung 2.1: Aufbau eines GSM-Netzes ohne zentrale Einrichtungen

2.2.3 Network and Switching Sub-System

Durch das *Network and Switching Sub-System* (NSS) werden die Nutzkanäle in benötigter Weise zusammengeschaltet. Es enthält Datenbanken für die Mobilitäts- und Sicherheitsverwaltung der Teilnehmer und stellt diesen spezielle Leistungsmerkmale des Netzes zur Verfügung. Als Grundlage für die Erstellung der Fernmelderechnung dienen die im NSS erzeugten Gebührensätze.

2.2.4 Operation Sub-System

Das *Operation Sub-System* (OSS) überwacht den Betrieb und die Funktion der Netzelemente und sorgt im Fehlerfall für die Einleitung von Instandhaltungsmaßnahmen.

Das zum OSS gehörende *Operation and Maintenance Center* (OMC) dient dem Betrieb und der Wartung der GSM-Systeme durch den Netzbetreiber. Diese Funktion kann für mehrere MSC mit ihren zugehörigen BSS bereitgestellt werden. Im OMC wird der Netzzustand und die Auslastung der einzelnen Systeme beobachtet und bei Bedarf steuernd eingegriffen.

Kapitel 3

Management

3.1 Telecommunication Management Network

Das Management von Mobilfunknetzen basiert auf den Festlegungen des *Telecommunication Management Network* (TMN), einem ganzheitlichen Netz- und Dienstmanagement für alle am Telekommunikations-Prozess beteiligten Instanzen.

Zu den wichtigsten Management-Aufgaben eines Netzes gehören:

Das Erkennen, Lokalisieren und Beheben von Fehlern durch das *Fault Management*. Die Durchführung von Routineuntersuchungen und die Alarmbehandlung fällt ebenso in diesen Bereich.

Das Verwalten von Netzkonfigurationen, Einbinden von Erweiterungen und Neuerungen sowie Rekonfigurationen durch das *Configuration Management*.

Das *Accounting Management* zur Teilnehmerverwaltung in Bezug auf Gebührenerfassung und -verrechnung, sowie der Erstellung von Teilnehmerstatistiken.

Die Überwachung der Dienstgüte des Gesamtsystems durch das *Performance Management* bezüglich Fehlerrate und Leistungsfähigkeit.

Die Verwaltung der Authentifizierungsdaten und der Zugangskontrolle durch das *Security Management* sowohl für den Teilnehmer als auch für das Managementsystem.

Die Ausführungen dieses Kapitels stützen sich im Wesentlichen auf [Sel94] sowie auf [Sieg99].

3.1.1 Managed Objects

Ein *Managed Object* stellt eine abstrakte Sicht einer realen Ressource für Management-Zwecke dar. Es wird durch seine Attribute, die auf ihm ausführbaren Operationen und durch die Beziehungen zu anderen Managed Objects beschrieben.

In Bezug auf Managed Objects sind drei weitere Begriffe von Bedeutung:

> *Einkapselung* beschreibt die Beziehung zwischen einem Managed Object und seinen Attributen sowie seinem Verhalten. Attribute und Verhalten werden nur durch die am Managed Object ausführbaren Operationen und den zurückgegebenen Meldungen erfasst.
>
> *Vererbung* ist ein Mechanismus der Attribute, Meldungen, Verhalten und Operationen von Managed Objects einer Klasse auf die einer Unterklasse überträgt.
>
> Ist ein Managed Object von der Existenz eines übergeordneten abhängig, wird von *Beinhaltung* gesprochen.

3.1.2 Manager und Agent

Ein *Manager* veranlasst Management-Operationen und empfängt Meldungen, die Management-Information enthalten. Ein *Agent* führt die vom Manager beauftragten Management-Operationen an den Managed Objects aus und leitet von diesen kommende Meldungen an den Manager weiter. Anfragen bezüglich der Managed Objects gehen damit grundsätzlich über den Agent. Vorteil dieses Verfahrens ist die Entlastung des Managers sowie die Möglichkeit zur Bildung spezialisierter und optimierter Management-Einheiten.

Die Aufgaben des Agent betreffen:

- die Darstellung der durch die Managed Objects dargestellten Ressourcen
- die Weitergabe nur vom Manager benötigter Informationen (Filterung)
- das Ausführen von Management-Operationen und die Rückgabe der Ergebnisse an den Manager (Vermittlung)

Manager und Agent sind in der Lage ihre Rollen zu tauschen, wodurch die Bildung verteilter, hierarchischer Manager-zu-Manager-Strukturen ermöglicht wird.

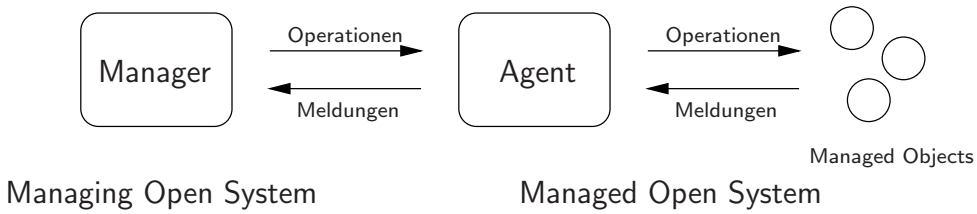

Abbildung 3.1: Manager und Agent

3.1.3 Management Information Base

Ein Satz von Managed Objects innerhalb eines Systems bildet zusammen mit ihren Attributen die *Management Information Base* (MIB). Die MIB stellt die in ihr gespeicherten Informationen über die Managed Objects in Form einer *Management Information Tree* (MIT) genannten hierarchischen Baumstruktur dar. Der Manager sendet Anfragen und Operationen an den Agent. Dieser sucht ausgehend von der Wurzel über die Arten von Objektklassen das geforderte Objekt heraus und sendet die gefunden und vom Manager benötigten Informationen (Filterfunktion des Agent) an diesen zurück.

3.2 Fault Management in GSM-Netzen

Zur Realisierung eines weitgehend störungsfreien Netzbetriebes gehört die Überwachung und Behandlung auftretender Fehler. Defekte Netzelemente müssen vom System erkannt und mittels geeigneter Alarme signalisiert werden. Da nur die Netzelemente der obersten Hierarchie über dedizierte Meldungskanäle verfügen, alle anderen Netzelemente aber über Kommunikationskanäle signalisieren, kommt es im praktischen Betrieb zu einer Reihe von Problemen.

3.2.1 Fault Management im Base Station Sub-System

Das Fault Management im Base Station Sub-System ist gekennzeichnet durch ein hohes tägliches Alarmaufkommen. Neben der generellen Belastung des Gesamtsystems kommt erschwerend hinzu, dass die Komponenten unterschiedlicher Hersteller unterschiedliche Überwachungseinrichtungen besitzen. Die Meldungen werden zudem an verschiedene, herstellerabhängige OMC versandt. Allgemein lässt sich sagen, dass bei der Generierung von Alarmmeldungen durch die Netzelemente ein hohes Maß an Redundanz erzeugt wird. Mehrfachfehler führen zu *Alarmfluten*, die eine Überlastung der Meldungsnetze zur Folge haben können. Eine weitere negative Eigenschaft von Alarmfluten besteht in der Möglichkeit der Verdeckung einzelner – möglicherweise bedeutender – Alarme durch eine Vielzahl unwichtiger Meldungen. Aus diesen

Gründen ist eine problembezogene Reduzierung des Alarmaufkommens notwendig.

3.2.2 Ziel der Alarmreduktion

Ziel der Alarmreduktion ist die Steigerung der Effizienz der Netzüberwachung durch Reduktion der Alarme auf das notwendige Minimum. Im Idealfall kommt es dabei zu folgendem Effekt: Der Operator im OMC wird vor Informationsüberflutungen geschützt wodurch eine zielgerichtete und schnelle Einleitung von Instandhaltungsmaßnahmen gewährleistet werden kann.

Die Hauptaufgabe der Alarmreduktion besteht somit in dem Eliminieren von redundanten Alarminformationen.

3.2.3 Methoden zur Alarmreduktion

An Methoden zur Realisierung einer praxisgerechten Alarmreduktion kommen folgende Ansätze in Frage:

- Netzelemente und BSS-Bereiche sind zu optimieren
- Filtern von redundanten Alarmen
- Zurückhalten „unwichtiger“ Alarminformationen
- Alarmreduktion durch Alarmkorrelation

Häufig hat eine Fehlerursache im Netz eine Vielzahl von Symptomalarmen im OMC zur Folge. Ein besonders vielversprechender Ansatz ist daher die Alarmreduktion durch Alarmkorrelation. Mit Hilfe gefundener Alarmmuster können Korrelations-Regeln aufgestellt werden, die zu einer Reduktion von Alarmen im OMC beitragen. Dem Finden solcher Muster ist der Hauptteil dieser Arbeit gewidmet (Kapitel 5).

Kapitel 4

Data Mining

Ziel des folgenden Kapitels ist es, den Leser mit den grundsätzlichen Inhalten und Konzepten des noch jungen Forschungsgebietes Data Mining vertraut zu machen. Die Themenbreite dieses Kapitels geht zum Teil über das für die Realisierung der eigentlichen Aufgabe dieser Arbeit benötigte Wissen hinaus. Es wurde dennoch Wert auf eine umfassende Darstellung gelegt, um die theoretischen Grundlagen für weiterführende Studien zu liefern.

Die Ausführungen dieses Kapitels stützen sich – soweit nicht anders angegeben – auf [HS94].

4.1 Vorbetrachtungen

Eine Datenbank ist ein zuverlässiger Aufbewahrungsort für Informationen. Ein vorrangiger Zweck eines solchen Aufbewahrungsortes ist die effiziente Bereitstellung von Daten. Dabei ist die bereitgestellte Information nicht notwendigerweise ein getreues Abbild eines gespeicherten Datensatzes, sondern vielmehr eine mögliche Schlussfolgerung aus diesem. Dabei handelt es sich bereits um Informationen auf einer höheren Ebene, um Wissen: Verallgemeinerungen über die Eigenschaften von Objekten, die nicht explizit in der Datenbank enthalten sind. An diesem Punkt setzen Methoden zur Wissensgewinnung wie *Data Mining* an. Ihr Ziel ist es, Abhängigkeiten und Regelmäßigkeiten zwischen den Datensätzen zu finden und in Regeln zu formulieren.

Aus logischer Perspektive werden dabei zwei verschiedene Techniken unterschieden:

> *Deduktion* ist die Ableitung des Besonderen und Einzelnen aus dem Allgemeinen bzw. die Erkenntnis des Einzelfalls aus einem allgemeinen Gesetz. Die meisten Datenbank-Management-Systeme bieten einfache Operatoren für die Deduktion von Informationen an. Beispielsweise kann ein

Verbindungs-Operator, angewand auf zwei relationale Tabellen – von denen die erste die Beziehungen zwischen den Objekten A und B und die zweite die zwischen den Objekten B und C beschreibt – die Beziehungen zwischen den Objekten A und C liefern. Deduktion ist im Data Mining-Kontext nicht von Bedeutung und wird daher in dieser Studie nicht weiter betrachtet.

Induktion ist eine wissenschaftliche Methode, vom besonderen Einzelfall auf das Allgemeine, Gesetzmäßige zu schließen. Auf das obige Beispiel bezogen könnte es die Schlussfolgerung bedeuten, dass es eine Beziehung zwischen A und C gibt.

Der wichtigste Unterschied zwischen Deduktion und Induktion ist, dass Deduktion in korrekten Aussagen über die abgebildete reale Welt mündet, wenn die zu Grunde liegende Datenbasis richtig ist. Induktion kann hingegen zu Ergebnissen führen, die aufgrund der Datenbasis zwar begründbar aber nicht notwendigerweise in der realen Welt richtig sind. Der entscheidende Aspekt beim Induktions-Prozess ist daher die Selektion der plausibelsten Regeln, die durch die Datenbasis gestützt werden.

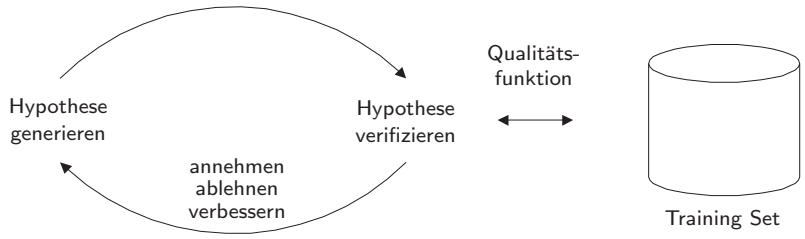

Abbildung 4.1: Induktions-Prozess mittels iterativer Suche

Bei der Ermittlung von Regelmäßigkeiten, die für jeden Data Mining-Prozess besondere Bedeutung besitzen, ergeben sich eine Reihe von Problemen: Aus einer großen Anzahl möglicher Beziehungen müssen die herausgefunden werden, die für die weitere Verwendung wichtig sind. Dabei bereitet die Tatsache Schwierigkeiten, dass in umfangreichen Datenbanken oft unklare, fehlende oder widersprüchliche Einträge gespeichert sind.

Einen direkten Nutzen – also einen Informationsgewinn ohne jede Vorverarbeitung – aus einer Datenbasis zu ziehen liegt jenseits menschlicher Möglichkeiten. Dies resultiert zum einen aus der Größe der üblicherweise vorhandenen Datenmenge und zum anderen aus der Eigenschaft von Datenbeständen unaufhörlich zu wachsen. Ziel ist es also, Datenbank-Management-Systeme so um induktive bzw. Data Mining-Fähigkeiten zu erweitern, dass sie zu einer Quelle verwertbaren Wissens werden.

