

Institut für Informatik, Intelligente Systeme
Universität Stuttgart

Diplomarbeit Nr. 1444

**Evaluation und Erweiterung eines Verfahrens zum Finden von
Regelmässigkeiten in relationalen Datenbanken**

Escher, Stefan

Januar 1997

1	Einleitung	3
2	Grundlagen	6
2.1	KDD und Data - Mining.....	6
2.2	Aspekte von ILP-Algorithmen.....	6
2.3	Vergleich: Hornklausellogik und relationale Datenbanken.....	7
2.4	Beispiel	8
3	Beschreibung des vorhandenen Algorithmus.....	11
3.1	Definitionen	12
3.2	Ablauf des Algorithmus.....	17
3.2.1	Determine Test Order.....	19
3.2.2	Test Test Order	21
3.2.3	Offline Candidate Generation.....	22
3.2.4	Next Depth Literal.....	23
3.3	Abbildung von Klauseln auf SQL-Anfragen	23
4	Systemarchitektur	27
4.1	Übersicht über die Architektur.....	27
4.2	Interne Klauseldarstellung	28
4.3	Subsumption	29
4.4	Darstellung von Testordnungen.....	29
4.5	Klauselgenerierung.....	31
4.6	Datenbank-Schnittstelle.....	31
4.7	Benutzung des Algorithmus / Eingabedatei.....	32
5	Anwendung des Verfahrens auf einer relationalen Datenbank	34
5.1	Beispiel: Phonetische Datenbank	34
5.1.1	Format der phonetischen Datenbank	34
5.1.2	Erzeugte Prädikate	36
5.1.3	Rumpfliteralmenge	36
5.1.4	Gefundene Klauseln.....	37
5.2	Abbildung von Relationen auf Prädikate	37
5.3	Schwachpunkte des Algorithmus und Verbesserungsmöglichkeiten	40
6	Eine Komponente zur automatischen Intervallbestimmung	45
6.1	Literatur.....	46
6.2	Bestimmung von Intervallen	48
6.2.1	Integration in den bestehenden Algorithmus.....	48
6.2.2	Beispiel: Lautstärke und Frequenz	50
6.2.3	Mögliche Ansätze.....	53
6.2.4	Heuristische Berechnung von Intervallen	55
6.2.4.1	Berechnung von linken und rechten Grenzen.....	55
6.2.4.2	Berechnung von Intervallen	59
6.2.4.3	Berechnung von Hyper-Rechtecken	60
6.2.4.4	Parameter der numerischen Komponente	61
6.2.4.5	Algorithmus.....	62
6.2.4.6	Laufzeit	64
6.3	Versuch mit einer Diabetes-Datenbank	64

6.4 Zusammenfassung.....	65
7 Zusammenfassung und Ausblick.....	67

1

Einleitung

In den letzten Jahren wurden die Techniken zur Datenerhebung und Speicherung stark weiterentwickelt. Zum Beispiel führen Barcodes auf nahezu allen Produkten und die Automatisierung von Betriebsabläufen zu immer größeren Datenmengen, die interpretiert werden müssen. Das Problem liegt darin, daß eine große Menge von Information vorhanden ist, das darin enthaltene Wissen jedoch aufgrund der großen Datenmenge nicht zugänglich ist. Daraus ergibt sich die Notwendigkeit von automatischen Methoden zur Entdeckung von Wissen in großen Datenbanken (Knowledge Discovery in Databases, KDD).

Das aus einer Datenbank gewonnene Wissen kann vielseitig verwendet werden. Mit Data-Mining-Verfahren können Kundendatenbanken und damit das Käuferverhalten analysiert werden. So können aus Daten, die an Supermarktkassen erhoben werden, zum Beispiel sogenannte Association Rules [3] abgeleitet werden. Verfahren wie [3] arbeiten auf großen Tabellen über dem Wertebereich $\{0, 1\}$, sogenannten boolschen Datenbanken. Die Ausdrucksfähigkeit solcher Verfahren entspricht derjenigen der Aussagenlogik. Aufwendigere Verfahren benutzen Prädikatenlogik erster Ordnung zur Repräsentation des gefundenen Wissens. Dies ermöglicht die Formulierung von Beziehungen zwischen mehreren Relationen der Datenbank. Ein theoretisches Rahmenwerk für eine Teilmenge der prädikatenlogischen Systeme stellt das Inductive Logic Programming (ILP) dar.

Viele bisher existierende ILP-Systeme (FOIL [5], CLAUDIEN [6] [7]) sind in Prolog ohne direkte Schnittstelle zu einem relationalen Datenbanksystem implementiert. Ein praktisch einsetzbares System sollte jedoch wie RDT/DB [8] direkt mit relationalen Datenbanken auf SQL-Ebene kommunizieren.

Diese Diplomarbeit beschäftigt sich mit der Implementierung und Weiterentwicklung eines Verfahrens zum Finden von Hornklauseln in relationalen Datenbanken. Zum Beispiel könnte die Formel:

$$\text{jünger-als-18}(X) \wedge \text{besitzt}(X, Y) \wedge \text{Unfall}(Y) \rightarrow \text{großer - Schaden}(Y)$$

in einer Versicherungsdatenbank erkannt werden, wobei X vom Typ Kundename und Y ein Autotyp ist.

Eine solche Formel ist in einer realen Datenbank normalerweise nicht universell gültig. In der Regel wird die Aussage einer solchen Formel nicht von allen Tupeln einer Datenbank abgedeckt. Deshalb muß das Verfahren mit statistischen Mitteln berechnen, wie gut eine Klausel in der Datenbank erfüllt ist.

Grundlage des hier vorgestellten Verfahrens ist das angenäherte nichtmonotone ILP (Inductive Logic Programming). **ILP** ist ein aus dem Maschinellen Lernen bekanntes formales Rahmenwerk, in dem Bedingungen für gelernte prädikatenlogische Formeln formuliert sind (Vollständigkeit, Gültigkeit, Nichtredundanz). Im ILP werden zur Entdeckung von Klauseln positive und negative Beispiele für eine Formel benötigt. Das würde bedeuten, daß im Beispiel auch kleine Schäden explizit in einer Relation gespeichert sein müssen. Das **nichtmonotone ILP** arbeitet nur mit positiven Beispielen. Zur Erfassung der negativen Beispiele wird die Closed World Assumption angenommen.

Datenbanken enthalten immer Einträge, die nicht zur gefundenen Regelmäßigkeit passen. Diese können zum Beispiel durch Fehleingaben zustande kommen. Es ist aber auch möglich, daß Formeln nur auf einen Teil der Daten zutreffen, aber trotzdem interessant sind. Das **angenäherte nichtmonotone ILP** [1] verlangt nicht, daß eine erkannte Formel alle positiven Beispiele abdeckt. Das Akzeptanzkriterium für erkannte Formeln wird durch die Meßwerte **Support** und **Confidence** bestimmt..

Bisher existierende ILP-basierte Verfahren unterscheiden sich hauptsächlich in der Art, wie die Klauseln (Hypothesenklauseln) konstruiert werden.

RDT / DB [8] benutzt Metaprädikate, die vom Verfahren instantiiert werden. Die Eingabe für obiges Beispiele wäre

$$A(K) \wedge B(K, X) \wedge C(X, Y) \rightarrow D(Y)$$

RDT / DB instantiiert die Metaprädikate A, B, C, D so, daß obige Formel herauskommt. Die Struktur der zu erkennenden Formel muß dem Benutzer dabei schon vorweg bekannt sein.

CLAUDIEN [6] benutzt eine eigene Sprache DLAB zur Definition des Hypothesenraums. Die Definition für das obige Beispiel könnte folgendermaßen aussehen:

$$\{[\text{großer-Schaden}(X) \leftarrow 1..3 [\text{jünger-als-18}(X), \text{besitzt}(X, Y), \\ \text{Unfall}(Y), \text{nochwas}(X)]$$

Geprüft werden dann alle eins- bis dreielementigen Teilmengen der rechten Seite der Formel. Unter anderem auch die oben angegebene Formel.

Das hier beschriebene Verfahren benötigt eine Menge von Rumpfliteralen als Eingabe. Aus diesen konstruiert der Algorithmus Formeln, die als Rumpf Teilmengen dieser Rumpfliteralmenge haben. Eingabe für obiges Beispiel wären zum Beispiel die Rumpfliterale

jünger-als-18 (X)
 besitzt (X, Y)
 Unfall (Y)
 nochwas (X)

und das Kopfliteral

großer - Schaden (Y)

Selbstverständlich können noch viele weitere Literale in der Rumpfliteralmenge angegeben werden. Der Algorithmus sucht sich dann die Teilmengen heraus, die eine in der Datenbank gültige Formel bilden.

Zweck dieser Diplomarbeit ist unter anderem die Untersuchung des Algorithmus auf Verbesserungsmöglichkeiten. Dazu wird der Algorithmus auf eine phonetische Datenbank angewendet. Die dabei erkannten Schwächen und Verbesserungsmöglichkeiten werden beschrieben. Eine Verbesserungsmöglichkeit besteht in der Erweiterung des Algorithmus um ein Verfahren, das numerische Attributwerte behandelt. Im speziellen handelt es sich um ein Verfahren, das in der Lage ist, Beschränkungen an eine Formel mit numerischen Variablen anzufügen. Damit wäre dann eine Ausgabe der Form

großer - Schaden (Y) \leftarrow besitzt (X,Y), Alter (X, Z), $Z \in [18..21]$

möglich. Das Intervall [18..21] wird dabei vom Algorithmus automatisch bestimmt.

Kapitel 2 beschreibt die Grundlagen und den Prozeß des KDD, in den der Algorithmus eingebettet ist.

Kapitel 3 erklärt den von Irene Weber [2] entwickelten Algorithmus, der dieser Diplomarbeit zugrunde liegt.

In **Kapitel 4** wird die Architektur des implementierten Systems erklärt.

Kapitel 5 zeigt, wie der Algorithmus auf eine phonetische Datenbank angewendet wird, welche Ergebnisse erzielt wurden und welche Erweiterungsmöglichkeiten dabei zutage traten.

Kapitel 6 beschreibt, wie eine Behandlung von numerischen Attributen und Prädikaten in den Algorithmus integriert werden kann.

In **Kapitel 7** wird eine Beurteilung des bestehenden Systems und ein Ausblick für weitere Verbesserungen gegeben.

2

Grundlagen

2.1 KDD und Data - Mining

Der Begriff **Knowledge Discovery in Databases** bezeichnet den langen Prozeß, der notwendig ist, um Wissen in Daten zu finden. Dieser Prozeß beinhaltet den Einsatz von automatischen Methoden, genannt **Data-Mining**-Methoden. Zusätzlich gehören zum KDD-Prozeß auch manuelle Schritte, wie das Einbringen von Hintergrundwissen und die Interpretation der Ergebnisse.

Nach [FP] besteht der KDD-Prozeß aus folgenden Schritten:

1. **Verstehen des Anwendungsgebiets und der erwarteten Ergebnisse.**
2. **Auswahl der relevanten Relationen der Datenbank.**
3. **Vorverarbeitung der Daten**
4. **Entscheiden, welcher Data-Mining-Algorithmus in Frage kommt**
5. **Anwendung des Data-Mining Algorithmus.**
6. **Interpretation der Ergebnisse. Entfernung trivialer Ergebnisse. Ggf. Rückkehr zu Schritt 1-5.**

Der KDD-Prozeß ist iterativ und interaktiv. Der Benutzer bestimmt, bei welchem Schritt aufgesetzt wird, falls die Ergebnisse nicht zufriedenstellend sind.

In dieser Diplomarbeit geht es um die Weiterentwicklung eines Data-Mining Algorithmus. Das Thema der Diplomarbeit kann also bei Punkt 5 eingeordnet werden.

#

2.2 Aspekte von ILP-Algorithmen

Aufgabe des Algorithmus ist die Entdeckung von Mustern in der Datenbank. Diese Muster werden in der Hypothesensprache formuliert. Die hier verwendete

Hypothesensprache ist Hornklausellogik (eine Teilmenge der Prädikatenlogik erster Ordnung). Viele Data-Mining-Systeme verwenden als Hypothesensprache Entscheidungsbäume oder Association Rules, welche die Mächtigkeit der Aussagenlogik haben. Damit können nur Beziehungen zwischen den Attributen eines Tupels **einer** Relation dargestellt werden. Möchte man auch Regeln finden, die Beziehungen zwischen Attributen vieler Tupel in mehreren Relationen ausdrücken, braucht man Prädikatenlogik als Hypothesensprache. Diese Sprache ist komplexer, was eine aufwendigere Suchmethode notwendig macht.

Existierende ILP-Systeme erzeugen Formeln in einer bestimmten Reihenfolge, für die dann jeweils berechnet wird, ob sie dem Akzeptanzkriterium genügen. Wichtig ist, daß möglichst keine Formeln erzeugt werden, die nicht interpretierbar sind oder a priori ungültig sind. Bedingungen, die die Menge der möglichen Formeln verkleinern, also den Hypothesenraum einschränken, werden Bias genannt.

Ein weiterer wichtiger Aspekt eines Data-Mining-Algorithmus ist die Methode, mit der getestet wird, wie gut eine Formel das Akzeptanzkriterium des KDD-Prozesses erfüllt. In dem hier vorgestellten Algorithmus werden für eine Formel die Werte Support und Confidence aus der Datenbank berechnet. Dies geschieht mit Hilfe von SQL-Statements (select count ...).

Ein letzter Aspekt ist die Konstruktionsmethode, mit der Formeln erzeugt werden, die gegen die Datenbank getestet werden müssen. Dabei spielt zum Beispiel die Reihenfolge, in der die Formeln erzeugt werden, eine Rolle. Grundsätzlich sollten allgemeinere Formeln zuerst erzeugt werden. Erfüllt eine allgemeinere Formel das Akzeptanzkriterium nicht, brauchen alle Spezialisierungen dieser Formel nicht mehr erzeugt werden¹.

2.3 Vergleich: Hornklausellogik und relationale Datenbanken

Eine Datenbank besteht aus Relationen. Diese enthalten Tupel, die eine Teilmenge des kartesischen Produkts der Attributwertemengen aller Attribute sind. Tupel werden in der Logik als Fakten bezeichnet. Alle Tupel in der Datenbank drücken einen wahren Sachverhalt aus. Von allen Tupeln, die nicht in der Datenbank stehen, wird im nichtmonotonen ILP angenommen, daß sie einen falschen Sachverhalt ausdrücken. (Closed World Assumption).

Eine Datenbankrelation entspricht in der Logik einem Prädikat. In der Relation Kind(K, Y) ist die Extension des Prädikats Kind gespeichert. Die Attribute einer

¹ Dies gilt allerdings nur bei monotonen Akzeptanzkriterien. Das hier verwendete Akzeptanzkriterium Support ist monoton und wird in Kapitel 3 definiert.

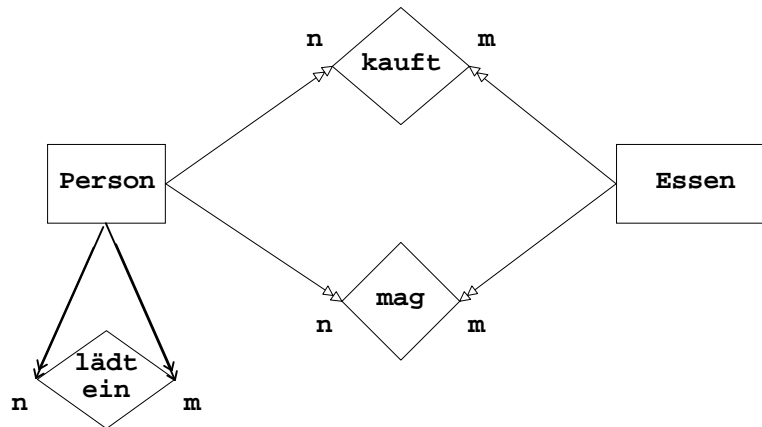
Relation heißen in der Logik Argumente. Hier zeigt sich ein Unterschied zwischen Datenbanken und Hornklausellogik. Den Attributen einer Relation sind Typen zugeordnet, während Argumente typfrei sind. [SD] gibt folgende Gegenüberstellung an.

<i>Datenbankterminologie</i>	<i>Logikterminologie</i>
Relationenname	Prädikatsymbol
Attribut einer Relation	Argument eines Prädikats
Tupel	Fakt
Tabelle	extensional definiertes Prädikat
View	intensional definiertes Prädikat

2.4 Beispiel

Der in dieser Diplomarbeit beschriebene Algorithmus versucht prädikatenlogische Formeln aufzustellen, die in der gegebenen Datenbank gültig sind. Die Form einer solchen Datenbank und mögliche Formeln sollen im folgenden illustriert werden. Das folgende Bild zeigt ein Relationenschema und das zugehörige Entity-Relationship Diagramm.

Abbildung 2.1 Beispiel für eine Datenbank



lädt_ein (Einladender, Eingeladener)

mag (wer, was)

kauft (wer, was)

In diesem speziellen Datenbankschema existieren nur n:m - Beziehungen. Dies ist in realen Datenbanken nicht üblich. Der Einfachheit halber soll dieses Beispiel für eine erste Motivation ausreichen.

Man kann in diesem Beispiel alle Relationen als Prädikate auffassen. Die entsprechenden Tabellen enthalten die Extension der Prädikate. So enthält die Tabelle

zu $\text{mag}(\text{wer}, \text{was})$ alle Paare von Person und Essen, so daß die Person das Essen mag. Die Extensionen der Prädikate können folgende Gestalt haben.

Abbildung 2.2

lädt ein	
Einladender	Eingeladener
Tom	Anne
Tom	Bill
Anne	Bill

mag	
wer	was
Tom	Erdnüsse
Anne	Erdnüsse
Bill	Cola
Bill	Erdnüsse
Anne	Kaviar

kauft	
wer	was
Tom	Erdnüsse
Tom	Cola
Anne	Erdnüsse
Anne	Wein

Angenommen, der Manager eines Supermarktes hätte eine solche Datenbank gegeben. Für ihn wäre es von großem Interesse, was jemanden dazu bewegt, ein bestimmtes Produkt zu kaufen. Seine Frage an den Data-Mining Algorithmus wäre also „Wovon hängt es ab, daß der Kunde X das Produkt Y kauft ?“. Er sucht also nach einer Formel der Form:

$$\text{kauft}(X,Y) \leftarrow ???$$

Die Aufgabe des Algorithmus besteht darin, die drei Fragezeichen durch eine logische Teilformel zu ersetzen. Der hier beschriebene Algorithmus ist aus Gründen der Semi-Entscheidbarkeit und der einfachen Interpretierbarkeit auf Hornklausellogik beschränkt. Das bedeutet, daß für die Fragezeichen eine Konjunktion von Literalen bestimmt werden soll. Diese Konjunktion wird Rumpf der Klausel genannt. Ist eine solche Klausel in der Datenbank erfüllt, wird sie Hypothesenklausel genannt. Wird die Menge der möglichen Rümpfe nicht genauer spezifiziert, kommen dafür unendlich viele in Betracht. Die Rümpfe können beliebig lang sein und prinzipiell beliebig viele verschiedene Variablen und Konstanten enthalten. Das Verfahren soll jedoch nach endlicher Zeit abbrechen. Das bedeutet, daß man Mechanismen braucht, die die

Anzahl der möglichen Hypothesenklauseln (den Hypothesenraum) möglichst klein hält. Dies kann zum Beispiel durch explizite Angabe der möglichen Rumpfliterale durch den Benutzer geschehen.

Der Manager könnte zum Beispiel folgende Rumpfliteralmenge spezifizieren:

lädt_ein(X,Z)
lädt_ein(Z,X)
mag(Z,Y)
kauft(X,Y)

Eine sinnvolle Hypothesenklausel wäre dann:

$\text{kauft}(X,Y) \leftarrow \text{lädt_ein}(X,Z), \text{mag}(Z,Y)$

Zusätzlich zu interessanten Klauseln werden auch solche gefunden, die kaum Informationsgehalt haben. Zum Beispiel Tautologien:

$\text{kauft}(X,Y) \leftarrow \text{kauft}(X,Y)$

Diese Formel wird zwar von der Datenbank erfüllt, ist aber vom Benutzer nicht erwünscht. Deshalb ist es immer notwendig, die gefundenen Hypothesenklauseln manuell nachzusortieren.

Das nächste Kapitel beschreibt den Algorithmus formal.

3

Beschreibung des vorhandenen Algorithmus

Der im folgenden beschriebene Data-Mining Algorithmus wurde in seiner Grundform von Irene Weber [1] [2] von dem Algorithmus zur Erkennung von Association Rules [3] abgeleitet und im Rahmen dieser Diplomarbeit weiterentwickelt.

Ziel des Algorithmus ist die Erkennung von Hornklauseln erster Ordnung in einer relationalen Datenbank. Die theoretische Grundlage dafür ist das nichtmonotone ILP¹. Im Gegensatz zu Algorithmen, die aussagenlogische Formeln suchen [3], muß die Datenbank nicht mit Attributwertbereich $\{0, 1\}$ vorliegen. Das bedeutet, die Datenbank muß nicht in einer großen Tabelle organisiert sein, sondern kann normalisiert in mehreren Tabellen vorliegen.

Eine teure Operation eines KDD-Systems ist der Test, ob eine Klausel das Akzeptanzkriterium erfüllt. Dafür müssen in diesem Algorithmus die Meßwerte Support und Confidence berechnet werden. Dazu müssen Datensätze in der Datenbank gezählt werden. Aus Effizienzgründen ist es wichtig, daß nur solche Klauseln gegen die Datenbank getestet werden, von denen man nicht schon vorher weiß, daß sie das Akzeptanzkriterium nicht erfüllen.

Deshalb spielt die Anzahl der Klauseln, die gegen die Datenbank getestet werden eine wichtige Rolle. In dieser Version des Algorithmus wird diese Kandidatenmenge durch folgende Beschränkungen reduziert:

- Offline Candidate Generation
- Berücksichtigung der Tiefe eines Literals in einer Klausel
- Berücksichtigung der Θ -Subsumptionsbeziehung zwischen Klauseln

Diese Beschränkungsmechanismen werden nach einigen Definitionen in diesem Kapitel beschrieben.

Die einschränkenden Bedingungen, die den Hypothesenraum möglichst klein halten, werden als Bias bezeichnet. Der Bias wird in diesem Algorithmus durch Angabe der Rumpfliterale spezifiziert.

Der Algorithmus kann auf existierende relationale Datenbanken angewendet werden. Das setzt voraus, daß der Test von Support und Confidence mit Hilfe von SQL-

¹ Inductive Logic Programming

Anfragen geschehen kann. Im folgenden werden mögliche Berechnungsvorschriften für diese Meßwerte definiert. Etwas später wird gezeigt, wie diese auf SQL-Anfragen abgebildet werden können.

3.1 Definitionen

Der Algorithmus sucht nach funktionsfreien Hornklauseln, die bereichsbeschränkt sind und deren Rumpfliterale mit dem Kopf verbunden und positiv sind.

Definition 3.1 positives funktionsfreies Literal

Ein positives funktionsfreies Literal hat die Form

$$P(x_1, \dots, x_n),$$

wobei P ein Prädikatsymbol ist und x_1, \dots, x_n Konstanten oder Variablen sein können.

Definition 3.2 funktionsfreie Hornklausel

Eine funktionsfreie Hornklausel hat die Form

$$l_{i_1} \wedge \dots \wedge l_{i_m} \rightarrow h,$$

wobei $l_{i_1}, \dots, l_{i_m}, h$ funktionsfreie Literale sind. h wird als Kopf der Klausel bezeichnet, die Konjunktion $l_{i_1} \wedge \dots \wedge l_{i_m}$ ist der Rumpf der Klausel. Die Funktion $head(c)$ hat den Kopf einer Klausel c zum Ergebnis, die Funktion $body(c)$ ihren Rumpf.

Definition 3.3 Bereichsbeschränktheit

Eine Hornklausel ist bereichsbeschränkt, wenn alle Variablen im Kopf der Klausel auch im Rumpf auftauchen:

$$x \in head(c) \rightarrow x \in body(c)$$

Definition 3.4 verbundenes Literale

Ein Literal l ist in einer Hornklausel verbunden, falls

- (1) eine seiner Variablen im Kopf vorkommt: $\text{var } s(l) \cap \text{var } s(h) \neq \emptyset$, oder
- (2) eine seiner Variablen in einem anderen Rumpfliteral vorkommt, das mit dem Kopf verbunden ist.

Zwei Literale sind miteinander verbunden, wenn sie gemeinsame Variablen besitzen. Ein Literal ist in einer Klausel verbunden, wenn es eine Verbindung über andere Literale bis hin zum Kopf gibt.

Beispiel 3.1 Verbundene Literale

Das Literal $\text{mag}(Z,Y)$ ist in der Klausel

$$\text{kauft}(X,Y) \leftarrow \text{lädt_ein}(X,Z), \text{mag}(Z,Y)$$

verbunden, denn es ist mit $\text{lädt_ein}(X,Z)$ über die Variable Z verbunden. $\text{lädt_ein}(X,Z)$ ist wiederum über die Variable X mit dem Kopf verbunden.

Bei der Generierung von Klauseln spielt die Tiefe von Variablen eine wichtige Rolle.

Definition 3.5 Tiefe von Variablen

Die Tiefe $\text{depth}(v)$ einer Variable v bezüglich eines Klauselkopfes h und eines Rumpfes B ist

$$\text{depth}(v) = \begin{cases} 0 & \text{falls } v \text{ in } h \text{ vorkommt} \\ \min\{\text{depth}(u) \mid u, v \text{ stehen im gleichen Literal in } B\} + 1 & \text{sonst} \end{cases}$$

Definition 3.6 Tiefe von Literalen

Die Tiefe $\text{depth}(l)$ eines Literals l bezüglich eines Klauselkopfes h und eines Rumpfes B ist

$$\text{depth}(l) = \max\{\text{depth}(v) \mid v \in \text{vars}(l)\}$$

Definition 3.7 Tiefe von Klauseln

Die Tiefe $\text{depth}(c)$ einer Klausel c mit Klauselkopf h und Rumpfliteralmenge B ist

$$\text{depth}(c) = \max\{\text{depth}(l) \mid l \in B\}$$

Die Tiefe gibt an, über wieviele andere Literale ein Literal mit dem Kopf verbunden ist.