4.2 Induktives Lernen

Menschen und andere intelligente Wesen („kognitive Systeme“) versuchen ihre Umwelt zu verstehen, indem sie Vereinfachungen dieser Umwelt – Modelle – benutzen. Die Bildung eines solchen Modells wird als *induktives Lernen* bezeichnet. Während der Lernphase beobachtet das kognitive System seine Umgebung und registriert Zusammenhänge zwischen Objekten und Ereignissen in dieser. Es teilt Objekte mit hinreichender Ähnlichkeit in Klassen ein und versucht Regeln zu konstruieren, nach denen das Verhalten der Angehörigen einer solchen Klasse vorausgesagt werden kann.

Zwei Lerntechniken sind dabei von besonderem Interesse. Beim *überwachten Lernen* definiert ein Lehrer die Klassen und unterstützt das kognitive System mit Beispielen für diese Klassen. Das System muss die gemeinsammen Eigenschaften der Klassenbeispiele herausfinden und in einer Klassenbeschreibung formulieren (Lernen aus Beispielen). Eine Klasse zusammen mit ihrer Beschreibung formt eine Regel:

wenn `(Beschreibung zutrifft)` *dann* `(Objekt gehört zur Klasse X)`

Diese wird benutzt, um weitere, unbekannte Objekte ihrer Klasse zuzuordnen. Beim *nichtüberwachten Lernen* muss das System die Klassen definieren, basierend auf den gemeinsammen Eigenschaften der Objekte (Lernen durch Beobachten).

4.3 Wissensgewinnung aus Datenbanken

Die Suche nach Beschreibungen wird *Data Mining* genannt, wenn es sich bei den zu Grunde liegenden Training Sets um Datenbanken[1] handelt. Diese sind im Allgemeinen sehr groß und enthalten Daten, die nicht generiert und gespeichert wurden um Lernprozesse zu unterstützen. Darüberhinaus repräsentieren die Datensätze nur einen geringen Teil aller möglichen Variationen. Prinzipbedingt ist das lernende System nicht in der Lage seine Umwelt zu beeinflussen, um „interessantere“ Daten zu erhalten. Die Größe der Datenbanken verteuert aber auch die Verifikation von aufgestellten Hypothesen (siehe Abbildung 4.1). Um diese Kosten zu verringern, kommen fortgeschrittene Datenbank-Techniken zum Einsatz. Statistische Methoden werden zur Behandlung verrauschter und unvollständiger Datensätze angewandt.

Zur Realisierung des induktiven Lernens müssen für Data Mining-Verfahren Beispielmengen zur Verfügung gestellt werden. Eine endliche Menge von Objekten, auf denen der Data Mining-Algorithmus arbeitet, wird dabei als *Training Set* bezeichnet. Die Extraktion von Regelmäßigkeiten aus dem Training Set heißt *Trainingspha-*

[1] Durch die Betrachtung von Daten als „Rohstoff“ erklärt sich der Begriff des Data Mining.

se des Data Mining-Verfahrens. Zur Überprüfung der Ergebnisse wird eine andere Menge von Objekten, das *Test Set*, verwendet. Die Überprüfung mit dem Test Set heißt daher *Testphase* des Data Mining-Verfahrens. Voraussetzung für das Test Set ist, dass die gewählten Objekte nicht in der Trainingsphase zum Einsatz kamen, also für das Regelwerk neu sind. Zeigen die gefundenen Regeln auch auf dem Test Set gute Ergebnisse, so kommt ihnen eine gewisse Allgemeingültigkeit innerhalb des Beispielbereichs zu.

Bei gegebenem Training Set und möglicherweise einigen benutzerdefinierten Klassen, ist ein Data Mining-System in der Lage Beschreibungen zu konstruieren. Einige dieser Beschreibungen werden korrekter sein als andere. Das bedeutet, dass diese Beschreibungen unbekannte Beispiele mit einer größeren Wahrscheinlichkeit ihrer zugehörigen Klasse zuordnen werden und damit ebenfalls einige – den Daten unterlegte Beziehungen – beschreiben. Mit der Definition einer Messmethode für die Qualität einer Beschreibung wird deren Konstruktion zum Such-Problem: Aus der Menge aller konstruierbaren Beschreibungen ist die beste herauszufinden. Für eine intensive Suche ist diese Menge aber bereits zu groß. Die Verwendung effizienter Such-Algorithmen ist daher unabdingbar. Die meisten Data Mining-Systeme wählen eine erste Beschreibung, deren Qualität in mehreren Iterationsschritten zunehmend verbessert wird. Diese Modifikationen sind Operationen auf den Beschreibungen. Die Menge der Beschreibungen zusammen mit den Operationen und der Qualitäts-Funktion wird *Such-Raum* genannt (search space).

Bei allen bisherigen Ausführungen wurde vorausgesetzt, dass die gesuchten Beschreibungen der Daten – und damit die Möglichkeit der Einteilung in Klassen – existieren. Dies ist bei der Benutzung einer Datenbank als Training Set aber nicht zwangsläufig der Fall und führt somit zu einer Reihe von Problemen: Die vom System beschaffbare Informationsmenge ist begrenzt. Darüberhinaus kann die verfügbare Information verfälscht oder unvollständig sein. Schließlich ist die Größe der Datenbank und ihr dynamisches Verhalten ein Problem.

Die Wahl einer angemessenen Form der Wissens-Repräsentation ist eine wichtig Entscheidung bei der Konstruktion eines Data Mining-Systems. Dabei treten eine Reihe von Forderungen auf: Die Art der Repräsentation muss hinreichend komplex sein, um Verbindungen zwischen Daten ausreichend genau zu beschreiben. Andererseits gilt es, das angebotene Wissen in Ausmaß und Struktur beschränkt zu halten, damit es für Menschen verständlich bleibt. Die optimale Lösung für eine bestimmte Anwendung ist somit ein Kompromiss zwischen diesen gegensätzlichen Forderungen.

4.4 Die Beschreibung von Klassen

Wie im Abschnitt 4.2 angedeutet, ist Lernen als Vorgang gleichbedeutend mit der Konstruktion von Regeln, basierend auf der Beobachtung der Umwelt. Handelt es sich bei dieser um eine Datenbank, wird von Data Mining gesprochen. Dabei modelliert die Datenbank die reale Umwelt, das heißt, jeder Statuswechsel innerhalb der Datenbank reflektiert einen Statuswechsel in der realen Welt. Der Lern-Algorithmus bildet somit ein Modell aus den Datensätzen. Für die Klassen bedeutet dies, dass der Algorithmus diejenigen Regeln bilden muss, die die Klassifikation der Datenbankobjekte gewährleisten. Regeln, die die Übergänge zwischen den Klassen betreffen, sollen aus den Statusübergängen der Datenbank gebildet werden.

Die Datenbank, genauer das Training Set S für den Lern-Algorithmus, repräsentiert Informationen über die Umwelt. Prinzipiell sind viele verschiedene Repräsentationsarten denkbar. In allen weiteren Ausführungen wird daher von relationalen Datenbanken mit SQL-Schnittstelle ausgegangen. Das heißt, Objekte der realen Welt werden durch Tupel in der Datenbank dargestellt. Weiter darf angenommen werden, dass diese Tupel nur die Eigenschaften dieser Objekte beschreiben und nicht Beziehungen zwischen ihnen. Die Repräsentation mehrerer Objekte durch ein einziges Tupel ist möglich, wenn die Unterschiede zwischen den Objekten als unwichtig erachtet oder nicht erfasst werden. Eine zweite Annahme betrifft die Anzahl der Tabellen in der Datenbank. Auf Grund praktischer Überlegungen wird diese zu Eins angenommen. Werte in der Tabelle dürfen Null sein oder ganz fehlen.

Definition 4.1 (Training Set)
Sei $A = \{A_1, \ldots, A_n\}$ ein Set von Attributen mit den Wertebereichen $B = \{B_1, \ldots, B_n\}$. Dann ist ein *Training Set* eine Tabelle über A. Ein *Beispiel* ist ein Tupel in einem Training Set. Das *Universum* U, wobei gilt $U = B_1 \times \cdots \times B_n$, ist die gesamte Relation über A, das heißt, jedes Training Set ist ein endliches Subset von U.

Abschließend sei angenommen, dass jeder Wertebereich B_i – und damit auch U – endlich ist.

Beispiel 4.1
Einfache Darstellung von Stoffen und ihren Eigenschaften als Training Set.

Stoff	Aggregatzustand bei Raumtemperatur	Leiter oder Nichtleiter
Kupfer	fest	Leiter
Quecksilber	flüssig	Leiter
Wasser	flüssig	Nichtleiter

Dabei umfasst die Attributmenge A das Set {Stoff, Aggregatzustand bei Raumtemperatur, Leiter oder Nichtleiter}, während die Wertebereiche B_i aus B_1 = {Kupfer, Quecksilber, Wasser}, B_2 = {fest, flüssig} sowie B_3 = {Leiter, Nichtleiter} bestehen.

Für eine Data Mining-Anwendung ist die Datenbank die Umwelt. Die Anwendung hat daher ein Modell der Datenbank zu bilden (siehe Abschnitt 4.2). Im Fall des überwachten Lernens erfordert dies die Definition einer oder mehrerer Klassen durch den Benutzer („externer Lehrer“). Dabei darf angenommen werden, dass in der Datenbank mindestens ein Attribut enthalten ist, das die Klasse eines Tupel definiert. Dieses Attribut wird als *vorhergesagtes Attribut* bezeichnet, alle anderen Attribute heißen *vorhersagende Attribute*. Eine Klasse wird demnach durch die Werte der vorhergesagten Attribute bestimmt.

Definition 4.2 (Klassen)
Eine Klasse K_i ist ein Subset des Training Set S, in der alle Objekte die Klassenbedingung $cond_i$ erfüllen.
Objekte, die die Klassenbedingung $cond_i$ erfüllen, werden *positive Beispiele* oder *Instanzen* der Klasse K_i genannt. Beispiele außerhalb des Subset heißen *negative Beispiele*.

Die Definition der Klassenbedingungen $cond_i$ durch den Benutzer teilt das Training Set S in Subsets – den Klassen K_i – ein. Aufgabe des Data Mining-Systems ist es, für diese Klassen Beschreibungen zu konstruieren.

Zur Definition der Klassen kommen verschiedene Techniken zum Einsatz. Zum einen ist es möglich, eine Klasse aus Termen vorhergesagter Attribute zu definieren. Auf Beispiel 4.1 bezogen könnte dies bedeuten, dass der Benutzer eine Klasse ‚Kupfer‘ mit der Bedingung ‚Stoffe = Kupfer‘ definiert, in der das Attribut ‚Stoffe‘ das vorhergesagte Attribut, die Attribute ‚Aggregatzustand bei Raumtemperatur‘ und ‚Leiter oder Nichtleiter‘ die vorhersagenden Attribute sind. Das bedeutet, jedes Beispiel im Training Set, dessen Wert für das Attribut Stoffe ‚Kupfer‘ ist, gehört zur Klasse ‚Kupfer‘.

Zum anderen können Bedingungen auch vorhersagende Attribute beinhalten. Objekte können zu einer Klasse ‚Positiv‘ gehören, wenn sie der Bedingung ‚Eingangswert > Ausgangswert‘ genügen. Dabei ist ‚Eingangswert‘ z.B. ein vorhersagendes Attribut, ‚Ausgangswert‘ ein vorhergesagtes Attribut.

Bei Vorliegen örtlich oder zeitlich getrennter Datenbanken lassen sich Klassen dazu benutzen, diese nicht direkt in den Datenbanken gespeicherte Information wiederzugeben. So können alle Tupel einer Datenbank am Standort A zur Klasse Ort_A gehören, alle Einträge einer Datenbank am Standort B zur Klasse Ort_B. Werden

aus einer Datenbank zu verschiedenen Zeiten i, j Daten entnommen, können diese jeweils den Klassen T_i und T_j zugeordnet werden.

Klassen können disjunkt oder konjunkt definiert werden. Ein hierarchischer Aufbau ist ebenso möglich.

Werden keine Klassen durch den Benutzer definiert, handelt es sich um nichtüberwachtes Lernen und das Data Mining-System muss seine Klassen selbst bestimmen. Ein möglicher Weg dorthin ist die Unterteilung des Training Set in Subsets genügend ähnlicher Objekte. Anschließend werden vom System Beschreibungen für die Subsets konstruiert. Zur Verdeutlichung der Komplexität dieses Vorgangs soll folgende Rechnung dienen:

Beispiel 4.2
Sei n die Zahl der Tupel einer Datenbank und m die Zahl der Subsets. Dann gilt für die Zahl $w(n, m)$ möglicher Einordnungen der n Tupel in m disjunkte, nichtleere Subsets [DO74]:

$$w(n,m) = \frac{1}{m!} \sum_{j=0}^{m} \binom{m}{j} (-1)^j (m-j)^n$$

Für die Gesamtzahl $G(n)$ aller Möglichkeiten gilt mit $1 \leq m \leq n$:

$$G(n) = \sum_{m=1}^{n} w(n,m) = \sum_{m=1}^{n} \frac{1}{m!} \sum_{j=0}^{m} \binom{m}{j} (-1)^j (m-j)^n$$

Somit ergeben sich beispielsweise für den Fall von $n = 1, \ldots, 12$ folgende Werte:

n	1	2	3	4	5	6
$G(n)$	1	2	5	15	52	203
n	7	8	9	10	11	12
$G(n)$	877	4.140	21.147	115.975	678.570	4.213.597

Für die Fälle einer Beschränkung der Anzahl der Subsets auf $m = 4$ und $m = 2$ folgt mit $n = 12$:

$$w(12,4) = 611.501 \qquad w(12,2) = 2.047$$

$w(n, m)$ und damit auch $G(n)$ ist eine Funktion mit exponentiellem Wachstum über n. Voraussetzung für die Anwendbarkeit des nichtüberwachten Lernens bei gegebenem, genügend kleinem n ist demnach eine möglichst kleine Anzahl von Subsets m. In jedem praktischen Fall dürfte $w(n, m)$ aber jenseits der Anwendbarkeit liegen.