Beispiel 3.2 Tiefe von Literalen

Gegeben sei die Klausel

$$h(X) \leftarrow p(X), q(X,Y), q(Y,Z), r(W)$$

Dann hat das Literal $p(X)$ die Tiefe 0, weil alle Variablen von $p(X)$ auch in $h(X)$ auftauchen. Das Literal $q(X,Y)$ hat die Tiefe 1, weil es eine Variable der Tiefe 0 (X) und eine Variable der Tiefe 1 (Y) enthält. Das Literal $q(Y,Z)$ hat die Tiefe 2, weil die Variable Z die Tiefe 2 hat. $r(W)$ hat eine undefinierte Tiefe, das Literal kann nicht mit dem Kopf verbunden werden. Literale mit undefinierter Tiefe können aus der Menge möglicher Rumpfliterale entfernt werden.

Literale mit Tiefe n können in einer Klausel mit einer kleineren Länge als n nicht verbunden sein. Deshalb können verbundene Klauseln mit Länge n keine Literale der Tiefe größer n enthalten.

Grundlage für den Algorithmus ist das nichtmonotone ILP. Eine Erweiterung um die Behandlung von Klauseln, die nicht in der ganzen Datenbank gelten, ist das angenäherte nichtmonotone ILP. Im folgenden werden diese Regelwerke definiert.

Die Datenbank besteht aus einer Menge von Fakten. Das zu der Datenbank D gehörige Modell wird mit $M(D)$ bezeichnet. Gesucht werden Hornklauseln c , die im Datenbankmodell gültig sind: $M(D) \models c$. Diese Klauseln werden Hypothesenklauseln genannt. Die Menge aller gültigen Klauseln heißt Hypothese. Das nichtmonotone ILP stellt folgende Anforderungen an Hypothesenklauseln. Die Hypothesensprache dient zur Erzeugung des Hypothesenraums und kann deshalb alle Klauseln erzeugen, die in der Hypothese auftauchen können.

Definition 3.8 nichtmonotones ILP

Gegeben ist eine Datenbank D und eine Hypothesensprache L_H .

Gesucht ist eine Hypothese H mit:

- | | |
|------------------------------------------------------------------------------------|-------------------|
| (1) $\forall h \in H: D \text{ erfüllt } h$ | (Gültigkeit) |
| (2) $\forall h \in L_H: D \text{ erfüllt } h \Rightarrow H \models h$ | (Vollständigkeit) |
| (3) $\neg \exists H' \subset H: H' \text{ vollständig und } M \text{ erfüllt } H'$ | (Nichtredundanz) |

Die Erfüllbarkeit besagt, daß jede Klausel der Hypothese in der Datenbank gültig sein muß. Vollständigkeit ist eine Eigenschaft, die gewährleistet, daß alle Klauseln, die in der Datenbank gültig sind auch in der Hypothese enthalten sind, oder zumindest aus der Hypothese folgen. Nichtredundanz verhindert, daß Klauseln in der Hypothese sind, die sowieso schon logisch aus den anderen Hypothesenklauseln folgen.

Das nichtmonotone ILP verlangt, daß eine Hypothesenklausel in der gesamten Datenbank gültig ist. Dies ist in realen Datenbanken kaum gegeben. Fehleingaben

können zu Inkonsistenzen führen. Außerdem sind durchaus auch Klauseln von Interesse, die nicht von der ganzen Datenbank, sondern nur von einem großen Teil davon, erfüllt werden.

Andererseits kann es passieren, daß eine Klausel nur von einem einzigen Datensatz erfüllt und von keinem anderen widersprochen wird. Diese Klausel würde als Hypothesenklausel akzeptiert, was nicht unbedingt erwünscht ist.

Deshalb muß die Bedingung dafür, daß eine Klausel als Hypothese akzeptiert wird, etwas gelockert werden. Dies geschieht hier mit den Werten Support und Confidence, die in dieser Version des Algorithmus als Akzeptanzkriterium fungieren. $\|c\|$ bezeichnet die Extension der Klausel c , das heißt, die Menge der Tupel in der Datenbank, für die c erfüllt ist.

Definition 3.9 Support

Der Support einer Klausel berechnet sich als

$$\text{sup}(c) = \begin{cases} 0 & \text{falls } |\text{head}(c)| = 0 \\ \frac{\| \text{body}(c) \wedge \text{head}(c) \|_{\text{vars}(\text{head}(c))}}{\| \text{head}(c) \|} & \text{sonst} \end{cases}$$

$\text{vars}(l)$ bezeichnet dabei die Menge der Variablen im Literal l .

Der Support ist groß, wenn nur wenige Kopfinstanzen nicht von der Klausel abgedeckt werden. Der Support ist also ein Maß für die Menge der Kopfinstanzen, die von der Klausel abgedeckt werden.

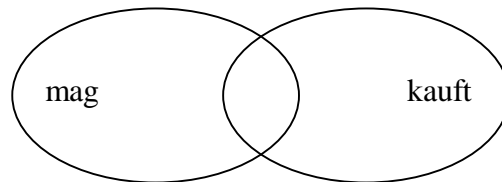
Beispiel 3.3 Support

Die Formel

$$\text{mag}(X, Y) \leftarrow \text{kauft}(X, Y)$$

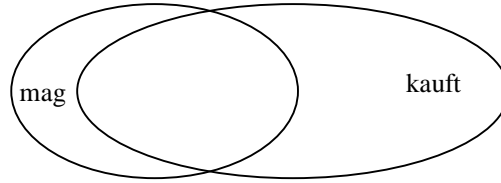
soll ausdrücken, daß wenn der Kunde X das Produkt Y kauft, er es auch gerne mag.

Diese Formel soll auf einen großen Teil der Kunden zutreffen, die das Produkt mögen. Der Support kann dann folgendermaßen illustriert werden:



Die Ellipsen bezeichnen die Extensionen der Prädikate mag , bzw. kauft . Die Schnittmenge bezeichnet die Extension von $\text{mag} \wedge \text{kauft}$, also den Zähler des Support.

In diesem Fall ist der Support klein, nur wenig Kunden, die das Produkt mögen, kaufen es auch.



Hier ist der Support groß. Es gibt nur wenig Kunden, die das Produkt mögen, es aber nicht kaufen.

Eine wichtige Eigenschaft des Support ist die **Monotonie**. Das bedeutet, daß der Support einer spezielleren¹ Klausel nie größer als der Support einer allgemeineren Klausel sein kann.

$$c_1 \rightarrow c_2 \Rightarrow \text{sup}(c_1) \geq \text{sup}(c_2)$$

Definition 3.10 Confidence

Die Confidence einer Klausel berechnet sich als

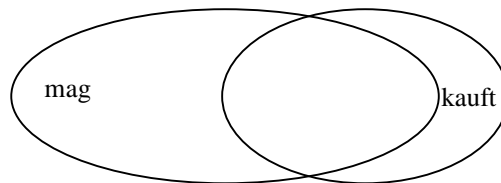
$$\text{conf}(c) = \begin{cases} 0 & \text{falls } \|body(c)\|_{\text{vars}(head(c))} = 0 \\ \frac{\|body(c) \wedge head(c)\|_{\text{vars}(head(c))}}{\|body(c)\|_{\text{vars}(head(c))}} & \text{sonst} \end{cases}$$

vars(l) bezeichnet dabei die Menge der Variablen im Literal l.

Die Confidence ist groß, wenn viele Tupel, die den Rumpf der Klausel erfüllen auch den Rumpf und den Kopf erfüllen. Das heißt, wenn nur wenig Kopftupel, die die Klausel gültig machen würden, in der Datenbank fehlen. Die Confidence ist ein Maß dafür, wie stark der Kopf aus dem Rumpf folgt.

Beispiel 3.4 Confidence

Die Formel aus Beispiel 3.3 soll für einen großen Teil der Kunden stimmen, die das Produkt kaufen.



¹ Eine Klausel c₁ ist spezieller als eine Formel c₂, wenn c₂ aus c₁ folgt (c₂ → c₁).

In diesem Fall ist die Confidence hoch. Viele Kunden, die das Produkt kaufen, mögen es auch.

Die Meßwerte Support und Confidence erlauben eine weniger exakte Definition der Erfüllbarkeit.

Definition 3.11 ((σ, γ) -Erfüllbarkeit)

Eine Klausel ist (σ, γ) -erfüllbar, wenn sie den Support $\geq \sigma$ und die Confidence $\geq \gamma$ erreicht.

Im angenäherten nichtmonotonen ILP gilt es folgende Klauseln in einer Datenbank zu finden:

Definition 3.12 angenähertes nichtmonotones ILP

Gegeben ist eine Datenbank D, eine Hypothesensprache L_H und die gewünschten Support und Confidence - Werte $\sigma, \gamma \in [0,1]$.

Gesucht ist eine Hypothesenmenge H mit:

- | | |
|--------------------------------------------------------------------------------------------------|----------------------------------|
| (1) $\forall h \in H: D(\sigma, \gamma) - \text{erfüllt } h$ | $((\sigma, \gamma)$ -Gültigkeit) |
| (2) $\forall h \in L_H: D(\sigma, \gamma) - \text{erfüllt } h \Rightarrow H \not\models h$ | (Vollständigkeit) |
| (3) $\neg \exists H' \subset H: H' \text{ vollständig und } (\sigma, \gamma) - \text{Erfüllbar}$ | (Nichtredundanz) |

Hier wird nicht mehr erwartet, daß Klauseln von der ganzen Datenbank erfüllt werden. Um in die Hypothese aufgenommen zu werden müssen Klauseln nur die geforderten Support- und Confidence-Werte erreichen.

3.2 Ablauf des Algorithmus

Im wesentlichen gibt der Benutzer des Algorithmus eine Menge von möglichen Rumpfliteralen an. Alle Rumpfe von Hypothesenklauseln sind Teilmengen dieser Rumpfliteralmenge. Der Algorithmus beginnt bei kleinen Klauseln (1 Rumpfliteral). Diese Klauseln werden dann in einer sinnvollen Reihenfolge gegen die Datenbank getestet. Alle Klauseln die zum Test anstehen werden Kandidatenklauseln genannt. Kandidatenklauseln können als Hypothese akzeptiert werden, verworfen werden oder zur Bildung neuer (längerer) Kandidatenklauseln zur Verfügung gestellt werden. Aus den letzten Klauseln werden neue Klauseln generiert und der Prozeß beginnt von vorne.

Der Algorithmus hat eine große while-Schleife. Diese Schleife iteriert über die Länge von Klauseln, das heißt über die Anzahl der Rumpfliterale.

Eingabe für den Algorithmus ist eine Menge von Rumpfliteralen B . Als Hypothesenklauseln kommen nur diejenige in Frage, deren Rumpf mit dem Kopf verbunden ist. Deshalb ist es im ersten Schritt nur sinnvoll, solche Literale zu berücksichtigen, die direkt mit dem Kopf verbunden sind, also die Tiefe 1 haben.

Aus jedem eingegebenen Rumpfliteral der Tiefe 1 wird im ersten Schritt eine Klausel mit diesem Literal als Rumpf und dem eingegebenen Kopfliteral als Kopf gebildet. Aus der Menge B_1 aus möglichen Rumpfliteralen wird also eine erste Kandidatenmenge C_1 erzeugt. Der Index 1 bezeichnet dabei die Länge des Klauselrumpfs.

$$C_1 := \{ h \leftarrow b \mid b \in B \}$$

Die Hypothese ist zu Anfang leer.

$$H := \emptyset$$

i bezeichnet im folgenden die Länge des Rumpfs von Kandidatenklauseln. Das heißt, die Anzahl der Literale, die der Rumpf enthält. Die Kandidatenmenge der Länge 1 ist bereits initialisiert.

$$I := 1$$

Als nächstes werden solange neue Kandidaten generiert, bis dies nicht mehr möglich ist. Die Suche nach Klauseln endet damit, daß eine leere Kandidatenmenge generiert wird.

WHILE $C_i \neq \emptyset$ *DO*

Falls eine Klausel c den nötigen Support nicht erreicht, müssen alle Klauseln c' , die spezieller sind (d.h. c subsumiert c' , siehe Abschnitt 3.2.1) nicht mehr getestet werden. Deshalb muß eine Testordnung erzeugt werden, die die Reihenfolge in der die Klauseln getestet werden festlegt. Diese Testordnung besteht aus Klassen von Klauseln. Sie ist in Abschnitt 3.2.1 ausführlich beschrieben.

$$C_i^{ord} := \text{determine_test_order}(C_i)$$

Jetzt wird jede in C_i^{ord} enthaltene Klasse gegen die Datenbank getestet. Dabei wird die Reihenfolge der Klauseln gemäß der Subsumption berücksichtigt. Wie dies von statten geht beschreibt Abschnitt 3.2.2. Alle Klauseln mit zu kleinem Support werden verworfen.

Für jede Klausel mit ausreichendem Support wird jetzt der Confidence-Wert berechnet. Falls er groß genug ist wird die Klausel in die Hypothese übernommen.

Aus Hypothesenklauseln können nur noch speziellere Klauseln generiert werden. Deshalb werden diese nicht mehr zur Generierung neuer Klauseln benutzt. L und H fungieren als Referenzparameter.

$$test_test_order (C_i^{ord} , H, L)$$

In der Menge L stehen jetzt alle Klauseln der Länge i , deren Support groß genug ist, deren Confidence aber zu klein ist. Durch Hinzufügen von neuen Rumpfliteralen kann sich die Confidence erhöhen, deshalb werden jetzt Klauseln der Länge $i+1$ erzeugt. Die Funktionen sind in den Abschnitten 3.2.3 und 3.2.4 genau beschrieben.

$$C_{i+1} = offline_cand_gen(L) \cup next_depth_literals(L, B_{i+1})$$

Die Länge der Klauseln ist jetzt um eins größer:

$$i := i + 1$$

OD;

Zurückgegeben werden alle Hypothesenklauseln

RETURN H;

3.2.1 Determine Test Order

Der Support-Wert ist monoton, das heißt er steigt bei spezielleren Klauseln nicht. Das hat zur Folge, daß alle Klauseln, die spezieller sind als eine Klausel mit zu kleinem Support, nicht getestet werden müssen. Diese hätten auf jeden Fall keinen ausreichenden Support. Die Frage, ob eine Klausel spezieller ist als eine andere kann auf syntaktischer Ebene beantwortet werden. Die hier relevante Relation zwischen Klauseln wird Θ -Subsumption genannt.

Definition 3.13 Θ -Subsumption

Eine Klausel c_1 subsumiert eine Klausel c_2 , falls

$$\exists \theta: c_1 \theta \subseteq c_2$$

wobei θ eine Substitution ist. Man sagt auch: c_1 ist allgemeiner als c_2 .

Beispiel 3.5 Subsumption

Die Klausel $H(X) \leftarrow P(X,Y,Z)$ ist allgemeiner als $H(X) \leftarrow P(X,X,Z)$, denn es existiert eine Substitution $[X/X][Y/X][Z/Z]$, so daß

$$H(X) \leftarrow P(X,Y,Z) [X/X][Y/X][Z/Z] = H(X) \leftarrow P(X,X,Z)$$

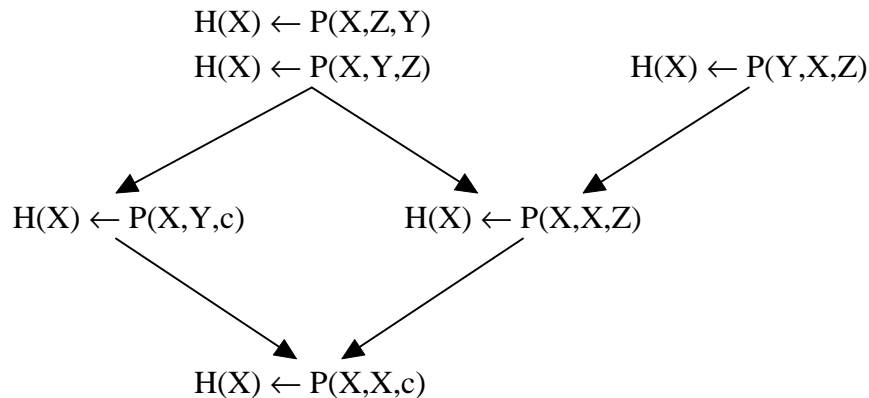
Mit Hilfe der Subsumptionsbeziehung ergibt sich eine Ordnung auf der Menge von Klauseln. Es gibt Klauseln, die äquivalent sind, solche, die durch Subsumption vergleichbar sind und solche, die Unvergleichbar gemäß der Subsumption sind. Dies induziert eine Halbordnung auf der Menge der Klauseln. Diese Halbordnung wird im folgenden Testordnung genannt.

Beispiel 3.6 Struktur in Testordnungen

Die Struktur, die aus den Klauseln

- $H(X) \leftarrow P(X,Z,Y)$
- $H(X) \leftarrow P(Y,X,Z)$
- $H(X) \leftarrow P(X,Y,c)$
- $H(X) \leftarrow P(X,X,Z)$
- $H(X) \leftarrow P(X,X,c)$

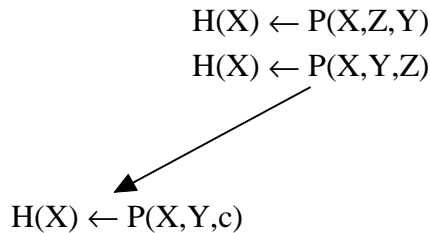
entsteht hat folgende Form:



Der Pfeil drückt dabei die Relation \geq_s aus. Zwei übereinander stehende Klauseln bedeuten, daß sich die Klauseln gegenseitig subsumieren.

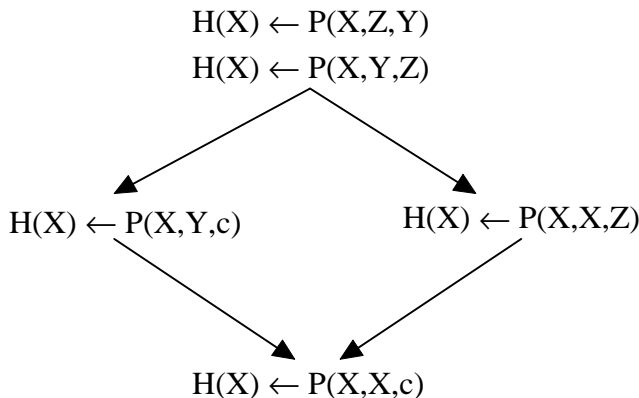
3.2.2 Test Test Order

Nach der Erzeugung einer Testordnung muß diese in der richtigen Reihenfolge durchgetestet werden. Dazu wird für ein beliebiges oberstes Element der Testordnung der Support-Wert berechnet. Entscheidend ist, daß von einer Klasse der Testordnung nur eine Klausel getestet werden muß, die anderen Klauseln der Klasse haben gleiche Support und Confidence. Ist der Support zu klein, wird die Klausel und alle von ihr subsumierten Klauseln verworfen. Angenommen, die Klausel $H(X) \leftarrow P(Y,X,Z)$ hätte zu wenig Support, dann würde die Testordnung anschließend folgendermaßen aussehen:



Die fehlenden Klauseln werden endgültig gelöscht, sie werden nicht mehr benötigt. Das gleiche Aussehen hat die Testordnung, wenn der Support und die anschließend berechnete Confidence ausreichend war. Der einzige Unterschied ist, daß eine Klausel aus der getesteten obersten Klasse in die Hypothese aufgenommen wird. Auch Spezialisierungen von Hypothesenklauseln werden nicht mehr gebraucht (sie wären redundant in der Hypothese).

Ein anderes Aussehen hat die Testordnung, falls der Support groß genug ist, die Confidence aber zu klein. Es ist durchaus möglich, daß eine speziellere Klausel die erforderliche Confidence hat. Deshalb darf nur die getestete Klauselklasse aus der Testordnung entfernt werden. Damit ergibt sich folgendes Bild, wenn $H(X) \leftarrow P(Y,X,Z)$ eine entsprechende Klausel ist.



Als nächstes stünde dann ein Element der obersten Klasse zum Test an. Es ist durchaus möglich, daß durch die Entfernung einer Klasse neue oberste Klassen entstehen. Dann kann eine beliebige Klasse als nächstes getestet werden. Auf die Berechnung der Support- und Confidence-Werte wurde bisher noch nicht näher eingegangen. Dies geschieht in Abschnitt 3.3.

3.2.3 Offline Candidate Generation

Die Generierung von Kandidatenklauseln aus Klauseln, die den Confidence-Wert nicht erreichen, erfolgt in zwei Schritten. Zuerst werden aus den bekannten Klauseln mit Länge i und maximaler Tiefe i neue Kandidaten der Länge $i+1$ und maximaler Tiefe i generiert.

Offline_cand_gen (L: Klauselmenge, n:Anzahl_Rumpfliterale): Kandidatenmenge

In der Menge L werden alle Paare von Rümpfen gesucht, die sich nur in einem Literal unterscheiden. Die Rümpfe werden zu einem Rumpf vereinigt. Dieser ist um genau um ein Literal länger als die ursprünglichen Rümpfe. Die maximale Tiefe der Rumpfliterale wächst nicht, weil keine Literale mit höherer Tiefe hinzugenommen wurden.

Der generierte Rumpf der Länge $n+1$ ist nur ein geeigneter Kandidat, falls alle um eins kürzeren verbundenen Teilmengen seines Rumpfes in L vorhanden sind, also den erforderlichen Support erreicht haben. Ist eine Teilmenge nicht in L vorhanden, ist die generierte Klausel eine Spezialisierung einer Klausel, die zu wenig Support hatte. Solche Klauseln können sofort verworfen, bzw. müssen gar nicht generiert werden.

$$C := \left\{ h \leftarrow body(c_1) \cup body(c_2) \mid \begin{array}{l} c_1, c_2 \in L \wedge |body(c_1) \cap body(c_2)| = n - 1 \wedge \\ \forall b \subset (body(c_1) \cup body(c_2)): \\ (h \leftarrow b \text{ verbunden}) \wedge |b| = n - 1 \Rightarrow h \leftarrow b \in L \end{array} \right\}$$

RETURN C;

Beispiel 3.7 offline candidate generation

Die Menge L enthalte folgende Rümpfe:

$$B(X,Y), C(Y,Z)$$

B(X,Y), D(Y,W)
D(Y,W), C(Y,Z)

Aus den ersten beiden Rümpfen kann der neue Rumpf

B(X,Y), C(Y,Z), D(Y,W)

generiert werden. In der obigen Menge L sind alle zweielementigen Teilmengen des Rumpfes enthalten, das heißt die generierte Klausel ist ein neuer Testkandidat.

In der Menge L befinden sich nur Rümpfe von Klauseln mit ausreichendem Support. Wäre D(Y,W), C(Y,Z) nicht in L, hätte die generierte Klausel bestimmt keinen ausreichenden Support (weil sie eine Spezialisierung von D(Y,W), C(Y,Z) ist).

3.2.4 Next Depth Literal

Mit offline candidate generation werden nur aus bereits in Klauseln der Länge n enthaltenen Literalen neue Rümpfe konstruiert. Diese haben höchstens die Tiefe n.

Um alle Kandidaten der Länge $n+1$ zu generieren müssen zu den bisher generierten Klauseln noch die Rumpfliterale hinzugefügt werden, die in diesen Klauseln die Tiefe $n+1$ haben und verbunden sind. Die Tiefe der gesamten Klausel erhöht sich dadurch auf $n+1$.

$$N_{i+1} := \left\{ h \leftarrow b_1, \dots, b_{n+1} \mid h \leftarrow b_1, \dots, b_n \in L \wedge b_{n+1} \in B_{i+1} \wedge b_{n+1} \text{ verbunden} \right\}$$

RETURN N_{i+1} ;

3.3 Abbildung von Klauseln auf SQL-Anfragen

Der Algorithmus soll direkt auf großen relationalen Datenbanken operieren. Durch SQL-Anfragen wird getestet, ob die Klausel die spezifizierten Support- und Confidence-Werte erreicht. Grundlage für die Abbildung von Klauseln auf SQL-Anfragen ist die Tatsache, daß eine logische und-Vernüpfung zwischen zwei Literalen einem relationalen Gleichverbund (join) der zugehörigen Relationen entspricht. Der Verbund geht dabei über die Attribute, die in der jeweiligen Relation für die gemeinsamen Variablen der Literale stehen. Zur formalen Definition der Abbildung werden folgende Funktionen benötigt:

L sei die Menge aller Literale, V die Menge aller Variablen, R die Menge aller Relationen und P die Menge aller Prädikate.

1. Eine Funktion $attr: L \times V \rightarrow A$, die einer Variablen in einem Literal das zugehörige Attribut zuordnet.

2. Eine Funktion $rel: L \rightarrow R$, die jedem Literal dessen Relation zuordnet.

Die Werte Support und Confidence werden aus der Anzahl der Instanzen eines Prädikats berechnet. Die Instanzen stehen in der Datenbank. Dies erfordert einen Datenbankzugriff bei der Berechnung dieser Werte. Der Zugriff erfolgt über SQL-Anfragen, die aus einer Klausel generiert werden. Eine Klausel muß auf folgende SQL-Anfragen abgebildet werden:

1. Eine SQL-Anfrage, die den gemeinsamen Zähler des Support- und Confidence-Werts berechnet.
2. Eine SQL-Anfrage, die den Nenner des Support-Werts berechnet.
3. Eine SQL-Anfrage, die den Nenner des Confidence-Werts berechnet.

Zur Berechnung des Support-Werts muß der Zähler

$$\|body(c) \wedge head(c)\|_{\text{vars}(head(c))}$$

berechnet werden. Dieser Ausdruck hat ausgeschrieben die Form

$$\|body_1(c) \wedge \dots \wedge body_n(c) \wedge head(c)\|_{\text{vars}(head(c))},$$

wobei $body_i(c)$ das i -te Rumpfliteral der Klausel ist.

Die Berechnung geschieht mit einer SQL-Anfrage der Form

$$\begin{array}{l}
 \text{SELECT COUNT (DISTINCT } \left\{ R.A \left| \begin{array}{l} \exists X: \\ A = attr(head(c), X) \wedge \\ R = rel(head(c)) \end{array} \right. \right\} \\
 \text{FROM } rel(pred(body_1(c))), \dots, rel(body_1(c)), rel(head(c)) \\
 \text{WHERE AND } \left(\left(\left(\begin{array}{l} \exists X: \\ A_1 = attr(L_1, X) \wedge A_2 = attr(L_2, X), \\ L_1, L_2 \in (body(c) \cup head(c)) \\ L_1 \neq L_2, \\ R_1 = rel(L_1) \wedge R_2 = rel(L_2) \end{array} \right) \right) \right)
 \end{array}$$

Diese Anfrage macht einen Gleichverbund über alle Relationen, deren Prädikatsymbole in der zu testenden Klausel c vorkommen. Dies entspricht der logischen Und-Verknüpfung. Das Ergebnis des Gleichverbunds wird auf die Attribute projiziert, die den Variablen des Kopfliterals entsprechen. Die Anzahl der entstehenden Tupel bildet den Zähler des Support-Werts (und auch des Confidence-Werts).