Sind die Klassen definiert, konstruiert das System Beschreibungen für jede Klasse. Diese Beschreibungen sollten sich ausschließlich auf die vorhersagenden Attribute des Training Set beziehen. Im Idealfall genügen alle positiven Beispiele der Beschreibung der jeweiligen Klasse, während alle negativen Beispiele sicher ausgeschlossen werden.

Da im Training Set nur Attribute der Objekte und nicht Beziehungen zwischen diesen gespeichert sind, können Beschreibungen auch nur aus Bedingungen dieser Attribute bestehen. Die in Data Mining-Tools gewöhnlich verwendeten Beschreibungen sind ein Subset der Selektions-Bedingungen aus der relationalen Algebra.

Definition 4.3 (Beschreibung)
Sei $A = \{A_1, \ldots, A_n\}$ ein Set von vorhersagenden Attributen mit den Wertebereichen $B = \{B_1, \ldots, B_n\}$. Für eine *elementare Beschreibung* gilt dann: $(A_1 = c_1) \wedge \cdots \wedge (A_n = c_n)$, so dass:

1. $A_i \in A$ und $i \neq j \rightarrow A_i \neq A_j$
2. $c_i \in B_i$

Eine Beschreibung ist eine nichtleere Disjunktion von elementaren Beschreibungen. Eine Bedingung $A_i = c_i$ wird *Attributwert-Bedingung* genannt. Das Set aller möglichen Beschreibungen heißt *Beschreibungs-Raum* D_S (description space).

Beachtenswert ist, dass die Zahl möglicher Beschreibungen von der Größe der Wertebereiche und nicht direkt von der Größe des Training Set abhängt.

Definition 4.4 (Regel)
Eine Regel besteht aus einer Beschreibung D und einer Klasse K. Demnach bedeutet: ‚*wenn* `(D)` *dann* `(K)`‘, dass jedes Objekt, welches D genügt, zu K gehört.

Für eine Klasse ist nicht notwendigerweise nur eine einzige Beschreibung möglich. Im Regelfall werden eine Reihe von Beschreibungen konstruierbar sein, die die Klasse – bezogen auf das Training Set – richtig spezifizieren. Jedoch ist nicht sicher, dass alle diese Konstrukte im Hinblick auf noch unbekannte Beispiele mit der gleichen Korrektheit arbeiten werden.

4.5 Data Mining-Algorithmen

Bei der Suche nach Algorithmen für die Lösung von Problemen bleibt immer die Frage zu klären, ob es solche Algorithmen geben kann und wenn ja, mit welcher

Komplexität zu rechnen ist. Im Falle des überwachten Lernens ist es vorstellbar, alle gefundenen Beschreibungen einer Klasse zu betrachten und diejenige auszuwählen, die die besten Ergebnisse auf dem Training Set liefert (*vollständige Suche*).

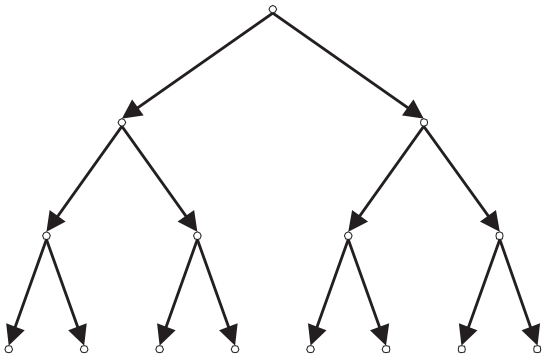

Abbildung 4.2: Vollständige Suche

Beim nichtüberwachten Lernen werden alle Subsets untersucht und in einem ersten Schritt für jedes Subset alle Beschreibungen auf alle Klassen in dem Set angewandt. Ausgewählt wird in einem zweiten Schritt das Set und die Beschreibungen, von denen das beste Ergebnis erwartet wird. Probleme wird bei dieser Lösung die schnell anwachsende Komplexität bereiten (siehe Beispiel 4.2). Weiter wird implizit angenommen, dass in dem Training Set S alle positiven Beispiele der jeweiligen Klasse vorhanden sind. Da S immer nur ein Teil des Universums U sein kann, ist die Korrektheit dieser Annahme nicht gewährleistet. Vielmehr ist die Wahrscheinlichkeit, mit der alle positiven Beispiele einer Klasse K in einem beliebig gewählten Training Set S vorhanden sind, eher gering. Dadurch kann nicht mit Sicherheit davon ausgegangen werden, dass eine für K gefundene Beschreibung diese Klasse – im Hinblick auf noch unbekannte positive Beispiele – richtig spezifiziert. Dennoch lassen sich Erkenntnisse über den Grad an Korrektheit gewinnen. Ziel ist es daher, die Wahrscheinlichkeit zu minimieren, dass eine gefundene Regel unbekannte Beispiele falschen Klassen zuordnet.

Ein Lern-Algorithmus wird *konsistent* genannt, wenn er sich auf seinem Training Set korrekt verhält. In der Praxis treten dabei hauptsächlich zwei Schwierigkeiten auf: Zum einen wird die vorhandene Datenbasis in vielen Fällen aus nur unvollständigen Informationen bestehen. Daher kann nicht davon ausgegangen werden, dass ein Lern-Algorithmus konsistent auf einer solchen Datenbasis ist. Zum anderen wird gefordert, dass die Beispiele der Datenbasis unabhängig voneinander sind. Wird jedoch die Tatsache berücksichtigt, dass in realen Datenbanken alle möglichen Arten von Datensätzen aus allen möglichen bekannten oder unbekannten Bereichen gespeichert sein können, lässt sich die Forderung nach Unabhängigkeit nicht mehr aufrecht erhalten.

Wie im Beispiel 4.2 gezeigt, kann die Suche nach Beschreibungen für Klassen, sowie

die Definition der Klassen selbst, sehr schnell zu einem nicht mehr beherrschbaren Problem werden. Der Einsatz effektiver Algorithmen ist daher eine unabdingbare Notwendigkeit für das Funktionieren von Data Mining-Prozessen.

4.5.1 Allgemeiner Ansatz

Es existieren verschiedene Algorithmen zur Durchquerung des Such-Raums. Die Grundidee ist die Verwendung einer Anfangsbeschreibung D_0, deren Qualität in iterativen Schritten so lange verbessert wird, bis ein vom Benutzer gesetzter Schwellwert erreicht ist.

Dabei kommen zwei Ansätze zum Tragen, die sich in der Wahl der Anfangsbeschreibung und in den Operationen zur Verbesserung der Qualität dieser Beschreibung unterscheiden:

> *Bottom-Up-Methode*: Die Anfangsbeschreibung deckt ausschließlich die positiven Beispiele der Zielklasse ab. Diese Beschreibung ist zwar korrekt, aber noch zu komplex. In weiteren Schritten erfolgt eine Vereinfachung mit dem Ziel der Formulierung einer allgemeingültigeren Regel. Die abschließende Beschreibung deckt bei minimaler Komplexität alle positiven Beispiele der Klasse ab, während negative Beispiele sicher ausgeschlossen werden.
>
> *Top-Down-Methode*: Beginnend mit einer Anfangsbeschreibung, die das gesamte Universum U abdeckt, werden die Beschreibungen durch Hinzufügen von Bedingungen an zusätzliche Attribute so weit ausdifferenziert, bis sie den Regeln des Training Set entsprechen.

Auf die Anfangsbeschreibung D_0 können verschiedene Operationen O_i angewandt werden. Die Anwendung einer Operation O_i führt zu einer neuen Beschreibung D_{1i}. Dieser Vorgang ist beliebig oft wiederholbar, so dass die Konstruktion eines Such-Graphen ermöglicht wird, in dem die Knoten Beschreibungen repräsentieren und die Äste Operationen auf die Wurzel des Teilgraphen darstellen. Dieser Graph ist nicht notwendigerweise ein Baum, da zu einem Knoten mehrere Wege führen können. Ziel ist es, eine Beschreibung ausreichender Qualität – einen *Zielknoten* – in minimaler Zeit zu finden. Die Konstruktion einer Regel kann daher als Such-Prozess bezeichnet werden. Die Art und Weise wie eine Sequenz von Operationen gefunden wird, die zu einem Zielknoten führt, wird *Suchstrategie* genannt. Zur Anwendung kommen im Wesentlichen zwei unterschiedliche Strategien:

> Bei der *endgültigen Suche* wird eine Operation ausgewählt und bis zum Erreichen eines Zielknotens angewandt. Dabei wird keine Sorge dafür getragen, zu einem späteren Zeitpunkt an einen bereits besuchten Knoten

des Graphen zurückkehren zu können. Es wird also immer nur ein möglicher Weg von der Wurzel zu einem Zielknoten des Graphen untersucht.

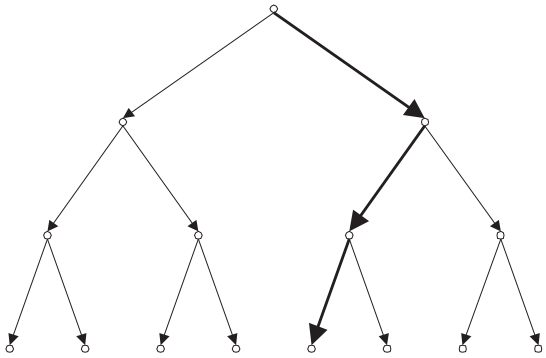

Abbildung 4.3: Endgültige Suche

Bei der *vorläufigen Suche* wird hingegen dafür gesorgt, zu einem späteren Zeitpunkt zu einem Knoten zurückkehren zu können um mit einer anderen Operation den Such-Vorgang fortzuführen. Dies kann entweder durch das Setzen eines Rückkehrpunktes geschehen oder durch das Speichern der gesamten bis dahin erforschten Struktur des Graphen.

Für diese Suchstrategien ist kein Vorwissen nötig, die zur Anwendung kommenden Operationen können willkürlich gewählt werden. Dabei gibt es im Fall der endgültigen Suche entweder nur eine mögliche anwendbare Operation oder die Reihenfolge der möglichen Operationen ist bedeutungslos. Im Fall der vorläufigen Suche kommen der *Tiefendurchlauf* bei Verwendung eines Rückkehrpunktes, sowie der *Breitendurchlauf* bei Speicherung der Graphenstruktur zum Einsatz.[2]

Ein Nachteil dieser Strategien sind die hohen Kosten, die durch die große Anzahl konstruierter Beschreibungen entstehen. Da die notwendige Berechnung der Qualitätsfunktion für jede Beschreibung sehr aufwändig ist, wird eine Reduktion der Anzahl konstruierbarer Beschreibungen nötig. Dies läßt sich beispielsweise durch die Wahl derjenigen Operationen O_i erreichen, die mit der höchsten Wahrscheinlichkeit den kürzesten Weg von der Wurzel des Such-Graphen zu einem Zielknoten beschreiben.

4.5.2 Heuristischer Ansatz

Um eine Beschreibung ausreichender Qualität in minimaler Zeit zu finden, muss der Suchaufwand durch sorgfältige Auswahl der zur Anwendung kommenden Operationen reduziert werden. Das bedeutet, dass das System zur Auswahl der Knoten

[2]Für eine genaue Erklärung dieser Durchlaufverfahren siehe z.B. [Par97]

auf dem kürzesten Weg zum Zielknoten des Such-Graphen zusätzliche Informationen benötigt. Da dieser Weg im Allgemeinen unbekannt ist, benötigt diese Strategie Informationen über den Suchbereich, also Bereichswissen oder *heuristisches Wissen*.

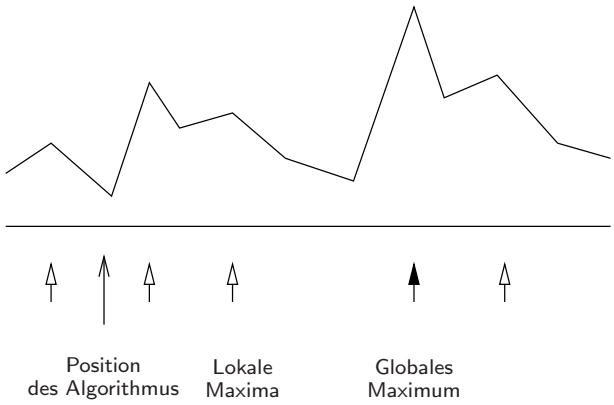

Abbildung 4.4: Qualitätsfunktion mit lokalen Maxima

Ein Verfahren mit heuristischem Ansatz ist das *Hill Climbing-Verfahren*. Die Idee eines solchen Verfahrens ist es, von dem augenblicklichen Zustand des Algorithmus aus den Schritt zu tun, bei dem der Wert einer gegebenen Gütefunktion maximal steigt. Problematisch ist dabei, dass der Algorithmus in einem lokalen Maximum enden kann. Eine Möglichkeit dies zu verhindern besteht in der Berechnung einer Anzahl aussichtsreicher Alternativen, von denen schließlich die beste gewählt wird. Dieses Verfahren wird *Beam-Search* genannt. Die Auswahlkriterien für die weiterzuverfolgenden Verfahren dürfen dabei verschiedene Sichtweisen und Eigenschaften berücksichtigen. So können beispielsweise verschiedene Bewertungsfunktionen verwendet werden, die sich in unterschiedlichen Situationen bewährt haben [Fer98].

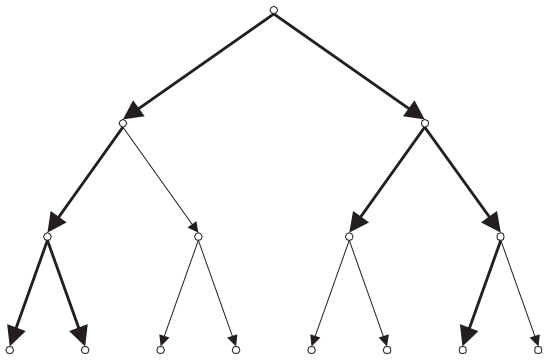

Abbildung 4.5: Beam-Search

Bereichswissen kann zur Unterstützung eines Such-Prozesses verwendet werden. Dieses Wissen bezieht sich immer auf einen ausgesuchten Bereich und muss vom Anwender selbstständig in den Prozess eingebracht werden. Dabei werden zwei Formen von Bereichswissen unterschieden:

Nicht relevante Attribute: Nicht alle Attribute eines Beispiels sind für eine gegebene Anforderung wichtig. Durch gezieltes Weglassen dieser Attribute reduziert sich die Zahl konstruierbarer Beschreibungen.

Attributbeziehungen: Für einige Anwendungen können bekannte Beziehungen zwischen Attributen bestehen, die ihren möglichen Wertebereich beschränken. Beispielsweise kann das Verhältnis der Länge L eines Rechtecks zu seiner Breite B immer mit der Bedingung $L \geq B$ definiert werden. Die Qualität einer Beschreibung verbessert sich somit durch das Hinzufügen einer bereits implizit vorausgesetzten Bedingung nicht.

Präsentiert das System eine Auswahl an Beschreibungen, möglicherweise zusammen mit dem berechneten oder geschätzten Wert einer zugehörigen Qualitätsfunktion, kann der Anwender eine oder mehrere dieser Beschreibungen für eine weitere Untersuchung durch das System auswählen. Somit kann Anwenderwissen, das nicht in Form von Bereichswissen zur Verfügung steht, in den Such-Prozess eingebracht werden.

Bereits entdeckte Regeln, Klassen und Beschreibungen lassen sich ebenfalls für die Unterstützung des Such-Prozesses nutzen. Klassen können in Hierarchien eingeordnet werden, ebenso ist die Konstruktion neuer Klassen mittels bekannter Klassen möglich. Werden neue Beispiele dem Training Set hinzugefügt, können Beschreibungen verfeinert und den neuen Gegebenheiten angepasst werden.

Alle bisherigen Ansätze haben den Nachteil geringer Leistungsfähigkeit bei schlechter Heuristik. Die Zeitdauer für das Finden einer Beschreibung ausreichender Qualität wird in vielen Fällen jenseits des Annehmbaren liegen. Daher ist die Notwendigkeit gegeben, alternative Ansätze zu diskutieren.

4.5.3 Genetischer Ansatz

Genetische Algorithmen behandeln nicht einzelne Datenobjekte, sondern agieren mit einer Sammlung von Objekten. Dies entspricht dem Verhalten der Evolution: Die Natur kümmert sich nicht um das Individuum, sondern um die Erhaltung der Art. Dazu wendet sie Strategien wie „nur die Besten überleben“ an, um aus der gesamten Art einzelne Individuen auszuwählen. Die Individuen der Natur entsprechen den Beschreibungen eines Data Mining-Algorithmus. Beim Genetischen Ansatz werden diese Beschreibungen als *Organismen* bezeichnet.

Diese Menge der Organismen wird wie in der Natur *Population* genannt, zwischen einzelnen Mitgliedern einer Population finden Wechselwirkungen statt: Mehrere Organismen können mittels *Vererbung* neue Organismen schaffen, sie können weiterleben oder mangels Eignung sterben.

Die Beschaffenheit eines Individuum wird in der Natur durch seine Gene bestimmt. Diese müssen bei Organismen modelliert werden. Die Qualität der Modulation entscheidet mit über die Leistungsfähigkeit des Algorithmus.

Genetische Algorithmen unterziehen eine Menge von Organismen einer Bewertungsfunktion. Organismen ausreichender Qualität überleben und bilden eine neue *Generation*. Es handelt sich somit um ein adaptives Verfahren, dass heißt, die Lösung nähert sich schrittweise der gestellten Aufgabe. Genetische Algorithmen finden dabei oft auch solche Lösungen, die bei herkömmlichen Verfahren nicht gefunden werden, da sie zu „exotisch“ sind.

Organismen werden als Symbolketten eines Alphabets dargestellt. Im Allgemeinen kommt das Binär-Alphabet mit dem Wertebereich $\{0, 1\}$ zur Anwendung. Jede Beschreibung wird als String fester Länge codiert. Diese Strings enthalten Substrings für jedes Attribut A_i. In diesen Substrings repräsentiert jede Position ein Element des zu A_i gehörenden Wertebereichs B_i.

Beispiel 4.3
Bei einem Training Set mit den Attributen $A_1 =$ Farbe und $A_2 =$ Form sowie den zugehörigen Wertebereichen $B_1 = \{$schwarz, weiß$\}$ und $B_2 = \{$Kreis, Rechteck$\}$ entspricht die Beschreibung ‚Farbe $\in \{$schwarz, weiß$\}$ $\wedge$ Form $\in \{$Kreis$\}$‘ beispielsweise dem String ‚1110‘.

Genetische Algorithmen nutzen nicht die Generalisierungs- und Spezialisierungs-Operationen der in Abschnitt 4.5.1 vorgestellten Verfahren, sondern folgende drei Operationen:

Bei der *Reproduktion* als einfachster Operation wird ein Organismus – basierend auf seiner *Fitness* – ausgewählt und in die nächste Generation kopiert. Ziel ist es, „guten“ Organismen die Weitergabe ihrer Gene in den folgenden Generationen zu ermöglichen.

Beim *Crossover* (Vererbung) werden Teile eines Strings durch Teile eines anderen Strings ersetzt. Beispielsweise können zwei Eltern-Strings einen neuen Tochter-String erzeugen.

Die dritte Operation ist die *Mutation*, bei der ein Gen eines in die folgende Generation kopierten Organismus zufällig oder gezielt verändert wird. Sinn der Mutation ist es, aus lokalen Maxima herauszukommen. Diese entstehen, da bei den Operationen ‚Reproduktion‘ und ‚Crossover‘ nur aus dem bestehenden Genpool geschöpft werden kann, neue Elemente aber nicht hinzukommen.

Genetische Algorithmen übertreffen traditionelle Lerntechniken, besonders wenn die zu lernenden Beschreibungen komplexer Natur sind. Dennoch leiden genetische Algorithmen unter zwei wichtigen Nachteilen: Erstens übertreffen sie traditionelle Techniken nur dann, wenn fast kein Bereichswissen gegeben ist. Obwohl dieses auch zur Unterstützung genetischer Algorithmen eingesetzt werden kann, ist der Nutzeffekt verglichen mit den traditionellen Techniken gering. Ein zweiter Nachteil ist die hohe Zahl an benötigten Evaluationen. Ein genetischer Algorithmus erfordert typischerweise 10 Generationen mit 50–100 Organismen um qualitativ ausreichende Lösungen zu finden [HS94].

4.6 Phasen eines Data Mining-Prozesses

Data Mining bezeichnet nicht eine einzelne Technik, sondern umfasst den gesamten Prozess von der Bereitstellung der Daten bis zu ihrer Anwendung. Dabei hängen die zum Einsatz kommenden Einzelschritte von der jeweiligen Situation ab. Viele Data Mining-Prozesse lassen sich dabei in folgende Phasen unterteilen [Pet97]:

In einer ersten *Planungsphase* wird die konkrete Aufgabenstellung festgelegt. Dabei sind mit Hilfe fachkundiger Personen Ziele und erwartete Ergebnisse des Prozesses zu formulieren.

Im zweiten Schritt, der *Vorbereitungsphase*, werden die vorhandenen Daten in eine für den Data Mining-Prozess günstige Form gebracht. Dazu gehört die Auswahl der für den Prozess verwendeten Informationen, die – wenn möglich – Einarbeitung von Hintergrundwissen sowie die Integration verschiedener Datenbestände. Wenn nötig werden in dieser Phase Formatanpassungen unterschiedlicher Datenbestände vorgenommen und falsche bzw. widersprüchliche Daten aus dem Bestand entfernt.

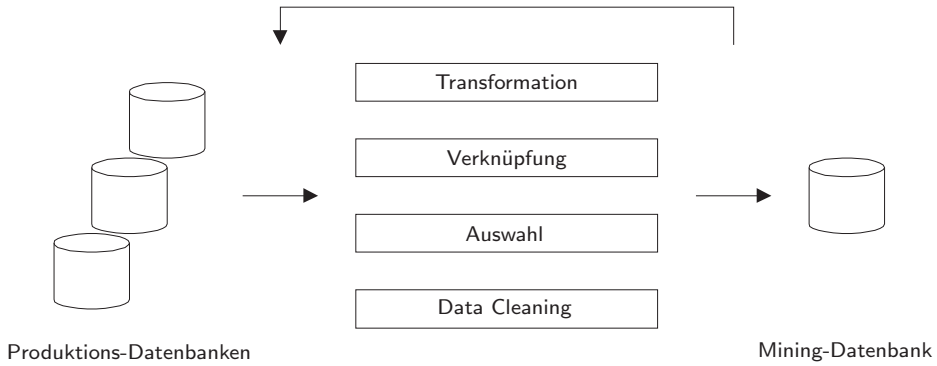

Abbildung 4.6: Vorbereitungsphase

Während der *Mining Phase* findet die eigentliche Suche nach interessanten Mustern statt. Unter Umständen kommt es am Ende dieser dritten Phase zu einem Rücksprung in die Vorbereitungsphase, wenn z.B. zusätzliche Daten in die Abarbeitung integriert werden müssen.

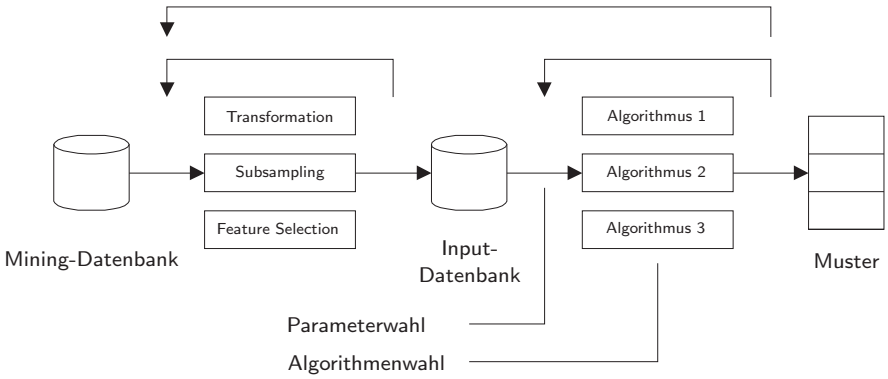

Abbildung 4.7: Miningphase

In der vierten und letzten Phase, der *Auswertungsphase*, erfolgt die Aufbereitung der Ergebnisse der Mining Phase in eine für den Benutzer lesbare Form.

4.7 Probleme und Lösungsansätze

Datenbanken werden aufgestellt und betrieben für Ziele jenseits des Data Mining. Daher ist die Repräsentation realer Objekte in der Datenbank auf die Bedürfnisse der sie nutzenden Applikationen abgestimmt. Aus diesem Grund sind Eigenschaften oder Attribute, die den Data Mining-Prozess unter Umständen vereinfachen können weder notwendigerweise vorhanden, noch können sie nachträglich durch den Prozess beschafft werden.

4.7.1 Unbekannte Werte vorhersagender Attribute

Im Fall des überwachten Lernens geht es darum, eine Beziehung zwischen den vorhersagenden Attributen und ihrer Klasse zu finden. Vorstellbar ist dies als ein Prozess, dessen Eingang durch die Werte dieser Attribute gebildet wird und dessen Ausgang die vorhergesagte Klasse ist.

Ein mögliches Problem besteht darin, dass nicht alle Werte der vorhersagenden Attribute bekannt sein müssen. Existiert ein für die Klassifikation wichtiges Set vorhersagender Attribute mit unbekannten Werten, kann nicht mehr mit Sicherheit davon

ausgegangen werden, dass die gefundenen Regeln jedes noch unbekannte Beispiel korrekt seiner Klasse zuordnen.

Es existieren im Wesentlichen zwei Ansätze zur Behandlung dieses Problems: Zum einen werden Regeln nur dann konstruiert, wenn die Werte aller dafür notwendigen Attribute bekannt sind. Dies führt zur Entdeckung sogenannter *starker Regeln*. Ein Nachteil dieses Verfahrens ist, dass ein großer Teil der in der Datenbasis verborgenen Informationen nicht gefunden wird. Zum anderen kann nach Regeln gesucht werden, die nicht notwendigerweise Beispiele korrekt klassifizieren, sondern stattdessen die Wahrscheinlichkeit angeben, mit der ein Objekt zu einer Klasse gehört. Durch diese *Wahrscheinlichkeits-Regeln* können wichtige Informationen über Beziehungen in der Datenbasis gewonnen werden. Beispielsweise ist die Beziehung zwischen Rauchen und Lungenkrebs keine „korrekte“ Beziehung, da Rauchen keine notwendige Bedingung für Lungenkrebs ist. Dennoch existiert zwischen beiden Erscheinungen ein wichtiger Zusammenhang.