Beispiel 3.8

Gegeben sei eine Datenbank mit den Relationen `parent`, `male` und `has_son` mit folgenden Schema:

```
parent (parent_name, child_name)
male (name)
has_son (name)
```

Die zu testende Klausel sei

```
has_son (X) :- parent (X,Y), male (Y)
```

Der Zähler des Support-Werts berechnet sich dann folgendermaßen:

```
SELECT      COUNT (DISTINCT has_son.name)
FROM        has_son, parent, male
WHERE       has_son.name = parent.parent_name AND
              parent.child_name = male.name;
```

Mehrfachverbunde in relationalen Datenbanken sind sehr aufwendig. Deshalb ist die Evaluation der Anfrage bei mehreren Rumpfliteralen eine teure Operation. Deshalb sollten nur diejenigen Klauseln getestet werden, für die dies unbedingt nötig ist.

Der Nenner des Support-Werts besteht aus

$$\|head(c)\|$$

Diese Extension entspricht der Anzahl der Tupel in der Relation die zum Prädikatsymbol von $head(c)$ gehört. Folgende Anfrage berechnet den Nenner des Support-Werts.

```
SELECT COUNT*
FROM rel(head(c))
```

Diese Anfrage stellt kein Effizienzproblem dar. Sie ist unabhängig von den Rumpfliteralen der Klausel. Das Kopfliteral ist fest, deshalb kann der Wert einmal berechnet werden und ist für die Berechnung aller Support-Werte konstant.

Der Support-Wert verhält sich monoton. Das heißt, er kann beim Hinzufügen eines Rumpfliterals zu einer Klausel nicht steigen. Ist also der Support einer Klausel unter dem spezifizierten Wert σ , braucht diese Klausel nicht mehr zur Generierung längerer Klauseln verwendet werden. Sie kann gleich nach der Berechnung des (zu kleinen) Supports verworfen werden ohne daß der Confidence-Wert berechnet werden muß.

Ist der Support jedoch ausreichend, muß getestet werden, ob die Klausel als Hypothesenklausel akzeptiert werden kann. Dies leistet der Confidence-Wert. Der Zähler des Confidence-Werts ist äquivalent zum Zähler des Support-Werts und braucht nicht neu berechnet zu werden. Die Berechnung des Nenners ist jedoch wesentlich aufwendiger als beim Support. Die Extension

$$\|body(c)\|_{\text{vars}(head(c))}$$

wird durch folgende SQL-Anfrage berechnet:

$$\begin{array}{l}
 \text{SELECT COUNT (DISTINCT } \left\{ \begin{array}{l} \exists i, X: \\ A = attr(body_i(c), X) \wedge \\ X \in \text{vars}(head(c)) \end{array} \right\} \\
 \text{FROM } rel(body_1(c)), \dots, rel(body_n(c)) \\
 \text{WHERE AND } \left\{ \begin{array}{l} \exists i, j, X: \\ A_1 = attr(body_i(c), X) \wedge \\ A_2 = attr(body_j(c), X) \wedge \\ i \neq j \wedge \\ R_1 = rel(body_i(c)) \wedge \\ R_2 = rel(body_j(c)) \end{array} \right\}
 \end{array}$$

Diese Anfrage berechnet den Gleichverbund aller Relationen, deren Prädikate in Rumpfliteralen der Klausel auftauchen. In der WHERE-Klausel stehen Paare aller Attribute, deren zugehörige Variablen in zwei verschiedenen Rumpfliteralen stehen. Die Projektion geht auf alle Attribute, die zu den Variablen eines Rumpfliterals gehören, die auch im Kopfliteral auftauchen.

Beispiel 3.9

Gegeben sei das Datenbankschema und die Testklausel aus Beispiel 3.8. Der Confidence-Wert berechnet sich aus:

```

SELECT    parent.parent_name
FROM      parent, male
WHERE     parent.child_name = male.name
    
```

4

Systemarchitektur

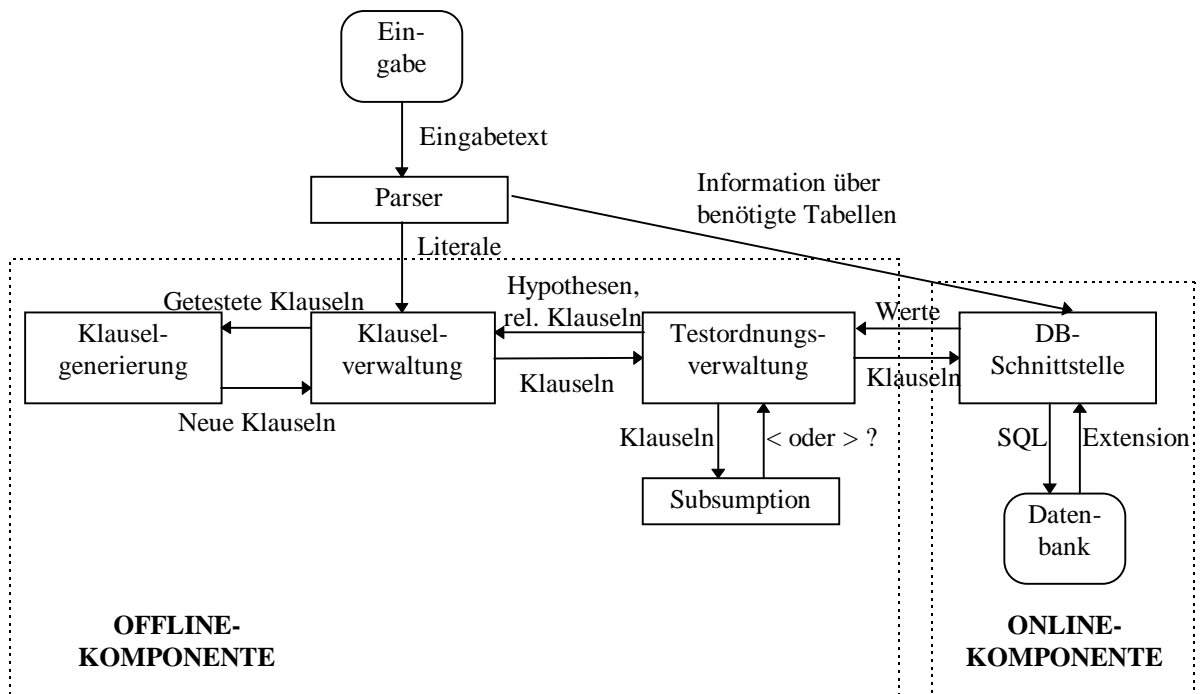
Der in Kapitel 3 beschriebene Algorithmus wurde von Irene Weber, Thomas Klenk und mir implementiert. Dieses Kapitel soll einen Überblick über die modulare Struktur des Systems geben.

Die Implementierungssprachen sind C, Prolog und Dynamic SQL. Verwendete Produkte sind gnu-c, Quintus-Prolog und Ingres-SQL. In den folgenden Abschnitten werden die Entwurfsentscheidungen für die jeweilige Programmiersprache erläutert.

4.1 Übersicht über die Architektur

Das System besteht aus sechs Hauptkomponenten. Im folgenden Schema sind diese Komponenten und der Informationsfluß zwischen ihnen dargestellt.

Abbildung 4.1 Struktur des Systems



Das System besteht im Wesentlichen aus zwei Komponenten, der Online- und der Offline-Komponente.

Die Online-Komponente ist für die Kommunikation mit der Datenbank zuständig. Hier werden für Klauseln die Werte¹ berechnet, die mit den definierten Schwellenwerten verglichen werden. Dazu werden SQL-Anfragen erzeugt, die die Extension von logischen Ausdrücken als Ergebnis haben. Diese werden zum DBMS² geschickt.

In der Offline-Komponente dreht sich alles um die Manipulation von Klauseln. Hier werden zu Beginn aus den in der Eingabedatei spezifizierten Rumpfliterale einstellige Klauseln erzeugt. In der Offline-Komponente werden auch mit Hilfe der Subsumption Testordnungen erzeugt. Aus diesen Testordnungen werden Klauseln, die getestet werden müssen an die Online-Komponente übergeben. Diejenigen, die das Akzeptanzkriterium erfüllen werden in die Hypothese aufgenommen. Aus denen, die noch nicht endgültig verworfen wurden werden neue Klauseln generiert.

4.2 Interne Klauseldarstellung

Im Zentrum der Offline-Komponente steht die Klauselverwaltung. Hier werden Hornklauseln gespeichert. Diese bestehen aus Literalen, die wiederum aus Prädikaten, Variablen und Konstanten bestehen. Variablen und Konstanten bilden die Argumente der Prädikate.

Die Klauselverwaltung besteht im wesentlichen aus vier Symboltabellen:

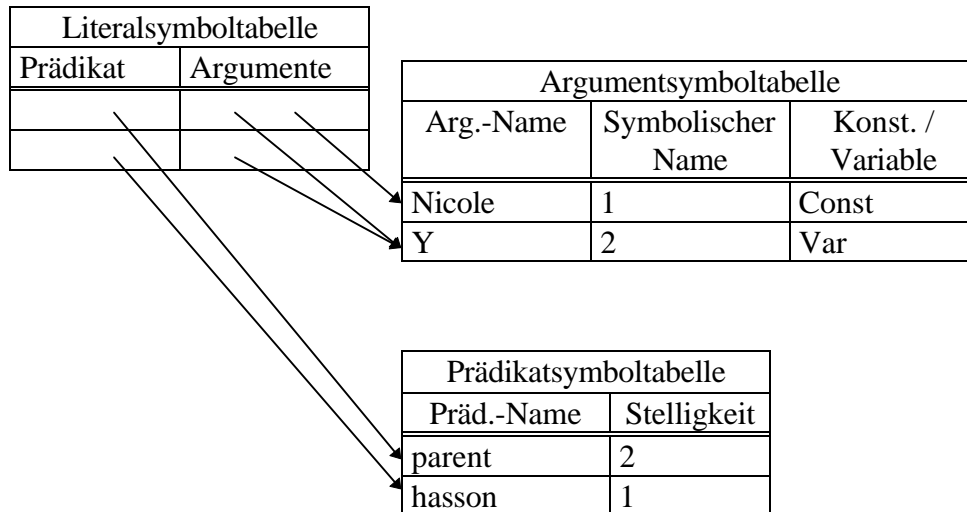
- Prädikatsymboltabelle
- Argumentsymboltabelle
- Literalsymboltabelle
- Rumpfsymboltabelle

Die Prädikatsymboltabelle enthält alle in der Eingabedatei spezifizierten Prädikate und deren Argumentanzahl. Die Argumentsymboltabelle enthält alle in der Eingabedatei vorkommenden Variablen und Konstanten. Dort werden auch symbolische Bezeichner für die Argumente und die Information, ob es sich um eine Variable oder eine Konstante handelt, gespeichert. Ein Eintrag in der Literalsymboltabelle besteht aus einem Verweis auf das zugehörige Prädikat in der Prädikatsymboltabelle und aus Verweisen auf alle Attribute des Literals. Ein Eintrag in der Rumpfsymboltabelle besteht hauptsächlich aus Verweisen in die Literalsymboltabelle. Diese Verweise zeigen auf alle Rumpfliterale des Rumpfes. Zusätzlich wird in der Rumpfsymboltabelle die Tiefe der zum Rumpf gehörigen Klausel gespeichert. Da der Kopf für jede Klausel gleich ist, wird er als Systemvariable einmal gespeichert. Deshalb kann die Rumpfsymboltabelle auch als Klauselsymboltabelle aufgefaßt werden. Im folgenden wird oft von Klauseln gesprochen, die in der Rumpfsymboltabelle gespeichert sind. Dies sollte nicht zur Verwirrung führen.

Den Aufbau der Symboltabellen und die Beziehungen zwischen ihnen zeigt die folgende Abbildung.

¹ hier Support und Confidence

² Database Management System

Abbildung 4.2 Darstellung des Rumpfes $\text{parent}(Y, \text{Nicole}) \wedge \text{hasson}(Y)$ 

Die Rumpfsymboltabelle wurde aus Gründen der Übersichtlichkeit weggelassen.

4.3 Subsumption

Zur Erstellung von Testordnungen muß für jedes Klauselpaar die Subsumptionsbeziehung getestet werden. Es gibt einen Algorithmus, der für Hornklauseln mit wenigen Literalen relativ effizient arbeitet. Die Implementierung dieses Algorithmus auf den vorhandenen Symboltabellen wäre aufwendig. Deshalb wurde auf eine Prolog-Version des Algorithmus zurückgegriffen. Folglich konvertiert die Subsumptionskomponente die interne Klauseldarstellung in das Prolog-Format und ruft dann das Prolog-Prädikat auf.