Ist kein bekannter Wert eines vorhersagenden Attributes wichtig für eine Klassifikation, sind Wert und Klasse unzusammenhängend. In diesem Fall können selbst Wahrscheinlichkeits-Regeln nicht mehr gefunden werden. Beispielsweise ist es nicht möglich, einen Beziehung zwischen dem Vornamen eines Menschen und seiner potenziellen Gefährdung durch Lungenkrebs zu finden.

4.7.2 Formulierbarkeit von Klassenbegrenzungen

Die Qualität konstruierbarer Regeln hängt unter Anderem davon ab, wie genau die damit zu beschreibenden Klassen von anderen Klassen abgegrenzt sind. Der exakte Verlauf einer Klassenbegrenzung kann aber nur gefunden werden, wenn die Datenbasis genügend viele grenznahe Beispiele beinhaltet. Dies gilt gleichermaßen für „gerade noch“ innerhalb wie „gerade noch“ außerhalb der Klasse liegende Beispiele.

Im Allgemeinen repräsentiert eine Datenbank nur einen kleinen Teil aller denkbaren Zustände der realen Welt. Auf Grund dieser Eigenschaft ist es nicht möglich, die Grenzen der Klassen exakt zu formulieren. Dies hat zur Folge, dass noch unbekannte Objekte inkorrekt klassifiziert werden können.

4.7.3 Objektbeziehungen zwischen Set und Subset

Ein Training Set S kann gesehen werden als Beispiel-Sammlung aller konstruierbaren Objekte – dem Universum U. Ein Data Mining-System sucht nach Regeln in diesem Set. Eine Regel besteht aus einer Beschreibung D und einer Klasse K und selektiert aus S ein Subset $\sigma_D(S)$. Dieses Subset kann ebenfalls als Beispiel-Sammlung gesehen werden.

Gehört ein beliebig gewähltes Objekt aus S mit der Wahrscheinlichkeit p_S zur Klasse K und gehört weiter ein beliebig gewähltes Objekt aus σ_D mit der Wahrscheinlichkeit p_{σ_D} zu K und gilt $p_S \neq p_{\sigma_D}$, so stellt sich die Frage, ob der Unterschied zwischen p_S und p_{σ_D} statistisch signifikant – und damit Teil einer neuentdeckten Beziehung zwischen den beteiligten Objekten – oder zufällig ist. Zur Klärung dieser Frage ist ein Data Mining-System auf die Anwendung statistischer Techniken zur Überprüfung der Gültigkeit derart gefundener Beziehungen angewiesen.

4.7.4 Fehlerhafte Datensätze

Ein Problem realer Datenbanken ist, dass die gespeicherten Datensätze Attribute enthalten können, deren Werte auf Messungen oder subjektiven Entscheidungen basieren. Beide Fälle enthalten die Möglichkeit fehlerhafter oder ungültiger Attributwerte, in deren Folge Beispiele falsch klassifiziert werden. Handelt es sich dabei um nichtsystematische Fehler, wird von *Rauschen* gesprochen. Rauschen führt im Wesentlichen in zwei Fällen zu Problemen: Zum einen bei der Konstruktion von Beschreibungen mittels eines „verrauschten“ Training Set, zum anderen bei der Klassifikation von Beispielen unter Anwendung der so gefundenen Beschreibungen.

Ein weiteres Problem, das bei der Benutzung von Datenbanken auftauchen kann, ist das Fehlen von Attributwerten. Beispiele mit fehlenden Werten können zur Lösung dieses Problems entweder aus dem Training Set entfernt oder mit getrennt zu berechnenden Repräsentativwerten ergänzt werden.

4.7.5 Größe und Dynamik der Datenbasis

Datenbanken können sehr groß sein, sowohl in Bezug auf die Zahl gespeicherter Datensätze als auch auf die Zahl von Attributen für jeden Datensatz. Für das Finden von Beziehungen zwischen den Objekten ist eine hohe Zahl von Attributen von Vorteil, da sie die Wahrscheinlichkeit erhöht, dass solche Beziehungen existieren. Zugleich steigert diese Informationsfülle aber auch die Zahl konstruierbarer Beschreibungen, so dass Techniken zur Begrenzung der damit einhergehenden Komplexität zum Einsatz kommen müssen.

Die Inhalte von Datenbanken können laufenden Veränderungen unterliegen: Datensätze werden hinzugefügt, verändert oder gelöscht. Zuvor aus der Datenbasis gewonnennes Wissen kann daher im Lauf der Zeit inkonsistent werden. Ein lernendes System muss sich derartigen Veränderungen anpassen können. Ein möglicher Lösungsansatz für die Problematik veralteter Informationen ist eine stärkere Gewichtung aktueller Daten innerhalb des Lern-Prozesses. Data Mining-Systeme können ihr verwendetes Regelwerk beispielsweise neu anpassen, wenn während eines

Zeitraumes eine bestimmte Zahl fehlerhafter Vorhersagen getroffen wurden.

4.8 Wissens-Repräsentation

4.8.1 Entscheidungsbäume

Ein *Entscheidungsbaum* klassifiziert Beispiele in eine endliche Anzahl von Klassen. Die Knoten des Baumes tragen die Namen der Attribute, die Äste werden mit den möglichen Werten dieser Attribute bezeichnet. Jedes Blatt des Baumes repräsentiert eine Klasse. Ein Beispiel wird klassifiziert durch das Durchwandern des Baumes von der Wurzel bis zu einem Blatt, wobei jeder Knoten einem Vergleich entspricht.

4.8.2 *if-then*-Regeln

Der Konstruktion eines Entscheidungsbaumes liegt die Eigenschaft von Attributen zu Grunde, Wertemengen zu besitzen. Mit Hilfe der Attribute kann jeder Ast des Baumes in eine Menge verknüpfter Bedingungen und damit in eine *if-then*-Regel umgewandelt werden. Regeln haben die Vorteile, dass:

- sie häufig als Repräsentationsform verwendet werden
- sie für Menschen verständlich sind
- jede einzelne für sich allein verwendet werden kann
- sie sich leicht generalisieren und spezialisieren lassen

Sie haben den Nachteil, dass viele der Bedingungen in der Nähe der Wurzel des Entscheidungsbaums, oft gespeichert und überprüft werden müssen. Teile des Baums, die zwei Äste gemeinsam haben, erscheinen auch als gleiche Bedingungen in den Regeln. Dadurch wird die Zahl der Regeln sehr groß und die Abarbeitung beschränkt sich im Wesentlichen auf viele Einzelvergleiche.

4.8.3 Entscheidungslisten

Um die Redundanz der Gesamtmenge der Regeln zu verringern, können diese in eine der Reihe nach abzuarbeitende Liste geschrieben werden. Solche *Entscheidungslisten* bestehen aus Paaren von Bedingungen und Klassen. Als letztes Element enthält die Liste eine leere elementare Angabe, die immer zutreffend ist und damit die korrekte Abarbeitung des Algorithmus gewährleistet. Um ein Beispiel zu klassifizieren, wird

beginnend mit dem Anfang der Liste jede Bedingung auf Korrektheit geprüft. Ist dies der Fall, erfolgt die Zuordnung des Beispiels zur entsprechenden Klasse und das Verfahren bricht ab, andernfalls wird die folgende Bedingung geprüft. Ein Nachteil von Entscheidungslisten ist die – bei zu erwartender Komplexität $O(N)$ – hohe Zahl von benötigten Vergleichen (lineares Verhalten). Entscheidungslisten sind insbesondere dann interessant, wenn es eine Klasse gibt, in die die Mehrzahl der Beispiele fällt und nur einzelne unsystematische Ausnahmen getrennt behandelt werden müssen. Spezialfälle und Ausnahmen von allgemeineren Regeln stehen dabei am Anfang der Liste, so dass in diesen Fällen vor dem Erreichen der häufigeren Fälle abgebrochen wird. Dies kann zu einem verhältnismäßig ineffektiven System führen, da für viele Beispiele die unzutreffenden Spezialfälle zwangsläufig mit geprüft werden müssen. Zudem werden inhaltlich zusammengehörende Bedingungen nicht notwendigerweise in der Liste zusammenstehen, da die strikte Sequentialität dies verhindert.

4.8.4 *Ripple-Down*-Regelmengen

Um die Abhängigkeit von der Reihenfolge aufzuheben, werden *Ripple-Down*-Regelmengen verwendet. Diese Mengen fassen Regeln und ihre Ausnahmen in geeigneten *if-then*-Strukturen zusammen. Durch die lokale Anbindung der Regeln an die Ausnahmen entfällt die Notwendigkeit der Abarbeitung einer vorgegebenen Reihenfolge. Regeln können nach inhaltlichen Gesichtspunkten oder nach Häufigkeit angeordnet werden.

4.8.5 Graphen und Schemata

Die Repräsentation von Wissen kann durch die Nutzung ausdrucksstärkerer Formalismen ebenso verbessert werden, wie durch die Verwendung verständlicher, strukturierter Repräsentationen. Dabei kommen vorrangig folgende Konzepte zur Anwendung:

> Ein *semantisches Netz* ist ein Graph, in dem die Knoten Konzepte bezeichnen, während die Pfeile Beziehungen zwischen diesen Konzepten darstellen.

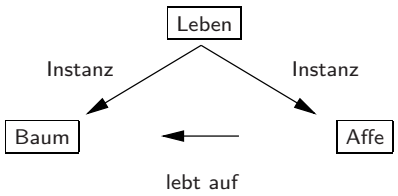

Abbildung 4.8: Beispiel eines einfachen semantischen Netzes

Der Hauptvorteil semantischer Netze ist das schnelle Finden sämtlicher Informationen über ein Objekt, durch einfaches Verfolgen der von diesem Objekt ausgehenden Beziehungs-Pfeile. Werden semantische Netze für Data Mining benutzt, stellt jedes Beispiel ein separates Netz dar. Operationen auf diesen Beispielen sind Manipulationen der zugehörigen Graphen, beispielsweise zur Findung eins Subgraphen, der für alle Mitglieder einer Klasse identisches Aussehen besitzt.

Ein *Schema* ist ein strukturiertes Objekt, bestehend aus einem Namen und bezeichneten Attributen, die mit Werten der entsprechenden Instanz gefüllt sind. Instanzübergreifende Informationen können durch die Benutzung spezieller Attribute realisiert werden.

4.8.6 Neuronale Netze

Neuronale Netze sind Architekturen, deren Struktur und Funktion sich an den Nervennetzen lebender Organismen orientiert. In Analogie zu den aus natürlichen Neuronen bestehenden Nervennetzen sind die parallel arbeitenden künstlichen Neuronen in logischen Schichten angeordnet und innerhalb derer sowie über das gesamte Netz hinweg miteinander verbunden. Jeder Verbindung zwischen zwei Knoten ist ein Gewicht zugeordnet. Ein Neuron berechnet die gewichtete Summe seiner Eingänge und reagiert entsprechend seiner Schwellwertfunktion.

Neuronale Netze haben in Bezug auf Data Mining verschiedene Nachteile: Zum einen verläuft der Lern-Prozess, verglichen mit herkömmlichen Techniken, sehr langsam. Zum anderen erfolgt die Repräsentation gefunden Wissens nicht durch Regeln oder Muster, sondern durch das Aussehen des Netzes selbst. Eine der Hauptaufgaben eines Data Mining-Prozesses ist aber die Präsentation von Wissen in für Menschen verständlicher Form. Auf neuronale Netze wird daher in dieser Arbeit nicht näher eingegangen.

Kapitel 5

Praktische Ausführungen

In einigen Anwendungsgebieten lassen sich Sequenzen von Ereignissen sammeln, die das Verhalten von Nutzern oder Systemen beschreiben. In diesem Kapitel wird das Problem der Entdeckung relevanter Muster durch Data Mining Algorithmen in solchen Ereignis-Sequenzen am Beispiel von Alarmmeldungen eines GSM-Mobilfunknetzes betrachtet. Dabei wird eine *Alarmgruppe* definiert als eine partiell geordnete Sammlung von *Einzelalarmen*, die innerhalb eines Zeitintervalls gegebener Länge auftreten. Nach der Entdeckung der *relevanten Alarmgruppen* lassen sich Regeln für die Beschreibung oder Vorhersage des Verhaltens des betrachteten Systems konstruieren.

Die praktischen Ausführungen dieses Kapitels gliedern sich in zwei große Bereiche, der Vorverarbeitung der vorliegenden Daten sowie der anschließenden Musterfindung. Das C++-Programm ISIS (Intelligent Search of Interesting Patterns in Sequences) wurde für diese Zwecke im Rahmen dieser Arbeit entwickelt. Die für einen vollständigen Data Mining-Prozess notwendige Regelbildung als darauf aufbauender Vorgang ist nicht Gegenstand dieser Arbeit.

Die im Folgenden vorgestellten Algorithmen sind nicht auf das hier benutzte Beispiel beschränkt. Die Begriffe Alarmgruppe und Einzelalarm können problemlos durch die allgemeineren Termini Ereignisgruppe und Einzelereignis ersetzt werden.

Die theoretischen Ausführungen dieses Kapitels, sowie die verwendeten Algorithmen zur Musterfindung, stützen sich im Wesentlichen auf [Toi96].

5.1 Ereignis-Sequenzen

Die meisten Data Mining Systeme sind angepasst an die Analyse ungeordneter Datenmengen. Daneben gibt es aber bedeutende Anwendungsgebiete, in denen die zu analysierenden Daten einer bestimmten Struktur unterliegen. Es ist beispiels-

weise leicht möglich durch die Überwachung von Telekommunikations-Anlagen eine große Menge an Informationen über das Verhalten eines solchen Systems zu sammeln. Solch eine Datensammlung kann als eine Sequenz von Ereignissen (in unserem Beispiel Alarmen) gesehen werden, in der jedem Ereignis eine bestimmte Zeit des Auftretens zugeordnet werden kann (ein Zeitstrahl von Ereignissen).

Zeitpunkt t_n	aufgetretene Alarme A_n
t_1	$A_1\ A_2\ A_{17}\ A_{55}$
t_2	A_6
t_3	$A_3\ A_{13}\ A_{13}$
$\vdots$	$\vdots$

Tabelle 5.1: Map „alarm_sequence“

Ein bedeutendes Problem in der Analyse einer solchen Sequenz ist das Finden *frequenter* Ereignisgruppen (Alarmgruppen), beispielsweise einer Sammlung von Alarmen, die mit einer gewissen Wahrscheinlichkeit nacheinander auftreten.

Alarmgruppen können als geordnete, azyklische Graphen beschrieben werden:

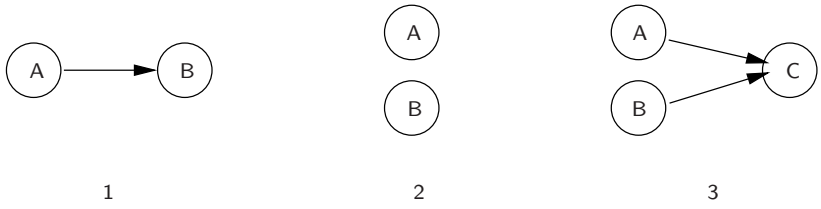

Abbildung 5.1: Alarmgruppen

Beispiel 5.1 (Alarmgruppen)
Sei a eine Alarmgruppe. A, B und C seien Einzelalarme. Dann wird a (mit A und B als zugehörigen Einzelalarmen) als *serielle* Alarmgruppe bezeichnet, wenn auf A immer B folgt (Abbildung 5.1, Fall 1). Ist die Reihenfolge hingegen beliebig, handelt es sich bei a um eine *parallele* Alarmgruppe (Fall 2). Besteht a aus allen drei Einzelalarmen und folgt C immer auf A und B (deren Reihenfolge wiederum beliebig ist), handelt es sich um eine nicht-serielle und nicht-parallele Alarmgruppe (Fall 3). Beachtenswert ist weiterhin, dass zwischen dem Auftreten von A, B und C weitere Alarme liegen können.

Das Ziel bei der Entdeckung von Alarmgruppen in Telekommunikations-Netzwerken ist das Finden unbekannter Beziehungen zwischen Alarmen. Solche Beziehungen

können für die Echtzeit-Analyse eingehender Datenströme benutzt werden, um eine bessere Erklärung für die Ursachen aufgetretener Alarme zu geben. Eine Filterung unwichtiger Alarme ist ebenso möglich, wie die Vorhersage für das zukünftige Verhalten des Systems.

5.2 Voraussetzungen

ISIS wurde mit dem Ziel entwickelt, Korrelationen von Alarmmeldungen in GSM-Mobilfunknetzen zu entdecken. Dazu wird eine Alarm Datenbank in Textform vorausgesetzt, in der die Einträge folgendes Aussehen besitzen:

Beispiel 5.2 (Alarmmeldung)

```
12-60-0001;12-70-0093;12-71-0102;12-72-0176;24;7166811;4;
 7006;TRX/FU: NO DATA FROM DSPx TO FUCx;1998-02-27 03:16:30;0;
 1998-02-27 03:16:30;0;autoack;1998-02-27 03:16:30;system;
```

Eine solche Alarmmeldung (im Folgenden nur noch als Alarm bezeichnet) entspricht im Wesentlichen der Art von Meldung, die ein Netzelement der Firma Nokia erzeugt. Prinzipiell ist aber jede in Textform darstellbare Art von Alarmen nach einer Anpassung der Einlesefunktion verarbeitbar.

Die Bedeutung der einzelnen Elemente eines solchen Alarms ist Folgende:

```
12-60-0001: root_object
12-70-0093: child_object_1
12-71-0102: child_object_2
12-72-0176: child_object_3
24        : managed_object_class
7166811   : consec_number
4         : severity_number
7006      : alarm_number
TRX/FU... : alarm_text
1998-02-27: event_time
03:16:30  : event_time
0         : alarm_status
1998-02-27: acknowledge_time
03:16:30  : acknowledge_time
autoack   : acknowledge_status
1998-02-27: cancel_time
03:16:30  : cancel_time
system    : cancelled_by
```

Für die weiteren Ausführungen sind jedoch nur wenige Elemente von Bedeutung. Dazu gehört child_object_3, das als Auslöser des Alarms betrachtet wird[1], sowie alarm_number als eindeutigen Bezeichner für die Art des Alarms und event_time, welche die Auftretenszeit des Alarms bezeichnet. Für eine spätere Erweiterbarkeit der Programmfunktionalität werden dennoch alle Elemente eines Alarms in die interne Datenstruktur eingelesen und verfügbar gehalten.

Die zur Vorverarbeitung (Abschnitt 5.3) benötigte Konfigurations Datenbank spiegelt den Aufbau der Netzelemente wider. Dabei ist es prinzipiell nicht von Bedeutung, ob dieser Aufbau der logischen oder der physikalischen Sichtweise entspricht (das Gleiche kann für die zu erwartenden Ergebnisse nicht gesagt werden). Ein Eintrag dieser Datenbank ordnet einem Mutter-Element die zugehörigen Tochter-Elemente zu. Die Datenbank selbst liegt wie die Alarm Datenbank in Textform vor:

Beispiel 5.3 (Konfigurations Datenbank)

```
12-60-0001(12-70-0048;12-70-0049;12-70-0051;12-70-0052;...)
12-70-0048(12-71-0022;12-71-0023;12-71-0024)
12-71-0022(12-72-0070;12-72-0133)
12-71-0023(12-72-0071;12-72-0134)
12-71-0024(12-72-0072;12-72-0135)
12-72-0070()
12-72-0133()
...
```

Der Bezeichner eines Netzelementes setzt sich aus seinem Gültigkeitsbereich oder Sektor (12 steht beispielsweise für Hannover), seinem Typ (60 = BSC, 70 = BS, 71 = BTS, 72 = TRX), sowie einer Nummer für die Unterscheidung der Netzelemente einer Hierarchiestufe untereinander zusammen. Die Nomenklatur der Netzelement-Bezeichner entspricht ebenfalls der der Firma Nokia. Wie bei der Alarm Datenbank ist eine Anpassung an andersartig aufgebaute Bezeichner durch Modifikationen an der zuständigen Einlesefunktion möglich.

5.3 Vorverarbeitung

Ein wesentlicher Bestandteil eines jeden Data Mining-Prozesses ist die Vorverarbeitung der zu analysierenden Daten. Sie soll verhindern, dass Zusammenhänge zwischen Bereichen gesucht werden, die prinzipbedingt nicht existieren können. Deswei-

[1] In der Realität kann eine Meldung auch stellvertretend für child_object_3 von einem übergeordneten Netzelement erzeugt werden, beide Arten der Erzeugung sind nicht unterscheidbar und können daher nicht berücksichtigt werden.

teren ist mit einer geschickten Vorverarbeitung ein nicht unerheblicher Geschwindigkeitsgewinn verbunden, da die zu bearbeitende Aufgabe an Komplexität verliert.

Im Rahmen dieser Arbeit wurde eine Unterteilung der vorliegenden Alarmdaten in *Cluster* vorgenommen. Ein Cluster besteht aus den Alarmen einer BS mit den ihr zugehörigen Tochter-Elementen. Dieser Einteilung liegt die Annahme zu Grunde, dass Alarme zwischen verschiedenen Basisstationen unkorreliert sind. Die Alarme der BSC passen in dieses Schema nicht hinein und bilden daher ebenfalls einen eigenen Bereich. Allgemein lässt sich sagen, dass eine solche Einteilung ein extrem kritischer Vorgang ist, da er vermutlich das resultierende Ergebnis stark beeinflusst. Beispielsweise entspricht eine logische Unabhängigkeit zweier Elemente nicht zwangsläufig auch einer physikalischen Unabhängigkeit. Korrelationen von Alarmen physikalisch abhängiger Netzelemente (Basisstationen, die über gemeinsame Richtfunkstrecken mit ihrer BSC verbunden sind gehören dazu), sind aber über eine rein logische Sichtweise auf diese Elemente nicht begründbar. Das Einfließen von – im Data Mining-Kontext Expertenwissen genannten – Informationen in diesen Prozess ist deshalb unabdingbare Voraussetzung für eine erfolgversprechende Bearbeitung der vorliegenden Aufgabe.

5.4 Musterfindung

Die Aufgabe der Musterfindung lässt sich wie folgt umschreiben: Gegeben sei eine Klasse von Alarmgruppen a, ein Zeitstrahl von Ereignissen s, eine Fensterbreite $window\_width$ sowie ein Frequenz-Schwellwert $min\_frequency$. Finde alle Alarmgruppen a, die eine genügende Häufigkeit auf s aufweisen.

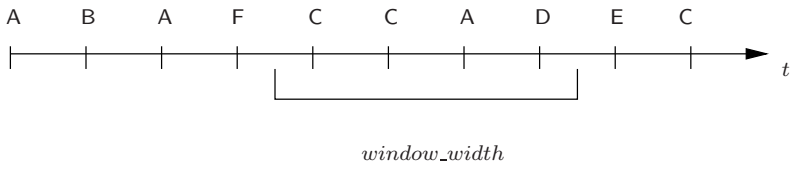

Abbildung 5.2: Ereignis-Sequenz s

Die Erkennung von Alarmgruppen erfolgt dabei in effizienter Weise durch das Hinübergleitenlassen eines Fensters der Breite $window\_width$ über die Ereignis-Sequenz (Abbildung 5.2). Eine bedeutende Eigenschaft eines solchen Fensters besteht in der großen Änlichkeit zu seinem angrenzenden Fenster, die in der weitgehenden Überlappung beider begründet ist. Ein nutzbarer Vorteil dieser Änlichkeit ist die Möglichkeit, die verwendeten Datenstrukturen schrittweise zu aktualisieren um zu bestimmen, welche Alarmgruppe im jeweils folgendem Fenster erscheinen wird.

Die im Folgenden beschriebenen Algorithmen fanden bei der Implementierung von

ISIS Verwendung. Die grundlegende Idee besteht in dem Wechsel zwischen der Generierung neuer potenziell interessierender Alarmgruppen und der Überprüfung auf ausreichende Häufigkeit des Vorkommens (Frequenz) an Hand der Ereignis-Sequenz. Den Startpunkt bilden Alarmgruppen der Länge $l = 1$. Das Verfahren bricht ab, wenn keine neuen Alarmgruppen der Länge $l + 1$ mehr gebildet werden können (es existieren keine frequenten Alarmgruppen der Länge l).

5.4.1 Generierung paralleler Alarmgruppen

Algorithmus 5.1 ist ein Verfahren zur Berechnung potenziell interessierender paralleler Alarmgruppen. Eine Alarmgruppe a wird repräsentiert durch ein lexikographisch geordnetes Array von Einzelalarmen. Ein Array wird durch den Namen der Alarmgruppe, die Einträge im Array über eine Notation mittels eckiger Klammern bezeichnet. Beispielsweise könnte sich eine Alarmgruppe a mit den zugehörigen Einzelalarmen A_1, A_2 und A_3 über das Array $a[1] = A_1$, $a[2] = A_2$, $a[3] = A_2$, $a[4] = A_2$ und $a[5] = A_3$ darstellen. Sammlungen von Alarmgruppen werden ebenfalls als lexikograpisch geordnete Arrays dargestellt, beispielsweise wird die i-te Alarmgruppe der Länge l einer solchen Sammlung durch $C_l[i]$ bezeichnet, wenn es sich um ein Array potenziell relevanter Alarmgruppen handelt („candidates“), oder durch $F_l[i]$, wenn die Relevanz der Alarmgruppen bereits bestätigt wurde („frequencys“).

Da die Alarmgruppen und die Sammlungen von Alarmgruppen sortiert sind, sind alle Alarmgruppen mit den gleichen Anfangsalarmen aufeinanderfolgend angeordnet. Teilen sich beispielsweise die Alarmgruppen $F_l[i]$ und $F_l[j]$ der Länge l die ersten $l - 1$ Alarme, dann gilt für alle k mit $i \leq k \leq j$, dass $F_l[k]$ ebenfalls diese Alarme beinhaltet. Eine maximale Sequenz derartig angeordneter Alarmgruppen der Länge l, mit den gleichen $l - 1$ Anfangsalarmen, wird als *Block* bezeichnet. Potenziell interessierende Alarmgruppen können durch Bilden aller möglichen Kombinationen zweier Alarmgruppen innerhalb eines solchen Blocks gefunden werden. Die fundamentale Idee dabei ist, dass nur solche Alarmgruppen a der Länge $l+1$ frequent sein können, wenn alle aus a bildbaren Teilgruppen b frequent sind. Durch diesen Ansatz entfällt die Notwendigkeit der Bildung und Überprüfung aller theoretisch möglichen Kombinationen.