4.4 Darstellung von Testordnungen

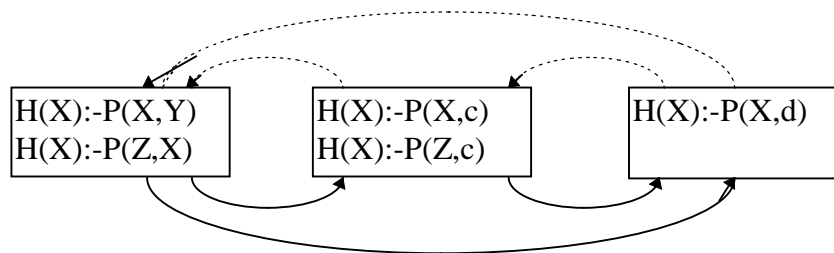
Mit Hilfe der Subsumption wird eine Testordnung erzeugt (s. Abschnitt 3.2.1). Dabei bildet die Menge aller Klauseln auf natürliche Weise eine Partition. Die Klassen dieser Partition enthalten jeweils Klauseln, die sich paarweise gegenseitig subsumieren. Im folgenden werden die Klauseln innerhalb einer Klasse auch äquivalent gemäß der Subsumption genannt. Jede Klasse hat eine (eventuell leere) Menge von „kleineren“ Klassen, die von den Elementen der Klasse subsumiert werden. Außerdem gibt es eine (eventuell leere) Menge von generelleren Klassen, die die Klasse subsumieren. Alle diese Beziehungen zwischen Klauselklassen werden in der Testordnung explizit gespeichert. Eine Testordnung hat folgendes Aussehen.

Abbildung 4.3 Beispiel für eine Testordnung

Die Testordnung

$$\begin{array}{c} H(X):-P(X,Y) \\ H(X):-P(Z,X) \\ \downarrow \\ H(X):-P(X,c) \\ H(X):-P(Z,c) \\ \downarrow \\ H(X):-P(X,d) \end{array}$$

wird folgendermaßen repräsentiert:



Durchgezogene Pfeile bedeuten „ist größer gemäß der Subsumption“, gestrichelte Pfeile bedeuten „ist kleiner gemäß der Subsumption“. Klauseln in einem Rechteck sind gleich gemäß der Subsumption und stellen somit eine Klasse dar.

In dieser Darstellung sind alle transitiven Abhängigkeiten explizit dargestellt. Der Vorteil dieser redundanten Darstellung liegt darin, daß man zum einen von einer Klasse schnell weiß, ob sie in der Testordnung ganz oben steht, das heißt keine größeren Klassen hat. Zum anderen können schnell alle Nachfolger, das heißt kleineren Klassen ermittelt werden.

Die Testreihenfolge ist folgende: Zuerst müssen alle Klauseln getestet werden, die in der Ordnung ganz oben stehen. Dazu wird die Partition durchlaufen, bis man eine oberste Klasse entdeckt. Ein Element dieser Klasse wird auf jeden Fall gegen die Datenbank getestet. Der Test kann drei verschiedene Ergebnisse haben:

- Die Klausel ist eine Hypothesenklausel.
- Die Klausel wird verworfen.
- Die Klausel wird zur Kandidatengenerierung noch benötigt.

In allen Fällen wird die Klasse selbst aus der Testordnung gelöscht.

Falls die getestete Klausel als **Hypothesenklausel** erkannt wurde, wird irgendeine Klausel der Klasse in die Hypothese aufgenommen. Andere Klauseln der Klasse wären in der Hypothese redundant und werden deshalb nicht aufgenommen. Die Klauseln in der Klasse werden zur Generierung neuer Klauseln nicht mehr benötigt (die daraus generierten wären spezieller und damit in der Hypothese redundant), deshalb werden sie in der Rumpfsymboltabelle gelöscht. Alle in der Testordnung kleineren Klassen sind spezieller als die neue Hypothesenklausel und werden deshalb gelöscht (s. 3.2.2). Auch alle Klauseln in diesen Klassen werden aus der Symboltabelle entfernt.

Falls die getestete Klausel **verworfen** werden soll, wird die Klasse aus der Testordnung gelöscht. Zusätzlich werden wie beim obigen Fall alle kleineren Klassen aus der Testordnung und deren Klauseln aus der Symboltabelle gelöscht.

Falls die Klausel weder eine Hypothesenklausel ist, noch verworfen werden soll, wird sie **noch zur Generierung neuer Klauseln benötigt**. Die Klauseln in der Klasse werden nicht aus der Symboltabelle gelöscht. Die Klasse wird jedoch in diesem Testschritt nicht mehr benötigt und wird deshalb aus der Testordnung entfernt. Dadurch entstehen neue oberste Klassen in der Testordnung, die im nächsten Schritt zum Test anstehen.

Zusammenfassend kann man feststellen, daß immer nur oberste Klassen getestet werden. Nach dem Test werden diese obersten Klassen und eventuell kleinere Klassen entfernt. Also sind alle Klassen entweder irgendwann einmal eine oberste Klasse oder sie werden im Zuge der Entfernung einer höheren Klasse gelöscht. Daraus folgt, daß der Testschritt damit beendet ist, daß die Testordnung keine Klassen mehr enthält.

4.5 Klauselgenerierung

Nach Abarbeiten der Testordnung bleiben nur noch Klauseln übrig, die zur Generierung neuer Klauseln benötigt werden. Entscheidend ist, daß in verbundenen Klauseln mit n Rumpfliteralen nur Literale vorkommen können, deren Tiefe höchstens n ist. Das bedeutet, daß im n -ten Schritt des Algorithmus nur Klauseln der maximalen Tiefe n generiert werden müssen. Die Klauselgenerierung ist deshalb zweigeteilt in *offline candidate determination* und *next depth literal*.

Der Ablauf ist der folgende: Zuerst werden aus den Klauseln in der Rumpfsymboltabelle mit *ocd* solche Klauseln generiert, die eine Tiefe kleiner als n haben. Dann fehlen nur noch die Klauseln der Tiefe n . Diese werden dadurch erzeugt, daß an jede Klausel in der Rumpfsymboltabelle jedes Literal aus der Literalsymboltabelle angehängt wird. Dann wird getestet, ob die Klauseln verbunden sind. Alle unverbundenen werden verworfen.

Ergebnis der Klauselgenerierung ist eine neue Rumpfsymboltabelle. Diese enthält alle Testkandidaten für den nächsten Schritt. Die alte Rumpfsymboltabelle wird nie wieder gebraucht und kann gelöscht werden.

4.6 Datenbank-Schnittstelle

Beim Entwurf des Algorithmus fiel die Wahl auf die Verwendung eines relationalen Datenbanksystems (Ingres). Vor allem der große Sprachumfang von SQL, der auch die Definition intensionaler Prädikate, wie „>“ erlaubt, die effiziente Pufferverwaltung und die

Anfrageoptimierung, die logische Ausdrücke in where-Klauseln reduziert, waren Argumente für die relationale Datenbank.

Der Algorithmus erlaubt, daß die Datenbank auf einem anderen Rechner installiert ist, als der Data-Mining-Algorithmus. Der Rechner, auf dem der Algorithmus läuft wird im folgenden Client-Seite, der entfernte Rechner, auf dem die Datenbank läuft, Server-Seite genannt. Auf der Server-Seite muß ein Server installiert sein, der über Dynamic-SQL Anfragen an die Datenbank schicken kann. Das Ergebnis dieser Anfragen ist in der Regel eine Zahl (select count...). Die Verbindung von Client und Server geschieht durch Remote Procedure Calls (RPC) im TCP/IP - Protokoll. Die Schnittstelle zwischen Client und Server wurde mit Hilfe des Werkzeugs *rpcgen* erzeugt. Die Evaluation eines SQL-Statement erfolgt in den Schritten:

1. Erzeugung des SQL-Statements auf der Client-Seite
2. Aufruf der (entfernten) Prozedur (eigentlich nur ein Stub) zur Auswertung der Anfrage (für den Client ist nicht sichtbar, daß die Prozedur auf einem anderen Rechner läuft)
3. Transport der Anfrage über das Netz
4. Empfang der Anfrage auf der Server-Seite
5. Aufruf der Prozedur zur Auswertung der Anfrage (Zurück kommt ein Cursor)
6. Auslesen des Cursor und Rücktransport des Ergebnisses

4.7 Benutzung des Algorithmus / Eingabedatei

Der Algorithmus erhält als Eingabe:

1. Eine relationale Datenbank D .
2. Eine Zuordnung $rel: P \rightarrow R$, die jedem Prädikatsymbol eine erzeugbare Relation in Form eines SQL-Statements zuordnet. Diese SQL-Statements (in der Regel CREATE VIEW...)
Die Definition neuer Tabellen ist komfortabel, wenn gegebene Relationen nicht genau die Information enthalten, nach der gesucht wird. Außerdem können hier verschiedene Relationen zu einer verbunden werden (denormalisierung).
3. Ein positives funktionsfreies Literal h , das den Kopf der zu entdeckenden Klauseln darstellt.
4. Eine Menge B aus positiven funktionsfreien Literalen, die im Rumpf einer zu entdeckenden Klausel auftauchen können. Das Prädikatsymbol jedes Literals muß wie in 2. beschrieben existieren.
5. Ein Support-Wert σ .
6. Ein Confidence-Wert γ .

Die Eingabedatei hat folgendes Aussehen:

Measure_Declaration:

support= 0.75
confidence= 0.75

Predicate_Declaration:

male (name)=~create view vmale as select * from male_original~
has_son (name)=~create view vhas_son as select * from hasson_original~
parent (elter,kind)= ~create view vparent as select * from parent_original~

Head_Declaration:

has_son (X)

Body_Declaration:

male (Y)
male (Z)
parent (X,Y)
parent (Z,X)
parent (X,~alf~)

End

5

Anwendung des Verfahrens auf einer relationalen Datenbank

Der Algorithmus wurde auf eine von M. Bauer und S. Rapp erstellten Datenbank angewendet. Dabei stellt sich die Frage, wie aus den existierenden Tabellen Prädikate erzeugt werden können, die leicht verständlich und interpretierbar sind. Notwendig sind Prädikate, die Eigenschaften von Attributen ausdrücken. Zusätzlich werden Prädikate benötigt, die Beziehungen (Relationships) zwischen Tabellen ausdrücken können. Abschnitt 5.2 beschreibt eine Möglichkeit zur Erzeugung von Prädikaten aus Datenbankrelationen. In Abschnitt 5.1 wird ein erster Probelauf auf einer phonetischen Datenbank wiedergegeben. Abschnitt 5.3 zeigt Verbesserungsmöglichkeiten auf, die sich aus dem Versuch ergaben.

5.1 Beispiel: Phonetische Datenbank

Der bisher beschriebene Algorithmus ist Grundlage für viele Erweiterungs- und Verbesserungsmöglichkeiten. Die Anwendung des Algorithmus auf eine einfache Datenbank mit kleiner Eingabe soll diese Probleme ans Tageslicht bringen. Im folgenden ist beschrieben, wie der Algorithmus auf eine Datenbank angewendet wurde, die phonetische Daten aus Nachrichtensendungen enthält. Zuerst wird der genaue Ablauf des Experiments beschrieben, danach werden die einzelnen Verbesserungsmöglichkeiten ausführlich herausgehoben.

5.1.1 Format der phonetischen Datenbank

Die Datenbank enthält Daten zu mehreren Radio-Nachrichtensendungen. Sie besteht aus drei Relationen, die Daten zu Wörtern, Silben und Phonemen enthalten. Zu jedem Wort sind Attribute, wie Wortart, vorhergehendes und nächstes Wort und seine Silben gespeichert. Zu jeder Silbe ist unter anderem gespeichert, ob sie betont ist. Phoneme enthalten Information über Lautstärke und Frequenzspektrum. Das Relationenschema ist in folgender Liste wiedergegeben:

Relation Wordtable:

Word	Codierung des Wortes, z.B. flw1 (file 1 word 1) als erstes Wort der ersten Datei
Filenumber	Nummer der Datei, die Dateien entsprechen einzelnen Nachrichten

	und sind chronologisch durchnummeriert
File	Name der Nachrichtendateien, aus denen die Informationen kommen.
Previous	Wort, das dem aktuellen Wort vorangeht.
Next	folgendes Wort
Inlexikon	Ob das Wort in dem vom ursprünglichen Bearbeitungsprogramm benutzten Lexikon enthalten war, wenn ja, gibt es nämlich eine Information, auf welcher Silbe die Wortbetonung laut Lexikoneintrag liegt, wenn nicht, fehlt diese Information.
Tag	Information über die Wortart, z.B. Nomen, finites Verb Adjektiv.

Relation Syltable:

Syllable	Codierung der Silbe f1w1s3 (file 1 word 1 syllable 3)
Filenumber	s.o.
File	s.o.
Accent	Betonung
Word	Codierung des Wortes, in dem die Silbe enthalten ist
Previous	vorherige Silbe
Next	folgende Silbe
Syls2wordbeg	Anzahl der Silben zum Wortanfang inklusive der Silbe selbst
Syls2wordend	Anzahl der Silben zum Wortende inklusive der Silbe selbst
Disttonextp	Abstand zur nächsten Pause in Zeiteinheiten (Sekunden?)
Disttoprevp	Abstand zur vorherigen Pause
Lenofnextp	Länge der nächsten Pause
Lenofprevp	Länge der vorhergehenden Pause
Lenexpected	Erwartete Länge der Silbe
Lenmeasure	Tatsächlich gemessene Länge der Silbe
Lenrelative	Verhältnis der erwarteten zur tatsächlichen Länge
Wordstress	=yes, wenn die Wortbetonung auf der Silbe liegt (nur von Wörtern, die im Lexikon stehen, bekannt.)
rms*	Lautstärkewerte aufgeteilt in bestimmte Frequenzbereiche

Relation Phontable:

Phonem_id	Codierung des Phonems s.o.
Filenumber	s.o.
File	s.o.
Insyl	Codierung der Silbe, der das Phonem zugeordnet wird
Previous	s.o.
Next	s.o.
onc	onset = Phonem steht in der Silbe vor dem ersten Vokal nucleus = erster Vokal in der Silbe coda = Phonem steht hinter dem ersten Vokal

phoneme	Codierung aller möglichen phone, z.B. nc_m nasal,consonant
vowcon	Konsonant oder Vokal ?
type	Nasal, Plosiv, Liquid etc. bei Konsonanten lang, kurz, schwa-, Diphthong bei Vokalen

5.1.2 Erzeugte Prädikate

Der erste Versuch, der sich nur auf Wortarten fixierte und nur auf die Relation words zugreift, brachte eine Vielzahl von Verbesserungsmöglichkeiten zutage. Deshalb wird hier diese einfache und übersichtliche Eingabe wiedergegeben. Ziel dieses einfachen Beispiels ist nicht, die Leistungsfähigkeit des Algorithmus bezüglich Laufzeit oder Ergebnisqualität zu beurteilen, sondern soll ausschließlich zum besseren Verständnis der Probleme bei der Anwendung des Algorithmus dienen.

Folgende Prädikate wurden aus der Datenbank erzeugt:

```
words_tag (word,tag) =
    ~create view words_tag as select word,tag from words~
```

```
words_next_word (word, next_word) =
    ~create view words_next_word (word, next_word) as
    select word, next from words~
```

```
words_prev_word (word, prev_word) =
    ~create view words_prev_word (word, prev_word) as
    select word, previous from words~
```

5.1.3 Rumpfliteralmenge

Die hier benutzte Rumpfliteralmenge ist sehr knapp gehalten. Dabei wurde viel Wert auf gute Verständlichkeit gelegt. Die Ergebnisse waren eigentlich schon im Vorfeld zu erwarten. Gesucht wird nach Regelmäßigkeiten in Wortfolgen mit besonderen Wortarten. Die Fragestellung war: Hängt die Tatsache, daß ein Wort eine Nomen ist von den Wortarten vor oder nach dem Wort ab.

Head_Declaration:

```
words_tag (word, ~nn~)
```

Body_Declaration:

```
words_next_word(word, word)
words_next_word(word, n_word)
words_next_word(n_word, nn_word)
words_prev_word(p_word, word)
words_prev_word(p_word, p_word)
```

```

words_tag(word, ~nn~)           # Tautologie
words_tag(word, ~art~)          # Widerspruch
words_tag(n_word, ~nn~)
words_tag(n_word, ~art~)
words_tag(p_word, ~nn~)
words_tag(p_word, ~art~)
words_tag(pp_word, ~nn~)
words_tag(pp_word, ~art~)
words_tag(nn_word, ~nn~)
words_tag(nn_word, ~art~)

```

5.1.4 Gefundene Klauseln

Da die zu erwartenden Formeln nur für einen geringen Teil der Datenbank zutreffen empfiehlt es sich, die Support- und Confidence-Werte sehr niedrig anzusetzen. Deshalb wurden folgende Einstellungen gewählt:

```

Measure_Declaration:
  support= 0.2
  confidence= 0.3

```

Gefunden wurden folgende Klauseln:

Hypothese:

=====

```

words_tag (word, ~nn~)← words_prev_word(word, p_word), words_tag(p_word, art)
words_tag (word, ~nn~)← words_next_word(word, n_word), words_tag(n_word, appr)
words_tag (word, ~nn~)← words_tag(word, nn)

```

Das bedeutet, daß **vor** 20% (Support 0,2) der Nomen ein Artikel steht. Wenn ein Wort ein Artikel ist, ist mit 30 prozentiger Sicherheit das nächste Wort ein Nomen. **Nach** 20% der Nomen steht eine Adposition. Mit 30 prozentiger Sicherheit kommt vor einer Adposition ein Nomen.

Die dritte Klausel sagt lediglich aus, daß Ein Nomen ein Nomen ist. Diese Formel sollte zur Kontrolle dienen. Sie muß auf jeden Fall gefunden werden (natürlich auch bei höheren Support- und Confidence-Werten)..

5.2 Abbildung von Relationen auf Prädikate

Im oben gezeigten Beispiel waren große Relationen gegeben. Diese können schlecht direkt als Prädikate verwendet werden. Deshalb stellt sich die Frage, wie aus bestehenden Datenbankrelationen Prädikate konstruiert werden können. Eine Möglichkeit, die sich an [4] anlehnt beschreibt dieser Abschnitt.

Inhalte von Datenbanken beschreiben Objekte (Entities) und Beziehungen zwischen diesen Objekten (Relationships). Diese werden beim Entwurf von Datenbanken auf Relationen mit Schlüssel-Fremdschlüssel-Beziehungen abgebildet. Diese Relationen sind jedoch oft nicht leicht zu interpretieren, wenn sie in Klauseln verwendet werden. Zum Beispiel die Relation

Kunde (KN_r, Name, Vorname, Telefon, ImBundestag)

drückt eigentlich schon eine Konjunktion von Tatsachen aus. Das Tupel

(1, Kohl, Helmut, 25467, true)

besteht aus folgenden Aussagen, die konjunktiv miteinander verknüpft sind

- Der Kunde mit der Nummer 1 heißt Kohl mit Nachnamen **und**
- Der Kunde mit der Nummer 1 heißt Helmut mit Vornamen **und**
- Der Kunde mit der Nummer 1 hat die Telefonnummer 25467 **und**
- Der Kunde mit der Nummer 1 sitzt im Bundestag

Für die Analyse einer Datenbank sind aber in der Regel Informationen über einzelne Attribute interessant. Das bedeutet, daß es sinnvoll ist, eine Relation in viele Prädikate umzuwandeln. Dafür existieren verschiedene Möglichkeiten (in Anlehnung an [4]).

1. Eine Relation $R(s_1, \dots, s_n, a_1, \dots, a_m)$ mit Primärschlüssel s_1, \dots, s_n wird in m zweistellige Prädikate der Form $a_i(s_1, \dots, s_n, a_i)$ umgewandelt. Der Name des Prädikats entspricht dem Namen des Attributs, das es beschreibt. Zweistellig bedeutet hier nicht, daß die Extension des Prädikats nur zwei Spalten hat. Vielmehr hat das Prädikat außer den Primärschlüsselattributen nur noch eine Stelle.

Ein sinnvolle Prädikate für obiges Beispiel wären:

- Name(KN_r, Name)
- Vorname(KN_r, Vorname)
- Telefon(KN_r, Tel)
- ImBundestag(KN_r, ImBundestag)

2. Bei Attributen über dem Wertebereich {true, false} kann auf die zweite Stelle verzichtet werden, falls die Negation uninteressant ist.

- ImBundestag(KN_r)

Auf diese Weise können Prädikate erzeugt werden, die Informationen über einzelne Attribute der Datenbank enthalten. Die Hornklausellogeik bietet aber auch die Möglichkeit, Beziehungen zwischen Relationen darzustellen. In Datenbanken existieren grundsätzlich

zwei Arten von Schlüssel-Fremdschlüssel-Beziehungen, 1:n-Beziehungen und n:m-Beziehungen. Eine 1:n Beziehung wird durch Fremdschlüsselattribute realisiert:

Kunde (KNr, Name, Vorname, Telefon, ImBundestag)
Bestellung (BNr, BestellerKNr, Artikel)

Wie kann jetzt ausgedrückt werden, daß ein Bundestagsabgeordneter K einen bestimmten Artikel X bestellt hat?

$\text{ImBundestag}(K) \wedge \text{BestellerKNr}(B, K) \wedge \text{Artikel}(B, X)$

Man erkennt, daß alle benötigten Prädikate sowieso durch das oben beschriebene Verfahren existieren. Zur Darstellung von 1:n-Beziehungen sind also keine weiteren Prädikate erforderlich. Falls der Benutzer jedoch an schnelleren Laufzeiten interessiert ist, kann er Prädikate, wie

$\text{hat_bestellt}(\text{KNr}, \text{Artikel})$

mit Hilfe eines relationalen Verbundes (Join) leicht selbst definieren. Dies geht jedoch auf Kosten der Flexibilität, weil dadurch der Hypothesenraum verkleinert wird.

Zusätzliche Prädikate werden zur Analyse von n:m-Beziehungen benötigt. Eine n:m-Beziehung wird in relationalen Datenbanken durch eine eigene Relation realisiert.

Kunde (KNr, Name, Vorname, Telefon, ImBundestag)
kann_leiden (KNr1, KNr2, seitJahr)

Will man darstellen, daß alle Kunden, die im Bundestag sitzen, sich gut leiden können benötigt man ein Prädikat, daß dies ausdrückt, nämlich

$\text{kann_leiden}(\text{KNr1}, \text{KNr2})$

Es ist also sinnvoll, Prädikate zu formulieren, die nur aus den Schlüsselattributen einer Relation bestehen. Zusätzlich kann dann auch ausgesagt werden, daß eine Person überhaupt Kunde ist ($\text{Kunde}(\text{KNr})$).

Die in diesem Abschnitt dargestellten Verfahren zur Konstruktion einer sinnvollen Eingabe sind keinesfalls zwingend, stellen jedoch in der Regel einen großen Teil der benötigten Prädikate dar. Der Vorteil dieses Verfahrens ist, daß es vollständig automatisiert werden kann. Ich stelle mir dabei einen Generator vor, der aus der Menge aller Datenbankrelationen mit Hilfe der Systemkataloge einer Datenbank alle Prädikate (oder erzeugende create view - Statements) erzeugt, die obiger Beschreibung entsprechen. Die Ergebnisprädikatmenge kann dann vom Benutzer um nicht relevante Prädikate verkleinert und um zusätzliche Prädikate erweitert werden. Dieser Schritt zeigt, daß ein

sinnvoller KDD-Prozeß interaktiv ist und daß Hintergrundwissen seitens des Benutzers eingebracht werden muß.

5.3 Schwachpunkte des Algorithmus und Verbesserungsmöglichkeiten

Das beschriebene Experiment zeigte einige Verbesserungsmöglichkeiten auf. Dies sind zum einen Änderungen im Eingabeformat, die dem Benutzer den Umgang mit dem Algorithmus erleichtern sollen. Zum anderen sind dies Änderungen im Ablauf des Algorithmus, die den Hypothesenraum einschränken oder Mächtigkeit des Algorithmus erhöhen. Diese Verbesserungen sind im folgenden aufgezählt:

1. Nichttesten von Klauseln, die neue Variablen enthalten

Der Versuch zeigt, daß viele Klauseln getestet werden, die sehr großen Support, aber kleine Confidence haben. Diese Klauseln benötigen eine SQL-Anfrage für den Support und zwei SQL-Anfragen für die Confidence¹ und sind damit sehr laufzeitintensiv. Zusätzlich werden aus diesen Klauseln oft neue Kandidaten generiert, die das gleiche Verhalten zeigen. Diese Klauseln haben alle eine Eigenschaft gemeinsam: sie enthalten Variablen, deren entsprechende Attribute in der select-Anweisung für die Extension von (Body \wedge Head) nicht in der where-Klausel auftauchen. Dies sind Variablen, die nur ein einziges Mal in der ganzen Klausel vorkommen. Zum Beispiel die Klausel

$$\text{tag}(\text{WORD}, \text{'nn'}) \leftarrow \text{next_word}(\text{WORD}, \text{NWORD})$$

enthält die Variable NWORD nur einmal. Die Aussage dieser Formel ist, daß es zu einem Wort mit Wortart Nomen ein nächstes Wort gibt. Dies ist natürlich fast immer der Fall. Deshalb ist der Support sehr hoch. Andererseits ist der Anteil der Wörter, die Nomen sind nur ein kleiner Teil aller Wörter, die ein Folgewort haben. Dies führt zu einer kleinen Confidence.

Klauseln mit Variablen, die nur einmal vorkommen drücken immer eine Existenzaussage aus. Solche Aussagen sind oft nicht erwünscht. Im Versuch fielen ca. zwei drittel aller Tests auf solche Existenzaussagen. Deshalb sollte der Benutzer die Möglichkeit haben, den Test solcher Klauseln zu verhindern.

Sollen diese Formeln nicht getestet werden, ist die linked- und range restricted-Eigenschaft² zu schwach. Man benötigt zusätzlich eine Eigenschaft, die hier Abgeschlossenheit genannt wird.

Definition 5.1

Eine Klausel ist abgeschlossen, falls sie keine Variable nur einfach enthält.

¹ Der Nenner des Support ist konstant und benötigt deshalb keine eigene Anfrage. Bei der Confidence müssen sowohl Zähler als auch Nenner bei jeder Klausel neu berechnet werden.

² deutsch: verbunden, bzw. bereichsbeschränkt

Es sollen also nur Klauseln getestet werden, die verbunden, bereichsbeschränkt und abgeschlossen sind.