Für einen effizienten Zugriff auf die Blöcke wird für jede Alarmgruppe $F_l[j]$ in $block\_start[j]$ die Position i des ersten Block-Elementes $F_l[i]$ gespeichert.

5.4.2 Erkennen frequenter paralleler Alarmgruppen

Algorithmus 5.2 startet mit der Betrachtung des ersten Fensters kurz vor Beginn der Alarm-Sequenz. In diesem Fenster ist der Alarm des ersten Zeitpunktes gerade

Eingabe: F_l
Ausgabe: C_{l+1}

```
1:  C_{l+1} := ∅;
2:  k := 0;
3:  if l = 1 then
4:    for h := 1 to |F_l| do
5:      F_l.block_start[h] := 1;
6:  for i := 1 to |F_l| do
7:    current_block_start := k + 1;
8:    for (j := i; F_l.block_start[j] = F_l.block_start[i]; j := j + 1;) do
9:      for x := 1 to l do
10:       a[x] := F_l[i][x];
11:     a[l + 1] := F_l[j][l];
12:     for y := 1 to l − 1 do
13:       for x := 1 to y − 1 do
14:         b[x] := a[x];
15:       for x := y to l do
16:         b[x] := a[x + 1];
17:       if b ist nicht in F_l then
18:         setze fort mit nächstem j in Zeile 8;
19:     k := k + 1;
20:     C_{l+1}[k] := a;
21:     C_{l+1}.block_start[k] := current_block_start;
22: Ausgabe C_{l+1}
```

Algorithmus 5.1: Generierung paralleler Alarmgruppen

Alarmgruppen $C_5[i]$ bzw. $F_5[i]$	Einzelalarme A	Blockbezeichner *block_start*
a_1	$A_1\ A_2\ A_2\ A_2\ A_3$	1
a_2	$A_1\ A_2\ A_2\ A_2\ A_4$	1
a_3	$A_5\ A_6\ A_6\ A_8\ A_8$	3
⋮	⋮	⋮

Tabelle 5.2: Array „candidates“ C_5 bzw. „frequencys“ F_5

noch nicht enthalten. Er endet mit dem letzten Fenster unmittelbar nach Ende der Alarm-Sequenz. In diesem Fenster ist der Alarm des letzten Zeitpunktes gerade nicht mehr enthalten. Dieses Vorgehen erspart eine Sonderbehandlung der Sequenzenden.

a_n	event_count	freq_count	in_window

Tabelle 5.3: Map „a_efw_count“

Das Prinzip des Algorithmus ist folgendes: Für jede in Frage kommende parallele Alarmgruppe a existiert ein Zähler *a_event_count* (Tabelle 5.3), der anzeigt, wie viele Einzelalarme A von a im aktuellen Fenster vorkommen. Wenn *a_event_count* identisch zur Größe der aktuell untersuchten Alarmgruppen *size_of_alarmgroup* wird, bedeutet dies, dass a vollständig im Fenster vorhanden ist. In diesem Fall wird die Startzeit des Fensters in *a_in_window* gespeichert. Verringert sich *a_event_count* wieder – ist a also nicht mehr vollständig im aktuellen Fenster vertreten – wird *a_freq_count* um die Anzahl Fenster erhöht, in denen a vollständig enthalten war. Der Zähler *a_freq_count* enthält am Ende somit die Gesamtzahl der Fenster, in denen a vollständig vorkam.

Alarm A_n	mit der Häufigkeit	ist Bestandteil dieser Alarmgruppen
A_1	1	$a_1\ a_5\ a_{13}\ a_{27}$
A_1	2	$a_3\ a_4$
⋮	⋮	⋮
A_2	1	$a_{23}\ a_{25}$
⋮	⋮	⋮

Tabelle 5.4: Map „contains(A_n,i)“

Für einen effizienten Zugriff auf die Kandidaten (die potenziell interessierenden Alarmgruppen), werden diese durch die Zahl der Alarme jeden Typs den sie enthalten, indiziert: Alle Alarmgruppen, die genau i Alarme des Typs A enthalten,

Eingabe: $C_l, s = (T_s, T^s, s), window\_width, min\_frequency$
Ausgabe: $F_l = f(window\_width, min\_frequency)$

```
1: for jedes a in C do
2:    for jedes A in a do
3:       A_count := 0;
4:       for i := 1 to |a| do
5:          contains(A, i) := ∅;
6: for jedes a in C do
7:    for jedes A in a do
8:       z := Zahl der Alarme des Typs A in a;
9:       contains(A, z) := contains(A, z) ∪ {a};
10:   a_event_count := 0;
11:   a_freq_count := 0;
12: for start := T_s − window_width + 1 to T^s do
13:   for alle Alarme (A, t) in s, so dass gilt t = start + window_width − 1 do
14:      A_count := A_count + 1;
15:      for jedes a ∈ contains(A, A_count) do
16:         a_event_count := a_event_count + A_count;
17:         if a_event_count = |a| then
18:            a_in_window := start;
19:   for alle Alarme (A, t) in s, so dass gilt t = start − 1 do
20:      for jedes a ∈ contains(A, A_count) do
21:         if a_event_count = |a| then
22:            a_freq_count := a_freq_count − a_in_window + start;
23:         a_event_count := a_event_count − A_count;
24:      A_count := A_count − 1;
25: for alle Alarmgruppen a in C do
26:   if a_freq_count/(T^s − T_s + window_width − 1) ≥ min_frequency then
27:      gib a aus;
```

Algorithmus 5.2: Erkennen frequenter paralleler Alarmgruppen

werden in der Liste *contains(A_n,i)* aufgeführt (Tabelle 5.4). Ändert sich der Fensterinhalt durch Weiterbewegen, werden alle betroffenen Alarmgruppen aktuallisiert. Gibt es beispielsweise einen Alarm des Typs A_1 im Fenster und ein zweiter Alarm des gleichen Typs kommt hinzu, werden alle Alarmgruppen der Liste *contains(A_1,2)* dahingehend „informiert“, dass beide erwarteten Ereignisse eingetreten sind.

Alarm A_n	Vorkommen im aktuellen Fenster
A_1	3
A_2	25
A_3	2
⋮	⋮

Tabelle 5.5: Map „A_n_count“

Der Zähler *A_count* enthält alle vorkommenden Alarme sowie deren Anzahl im aktuellen Fenster. Für jeden Alarm A_n des aktuellen Zeitpunktes t wird *A_count* um die Zahl des Auftretens erhöht. Verringert wird *A_count* in gleicher Weise für alle Alarme, die das Fenster wieder verlassen.

5.4.3 Generierung serieller Alarmgruppen

Algorithmus 5.1 kann in einfacher Weise zur Bildung serieller Alarmgruppen abgeändert werden. Nunmehr beruhen die Alarme eines Arrays, die eine Alarmgruppe repräsentieren, auf einer Ordnung. Beispielsweise wird die serielle Alarmgruppe b mit den zugehörigen Einzelalarmen C, A, F und C – in dieser Reihenfolge – durch ein Array $b[1] = C$, $b[2] = A$, $b[3] = F$, $b[4] = C$ dargestellt.

Die einzige notwendige Änderung bezieht sich auf Zeile 8:

// Ersetzen von Zeile 8 aus Algorithmus 5.1 durch:
for $(j := F_l.block\_start[i]; F_l.block\_start[j] = F_l.block_start[i]; j := j + 1;)$ **do**
...

Algorithmus 5.3: Generierung serieller Alarmgruppen

Der so modifizierte Algorithmus funktioniert, da die dem Verfahren zu Grunde liegenden frequenten Alarmgruppen nach wie vor in lexikographischer Ordnung vorliegen. Somit hat das in Abschnitt 5.4.1 über die Bildung neuer Alarmgruppen gesagte ebenfalls Gültigkeit.

5.4.4 Erkennen frequenter serieller Alarmgruppen

Serielle Alarmgruppen einer Alarm-Sequenz können durch die Benutzung eines *Zustands-Automaten* gefunden werden. Dieser akzeptiert ausschließlich potenziell relevante Gruppen, sonstige Eingangsdaten werden ignoriert. Die Idee eines solchen Automaten ist folgende: Für jede serielle Alarmgruppe a existiert ein Zustands-Automat. Zur gleichen Zeit können für a mehrere Instanzen eines Automaten existieren, so dass dessen aktive Zustände das Vorkommen von a im aktuellen Fenster reflektieren. Algorithmus 5.4 implementiert diese Idee.

Eine neue Instanz eines Automaten wird immer dann initialisiert, wenn der erste Alarm einer Alarmgruppe in das aktuelle Fenster aufgenommen wird. Der Automat wird entfernt, wenn der selbe Alarm das Fenster wieder verläßt. Wenn ein Automat für a seinen *Akzeptanz-Zustand* erreicht, bedeutet das, dass a vollständig im Fenster vorhanden ist. Existieren nicht bereits weitere Automaten für a, wird die Startzeit des Fensters in a_in_window gespeichert. Wird ein Automat im Akzeptanz-Zustand gelöscht und existieren keine weiteren Automaten für a im Akzeptanz-Zustand, wird a_freq_count um die Anzahl Fenster erhöht, in denen a vollständig enthalten war.

a_n	freq_count	in_window

Tabelle 5.6: Map „a_fw_count“

Es ist nicht sinnvoll, für einen bestimmten Zustand einer Gruppe mehrere Automaten zu verwenden, da damit kein Gewinn an Information verbunden ist. Somit reicht es aus, sich um den Automaten zu kümmern, der den gemeinsamen Zustand als letzter erreicht. Der Grund liegt in der Tatsache, dass eben dieser Automat auch als letzter gelöscht werden wird. Es existieren daher maximal $|a|$ Automaten für eine Alarmgruppe a, wobei $|a|$ der Anzahl an Einzelalarmen entspricht, aus denen a besteht. Somit ist es möglich, alle zu a gehörenden Automaten in einem Array der Größe $|a|$ zu repräsentieren: Der Wert von $a\_initialized[i]$ entspricht der letzten Initialisierungszeit des Automaten, der seinen i-ten Zustand erreicht hat. Durch die Darstellung von a als ein Array von Einzelalarmen (siehe Abschnitt 5.4.1) ist es möglich, dieses für die Kennzeichnung der Zustands-Übergänge zu benutzen.

Für einen effizienten Zugriff auf die Automaten sind diese in folgender Weise organisiert: Für jeden Einzelalarm $A \in R$ (R ist die Gesamtmenge aller vorkommenden Alarme), ist der A akzeptierende Automat verbunden mit einer Liste $waits\_A(A)$. Die Liste besteht aus Einträgen der Form (a, x) was bedeuted, dass die Alarmgruppe a auf ihren x-ten Einzelalarm wartet. Ändert sich der Fensterinhalt durch Weiterbewegen, wird die Liste $waits\_A(A)$ durchlaufen und die zu den betroffenen Alarmgruppen gehörenden Einträge aktualisiert. Erreicht ein Automat einen gemeinsamen Zustand mit einem anderen Automaten, wird dieser ältere Eintrag in $a\_initialized[]$ überschrieben.

Eingabe: $C_l, s = (T_s, T^s, s), window\_width, min\_frequency$
Ausgabe: $F_l = f(window\_width, min\_frequency)$

1: **for** jedes a in C **do**
2: **for** $i := 1$ **to** $|a|$ **do**
3: $a\_initialized[i] := 0$;
4: $waits\_A(a[i]) := \emptyset$;
5: **for** jedes $a \in C$ **do**
6: $waits\_A(a[1]) := waits\_A(a[1]) \cup \{(a, 1)\}$;
7: $a\_freq\_count := 0$;
8: **for** $t := T_s - window\_width$ **to** $T_s - 1$ **do**
9: $begins\_at(t) := \emptyset$;
10: **for** $start := T_s - window\_width + 1$ **to** T^s **do**
11: $begins\_at(start + window\_width - 1) := \emptyset$;
12: $transitions := \emptyset$;
13: **for** alle Alarme (A, t) in s, so dass gilt $t = start + window\_width - 1$ **do**
14: **for** alle $(a, j) \in waits\_A(A)$ **do**
15: **if** $j = |a|$ **and** $a\_initialized[j] = 0$ **then**
16: $a\_in\_window := start$;
17: **if** $j = 1$ **then**
18: $transitions := transitions \cup \{(a, 1, start + window\_width - 1)\}$;
19: **else**
20: $transitions := transitions \cup \{(a, j, a\_initialized[j - 1])\}$;
21: $begins\_at(a\_initialized[j - 1] :=$
$begins\_at(a\_initialized[j - 1]) \setminus \{(a, j - 1)\}$;
22: $a\_initialized[j - 1] := 0$;
23: $waits\_A(A) := waits\_A(A) \setminus \{(a, j)\}$;
24: **for** alle $(a, j, t) \in transitions$ **do**
25: $a\_initialized[j] := t$;
26: $begins\_at(t) := begins\_at(t) \cup \{(a, j)\}$;
27: **if** $j < |a|$ **then**
28: $waits\_A(a[j + 1]) := waits\_A(a[j + 1]) \cup \{(a, j + 1)\}$;
29: **for** alle $(a, l) \in begins\_at(start - 1)$ **do**
30: **if** $l = |a|$ **then**
31: $a\_freq\_count := a\_freq\_count - a\_in\_window + start$;
32: **else**
33: $waits\_A(a[l + 1]) := waits\_A(a[l + 1]) \setminus \{(a, l + 1)\}$;
34: $a\_initialized[l] := 0$;
35: **for** alle Alarmgruppen a in C **do**
36: **if** $a\_freq\_count/(T^s - T_s + window\_width - 1) \geq min\_frequency$ **then**
37: gib a aus

Algorithmus 5.4: Erkennen frequenter serieller Alarmgruppen

Alarm A_n	Zustands-automat
A_1	$(a_1, 1)$ $(a_2, 4)$ $(a_3, 1)$ ...
A_2	$(a_1, 3)$ $(a_4, 1)$ $(a_5, 2)$ ...
A_3	$(a_1, 2)$ $(a_2, 1)$ $(a_3, 2)$ ...
⋮	⋮

Tabelle 5.7: Map „waits_A(A)“

Die während des Weiterbewegens des Fensters um einen Zeitschritt gemachten Zustands-Übergänge werden in der Liste *transitions* erfasst. Diese wird in der Form (a, x, t) repräsentiert, was bedeutet, dass die Alarmgruppe a ihren x-ten Einzelalarm erhalten hat, wobei die letzte Initialisierungszeit des Vorgängers der Länge x, t ist. Aktualisierungen, die alte Zustände der Automaten betreffen, werden augenblicklich durchgeführt, Aktualisierungen für neue Zustände hingegen erst nachdem alle Übergänge identifiziert worden sind. Dieses Vorgehen soll vermeiden, dass nützliche Information überschrieben wird. Für ein leichteres Entfernen von Automaten, die das aktuelle Fenster verlassen haben, werden deren Initialisierungszeiten in der Liste $begins\_at(t)$ gespeichert.

Alarmgruppe a_n	aktueller Zustand x	Initialisierungs-zeit t
a_1	1	27345
a_1	2	27202
a_3	1	27295
⋮	⋮	⋮

Tabelle 5.8: Map „transitions“

5.4.5 Injektive parallele und serielle Alarmgruppen

Die Algorithmen 5.1 und 5.3 sind durch eine weitere Modifikation in der Lage, *injektive* Alarmgruppen – also Alarmgruppen in denen jeder Typ eines Einzelalarms nur einmal auftreten darf - zu bilden. Die einzige notwendige Ergänzung betrifft wie schon zuvor Zeile 8:

// Hinzufügen der Zeilen 8a und 8b zu den Algorithmen 5.1 und 5.3:
if $j = i$ **then**
  setze fort mit nächstem j in Zeile 8;

Algorithmus 5.5: Generierung injektiver Alarmgruppen

Der Effekt der hinzugefügten Zeilen ist, dass all die Alarmgruppen nicht gebildet werden, in denen ein Einzelalarm mindestens zweimal vorkommt.

Zur Überprüfung auf ausreichende Frequenz werden die Algorithmen 5.2 und 5.4 in unveränderter Form genutzt.

5.5 ISIS

Die Bedienung von ISIS beschränkt sich auf die Übergabe von Startparametern beim Aufruf des Programms. Während des Programmablaufs ist keine Interaktion mit dem Benutzer vorgesehen. Ein typischer Aufruf sieht in etwa so aus:

Beispiel 5.4 (Programmaufruf ISIS)

```
isis -a alarm_db.log -c config_db.log -m pPsS -w 60 -f 100
     -l /log/logdateien
```

Die Bedeutung der einzelnen Parameter ist Folgende:

`-a` (Pfad und) Dateiname der Alarm Datenbank
`-c` (Pfad und) Dateiname der Konfigurations Datenbank
`-f` Angabe der Frequency in Prozent
`-l` Pfad der Logdateien
`-h` Ausgabe eines Hilfetextes
`-m` Modus
`-v` Ausgabe eines Versionstextes
`-w` Angabe der Fensterbreite in Sekunden

Frequency bezieht sich auf die durchschnittliche Häufigkeit der aktuell bearbeiteten Alarmgruppen. Ein Wert von 100 bedeutet beispielsweise, dass eine Alarmgruppe mindestens eine Häufigkeit des Auftretens von 100 Prozent des Durchschnittswertes aufweisen muss, um als frequent angesehen zu werden. Ein Wert von 0 bedeutet, dass sie mindestens einmal vorkommen muss (in diesem Fall werden alle tatsächlich vorkommenden Alarmgruppen als frequent betrachtet). Ein Wert von 200 bedeutet, dass sie doppelt so oft vorkommen muss, wie eine durchschnittlich vorkommende Alarmgruppe.

Über den Modus wird bestimmt, nach welcher Art von Alarmgruppen gesucht werden soll. ‚P‘ entspricht parallelen Alarmgruppen mit Wiederholungen (ein Einzelalarm darf mehrfach in einer solchen Gruppe vorkommen), ‚p‘ entspricht parallelen Alarmgruppen ohne Wiederholungen. ‚S‘ und ‚s‘ besitzen die gleiche Bedeutung für serielle Alarmgruppen.

Die Reihenfolge der Parameter ist beliebig, ebenso die Reihenfolge der Schalter bei der Angabe des Modus.

Alle Ausgaben werden in Textdateien geschrieben, die nach folgendem Schema benannt sind:

Beispiel 5.5 (Ausgabedateien)
`cand-p-60s-100.log`
`freq-p-60s-100.log`
`cand-is-30s-90.log`
`freq-is-30s-90.log`

Mit Hilfe von ‚cand' und ‚freq' wird zwischen Freqencys und Candidates unterschieden. ‚p' und ‚s' bezeichnen parallele oder serielle Alarmgruppen mit Wiederholungen, ‚ip' und ‚is' deren Varianten ohne Wiederholungen (injektiv). ‚60s' ist ein Beispiel für eine gewählte Fensterbreite von 60 Sekunden, ‚100' entspricht einer gewählten Frequenz von 100%.

Kapitel 6

Zusammenfassung

Die Aufgabenstellung der vorliegenden Studienarbeit umfasste die Schritte von der Erarbeitung der theoretischen Grundlagen eines Data Mining-Prozesses über die Entwicklung und Umsetzung eines Konzeptes für eine problemgerechte Vorverarbeitung des gegebenen Datenbestandes bis zur Implementierung eines kompletten, das Problem der Musterfindung lösenden Programms in C++.

Das realisierte Programm ISIS (ISIS steht für „Intelligent Search of Interesting Patterns in Sequences“) wurde in zwei große Bereiche gegliedert: Der erste Bereich – die Datenvorverarbeitung – unterteilt den angebotenen Datenbestand in Cluster, die mit Hilfe einer Konfigurations Datenbank gebildet werden. Dadurch wird verhindert, dass Zusammenhänge zwischen Daten gesucht werden, die auf Grund gegebener Bedingungen nicht existieren können. Als ein weiterer positiver Effekt verringert sich die Komplexität und damit die Bearbeitungszeit der zu bewältigenden Aufgabe. Für die Cluster-Bildung wurde ein hierarchischer Ansatz gewählt, in dem die Alarmmeldungen der Netzelemente einer bestimmten Stufe, einschließlich denen ihrer Tochter-Elemente, einen eigenen Suchraum bilden.

Im zweiten Bereich – der Musterfindung – erfolgt die eigentliche Suche nach Zusammenhängen zwischen den Alarmmeldungen der Netzelemente. Dieser Teil stellt somit den Kernbereich der vorliegenden Arbeit dar. Es wurde die Möglichkeit geschaffen, Muster verschiedener Klassen zu suchen. Dazu gehören parallele Muster, deren gebildete Alarmgruppen sich ausschließlich über die Zugehörigkeit bestimmter Alarme definieren, sowie serielle Muster, bei denen zusätzlich die Reihenfolge der Alarme von Bedeutung ist. Für beide Musterarten kann weiterhin bestimmt werden, ob Alarme eines bestimmten Typs innerhalb einer Alarmgruppe mehrfach auftreten dürfen oder nicht.

Als eine Erkenntnis sowohl aus der Studie theoretischer Ausführungen über die Thematik des Data Mining, wie auch aus den auftauchenden Fragen bei der praktischen Umsetzung, ist die hohe Abhänigigkeit der Datenvorverarbeitung von den speziellen

Randbedingungen des späteren Einsatzgebietes zu nennen. Im völligen Gegensatz dazu steht der Prozess der Musterfindung. Die entwickelte Lösung ist in hohem Maße unabhängig von der Interpretation der zu verarbeitenden Daten. Damit ist es möglich, den Prozess der Musterfindung in nahezu unveränderter Form in verwandten Problemstellungen zum Einsatz kommen zu lassen.

Ausblick

Als eine primäre Aufgabe für die Weiterentwicklung von ISIS ist der Ausbau zu einem vollständigen Data Mining-Tool zu nennen. Zu diesem Zweck ist die Regelfindung und Ergebnis-Visualisierung als dritten und vierten großen Bereich eines Data Mining-Prozesses neben der Datenvorverarbeitung und der Musterfindung zu realisieren. Dabei erscheint es sinnvoll, den Prozess der Regelfindung auf die Gegebenheiten der verwendeten Alarmkorrelatoren zu optimieren. Die Visualisierung der Ergebnisse des gesamten Data Mining-Prozesses dient hingegen vorrangig zur Kontrolle durch die zuständigen Operateure in den Netzmanagement-Zentralen und ist daher an menschliche Bedürfnisse anzupassen.

Eine sekundäre Aufgabe betrifft die Optimierung der Entscheidung, ab wann eine Alarmgruppe als frequent anzusehen ist. Dieser Parameter hat sich in Experimenten – weit mehr als die Wahl der Fensterbreite – als sehr kritisch herausgestellt. Da von einer starken Beeinflussung des Endergebnisses durch den Frequenz-Schwellwert ausgegangen werden kann, verdient dieser Punkt in weiteren Entwicklungen und Untersuchungen erhöhte Aufmerksamkeit.

Der Prozess der Cluster-Bildung bietet einen weiteren Punkt für mögliche Optimierungen. Der realisierte Ansatz berücksichtigt die logische Struktur der Netzelemente. Die praktische Realisierung bestehender Mobilfunknetze weicht jedoch in vielfältiger Weise von dieser Struktur ab. Es ist daher zu vermuten, dass bei einer rein logischen Sichtweise auf die zu untersuchenden Elemente Informationen verloren gehen und somit bestimmte Korrelationen nicht gefunden werden können.

Algorithmenverzeichnis

Abkürzungsverzeichnis

BSC	Base Station Controller
BSS	Base Station Sub-System
BTS	Base Transceiver Station
DECT	Digital Enhanced Cordless Telecommunication
GSM	Global System for Mobile Communiction
MIB	Management Information Base
MIT	Management Information Tree
MS	Mobile Station
MSC	Mobile-Services Switching Center
NSS	Network and Switching Sub-System
OMC	Operation and Maintenance Center
OSS	Operation Sub-System
TMN	Telecommunication Management Network

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[DO74] Benjamin S. Duran, Patrick L. Odell
Cluster Analysis – A Survey
Seite 32–39
ISBN 3-540-06954-2
Springer Verlag 1974

[Alm92] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Bala Iyer, Arun Swami
An Interval Classifier for Database Mining Applications
IBM Almaden Research Center
1992

[Alm94] Rakesh Agrawal, Christos Faloutsos, Arun Swami
Efficient Similarity Search In Sequence Databases
IBM Almaden Research Center
March 4, 1994

[Sel94] Rüdiger Sellin
TMN – Die Basis für das Telekom-Management der Zukunft
ISBN 3-7685-4294-7
R. v. Decker's Verlag 1994

[HS94] M. Holzheimer, A.P.J.M. Siebes
Data Mining: The Search for Knowledge in Databases
Centrum voor Wiskunde en Informatica
CS-R9406 1994

[Wüth94] Beat Wüthrich
Knowledge Discovery in Databases
The Hong Kong University of Science and Technology
May 11, 1994

[Alm95] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, Kyuseok Shim
Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases
IBM Almaden Research Center
1995

[Alm96] Andreas Arning, Rakesh Agrawal, Prabhakar Raghavan
A Linear Method for Deviation Detection in Large Databases
IBM Almaden Research Center
1996

[Toi96] Hannu Toivonen
Discovery of Frequent Patterns in Large Data Collections
University of Helsinki
ISBN 951-45-7531-8
1996

[DGM97] Gautam Das, Dimitrios Gunopulos, Heikki Mannila
Finding Similar Time Series
University of Memphis and IBM Almaden Research Center and University of Helsinki
1997

[KJF97] Flip Korn, H.V. Jagadish, Christos Faloutsos
Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences
University of Maryland and AT&T Laboratories
1997

[MT97] Heikki Mannila, Hannu Toivonen
Levelwise search and borders of theories in knowledge discovery
University of Helsinki
1997

[Par97] Parchmann
Datenstrukturen
Universität Hannover
1997

[Pet97] Johann Petrak
Data Mining – Methoden und Anwendungen
Österreichisches Forschungsinstitut für Artificial Intelligence, Wien
1997

[Cech98] Markus Cech
Evaluation der Anwendungsmöglichkeiten von Data Mining-Algorithmen im Netzmanagement von Mobilfunknetzen
Universität Hannover
1998

[DLM98] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, Padhraic Smyth
Rule discovery from time series
University of Memphis and University of Helsinki and University of California at Irvine
1998

[Fer98] Reginald Ferber
Data Mining und Information Retrieval
Technische Universität Darmstadt
1998

[Navas98] Hugo F. Navas
Design und Implementierung eines Data Mining-Algorithmus für das Fault-Management in Mobilfunknetzen
Universität Hannover
1998

[RMS98] Sridhar Ramaswamy, Sameer Mahajan, Avi Silberschatz
On the Discovery of Interesting Patterns in Association Rules
Bell Labs and Informix Corp.
1998

[Sieg99] Gerd Siegmund
Technik der Netze
ISBN 3-7785-2637-5
Hüthig 1999

Index