2. Automatische Bestimmung von String-Konstanten

Im Versuch wurden viele Literale definiert, die sich nur in einer Konstanten unterscheiden.

```
tag(WORD, 'nn')
tag(WORD, 'vvfin')
tag(WORD, 'art')
...
```

Hier ist die Intuition des Benutzers, daß verschiedene Wortarten in Frage kommen. Diejenigen die zu Formeln mit ausreichendem Support und Confidence führen sollen vom Algorithmus bestimmt werden. Die Bestimmung der Eingabe fordert jedoch vom Benutzer ein hohes Maß an Wissen über den Wertebereich des Attributs, für das Konstanten angegeben werden.

Wesentlich komfortabler wäre die Eingabe einer „generischen Konstanten“, die vom System automatisch ermittelt wird. Zum Beispiel sollte der Benutzer die Möglichkeit haben, folgendes Literal zu definieren:

```
tag(WORD, generic_string_constant)
```

Dieses Literal faßt alle Literale zusammen, die aus dem Wertebereich des zweiten Attributs gebildet werden können.

Für die Realisierung einer automatischen Konstantenbestimmung gibt es zwei Möglichkeiten.

- **Generierung von Literalen mit Hilfe des Wertebereichs des Attributs.**
Hier ermittelt ein Precompiler alle Werte, die für das Attribut in Frage kommen. Der Vorteil dieses Verfahrens ist, daß der Algorithmus selbst nicht geändert werden muß. Die Implementierung des Precompilers wäre einfach. Der Nachteil liegt in der großen Anzahl von Literalen, die entstehen können. Es werden immer so viele Literale erzeugt, wie die Wertemenge Elemente hat. Beim Test von Klauseln mit solchen Literalen stehen diese in der Testordnung alle auf der selben Ebene, das heißt, sie müssen alle auf jeden Fall getestet werden. Dies bedeutet, daß der Algorithmus bei Attributen mit großen Wertebereichen kaum noch anwendbar sein wird.
- **Integration einer Komponente zur automatischen Konstantenbestimmung in den Algorithmus**
Dieser Ansatz hat den Nachteil, daß er wesentlich aufwendiger zu implementieren ist als der oben beschriebene. Die Vorteile rechtfertigen jedoch

den Aufwand. Es gibt die Möglichkeit, für eine Klausel, in der eine Konstante ermittelt werden soll, eine Tabelle oder View zu erzeugen, die alle Werte der Konstanten enthält. Aus dieser Tabelle kann dann mit einfachen und schnellen Selektionen Support und Confidence bestimmt werden, ohne daß ein aufwendiger Join durchgeführt werden muß.

Das Problem beider Ansätze liegt in der Bestimmung von mehreren Konstanten innerhalb einer Klausel. Prinzipiell müssen alle Kombinationen aller möglichen Werte durchprobiert werden. Der Aufwand entspricht also dem Produkt der Mächtigkeiten aller beteiligten Wertemengen.

Beispiel 5.1

In der Formel

$$\begin{aligned} \text{tag}(\text{WORD}, \text{'nn'}) \leftarrow & \\ & \text{next_word}(\text{WORD}, \text{NWORD}), \quad \text{tag}(\text{NWORD}, \\ & \text{generic_string_const}), \\ & \text{prev_word}(\text{WORD}, \text{PWORD}), \quad \text{tag}(\text{PWORD}, \\ & \text{generic_string_const}) \end{aligned}$$

müssen für beide generischen Konstanten jeweils alle Attributwerte des Attributs eingesetzt werden. Hat das Attribut n verschiedene Attributwerte, kommen n^2 Kombinationen in Frage.

3. Definition von Verbindungen zwischen Argumenten von Literalen statt Variablen

In der vorliegenden Version des Algorithmus müssen viele Literale mit gleichen Prädikatsymbolen definiert werden:

```
next_word(word, n_word)
next_word(n_word, nn_word)
tag(n_word, ~nn~)
tag(nn_word, ~nn~)
```

Diese Literale unterscheiden sich nur in den Bezeichnern für Variablen. Eigentlich spielen diese Bezeichner keine Rolle. Es sollen mit Hilfe der Variablen nur Verbindungen zwischen Literalen definiert werden. Eine einfachere und weniger redundante Art der Definition des Hypothesenraums wäre wünschenswert.

DLAB [9] ist ein Verfahren zur Definition eines eingeschränkten Hypothesenraums, das mit Hilfe eines Algorithmus aus einer DLAB-Definition Klauseln generiert. Obwohl dieses Verfahren für das hier beschriebene System nicht direkt anwendbar ist, wäre ein ähnliches Vorgehen denkbar.

4. Möglichkeit zur Nutzung eingebauter Prädikate zum Zahlenvergleich

Eine starke Einschränkung für das System ist, daß es über keinerlei Zahlenbehandlung verfügt. Oft stecken jedoch gerade in numerischen Attributwerten nützliche Informationen. Im vorliegenden Algorithmus können Zahlen nur miteinander verglichen werden, wenn das Vergleichsprädikat extensional berechnet wird. Die entstehende Tabelle kann (besonders bei reellen Wertebereichen) sehr groß werden. Außerdem müßte für jedes Paar von numerischen Attributen, die verglichen werden sollen, eine gesonderte Vergleichsrelation definiert werden. Enormer Speicherbedarf und hoher Berechnungsaufwand wären die Folge.

Zur Lösung dieses Problems bietet sich die Verwendung von Hintergrundwissen über den Vergleich von Zahlen an. Vergleichsprädikate, wie „>“ oder „<“ können vom System zur Verfügung gestellt werden. Dies kann durch Ausnutzung des SQL-Sprachumfangs geschehen. Dort werden in where-Klauseln Vergleichsoperatoren zur Verfügung gestellt.

Die Klausel

$$H(X) \leftarrow P(X, Y) \wedge X < Y$$

wird zur Berechnung des Zählers der Meßwerte in die SQL-Anweisung

```
SELECT COUNT ( DISTINCT X)
FROM H, P
WHERE H.X = P.X AND H.X < P.Y
```

Dies hat den zusätzlichen Vorteil, daß nur ein zweifacher statt einem dreifachen Join notwendig ist¹ [LS] .

Folgende Probleme ergeben sich dadurch, daß die Prädikate nicht extensional definiert sind:

- Ein Vergleichsprädikat kann nicht als Kopf der Klausel verwendet werden, denn die Extension ist nicht berechenbar. Diese wird aber für den Nenner des Support benötigt.
- Ein Vergleichsprädikat kann nicht als alleiniges Rumpfpredikat stehen, der Zähler der Confidence ist nicht berechenbar
- Die linked-Eigenschaft zum Testen von Klauseln ist zu schwach. Es dürfen nur Klauseln getestet werden, wenn alle Variablen in Vergleichsprädikaten auch in einem anderen Prädikat vorkommen.
- Wird ein Vergleichsliteral mit Konstanten definiert ergibt sich das Problem, daß aus der Subsumptionsbeziehung nicht mehr die Folgerungsbeziehung ableitbar ist.

Die Literale

¹ Ein k-fach Join hat eine Zeitkomplexität von n^k , wobei n die mittlere Anzahl der Tupel in den beteiligten Relationen ist. Das Einsparen einer Relation beim Join verringert die Laufzeit also um eine Potenz.

$$>(X,5)$$

$$>(X,4)$$

sind gemäß der Θ -Subsumption unvergleichbar. Trotzdem gilt

$$>(X,5) \not\vdash >(X,4)$$

5. Automatische Bestimmung numerischer Beschränkungen

Will man Schwierigkeiten bei der Subsumption von Vergleichsprädikaten vermeiden kann ein zur Bestimmung von String-Konstanten analoges Vorgehen verwendet werden. Der Benutzer gibt nur an, für welche Variablen Beschränkungen bestimmt werden sollen, falls sie in einer Klausel vorkommen. Beschränkungen sind alle Arten von mathematischen Zusammenhängen. Einfache Beschränkungen sind Vergleiche, wie $X < 5$, oder das Enthaltensein in Intervallen, wie $X \in [5..100]$. Kompliziertere Beschränkungen sind beispielsweise lineare Zusammenhänge zwischen mehreren Variablen ($2x + 3y = 10$).

Eine Komponente zur automatischen Bestimmung von Intervallen wurde im Rahmen dieser Diplomarbeit entwickelt.

Mit dieser Komponente sind Ausgaben der Form

$$h(V) \leftarrow b1(V, X), b2(V, Y), Y \in [10..20], X \in [100..200]$$

möglich

Wie diese Komponente arbeitet wird im nächsten Kapitel beschrieben.

6

Eine Komponente zur automatischen Intervallbestimmung

Thema dieses Kapitels ist die Erweiterung des beschriebenen Verfahrens um eine numerische Komponente.

Das bisherige Verfahren hat, wie im letzten Kapitel beschrieben, große Schwierigkeiten im Bezug auf Anwendbarkeit und Effizienz. Deshalb wurde im Rahmen dieser Diplomarbeit ein Verfahren entwickelt, das in der Lage ist, numerische Attribute durch Angabe von Intervallen so einzuschränken, daß Support und Confidence erreicht werden.

Beispiel 6.1

Die Klausel

$$\text{accent}(S) \leftarrow \text{inword}(S, W), \text{tag}(W, \text{'NN'}), \text{volume}(S, V)$$

sagt aus, daß eine Silbe S dann betont ist, wenn das Wort, in dem sie steht, ein Nomen ('NN') ist und die Silbe eine bestimmte Lautstärke V hat. Die Aussage über die Lautstärke ist dabei sehr schwach. Es handelt sich um eine Existenzaussage, die in diesem Kontext nicht interessant ist. Interessant wäre eine Aussage darüber, in welchem Bereich sich die Lautstärke bei betonten Nomen bewegt.

Aus diesem Grund wäre es wünschenswert, wenn die numerische Komponente eine Ausgabe der Art

$$\text{accent}(S) \leftarrow \text{inword}(S, W), \text{tag}(W, \text{'NN'}), \text{volume}(S, V), V \in [100..200]$$

erzeugen könnte.

Im folgenden wird zuerst eine Übersicht über die existierende Literatur zu numerischen Komponenten von ILP-Systemen gegeben. Danach wird der in dieser Diplomarbeit entwickelte Algorithmus zur Bestimmung von Intervallen in Klauseln vorgestellt, die nur eine Variable besitzen (eindimensionale Intervallbestimmung), für die ein Intervall bestimmt werden soll. Anschließend wird dieser Ansatz auf die Bestimmung beliebig vieler Intervalle innerhalb einer Klausel übertragen (mehrdimensionale

Intervallbestimmung). Das Kapitel schließt mit einer Zusammenfassung und Diskussion des Ansatzes.

6.1 Literatur

In der mir zugänglichen Fachliteratur herrscht großes Schweigen dazu, wie numerische Komponenten von ILP-Systemen arbeiten. Erkennbar ist, daß existierende Verfahren wenig Wert auf Effizienz legen (und deshalb nur bedingt anwendbar sind).

FOIL [5] produzierte in Versuchen von M. Bauer sehr lange, kaum verständliche Klauseln mit vielen Vergleichsprädikaten in kaum akzeptabler Zeit. Wie die numerische Komponente von FOIL arbeitet war der vorhandenen Literatur nicht zu entnehmen.

[AR] definieren in ihrem Foliensatz „Numerical Reasoning in ILP“ numerische Prädikate als Prolog-Klauseln, die dem System als Hintergrundwissen zur Verfügung gestellt werden. Dadurch kann man leicht komplexe Prädikate, wie Regressionszusammenhänge definieren. Die Auswertung dieser Prädikate wird ganz dem Prolog-System überlassen, was wohl zu Laufzeiten führt, die nicht akzeptabel sind. [AR] machen keine Laufzeitanalyse.

Mizoguchi und Ohwada [MH] entwickeln eine numerische Komponente für ihr System GKS. Mit welchen Qualitätsmaßen und mit welchem Verfahren nach den Beschränkungen gesucht wird, ist dem Artikel nicht entnehmbar. Es wird angedeutet, daß ein Greedy-Algorithmus verwendet werden kann.

Auch CLAUDIEN verfügt über eine Komponente, die per Hillclimbing numerische Beschränkungen ermittelt. In meiner Literatur zu CLAUDIEN [6] [10] ist diese Komponente jedoch nicht näher beschrieben.

Der einzige Artikel, der die Zahlenbehandlung tiefergehend beschreibt, ist [SA]. Leider handelt es sich dabei um ein aussagenlogisches Verfahren (kein ILP). Dort werden alle numerischen in einem Vorverarbeitungsschritt diskretisiert (partitioniert) und in boolschen Tabellen gespeichert, d.h. aus der Tabelle

Name	Alter
Stefan	26
Nicole	21
Peter	50

wird die Boolsche Tabelle

Name	20..25	26..30	40..50
Stefan	0	1	0
Nicole	1	0	0
Peter	0	0	1

erzeugt. Aus solchen Tabellen werden dann Klauseln, die Intervalle enthalten ermittelt. Die Bestimmung der Intervalle geschieht durch Kombination von nebeneinanderliegenden Klassen.

Unter Diskretisierung versteht man das Einteilen eines (evtl. reellen) Wertebereichs in Klassen, d.h. diskrete Werte. Dadurch wird die Anzahl der Attributwerte verkleinert. Ein Problem der Diskretisierung ist: je weniger Klassen gebildet werden, desto ungenauer wird das Verfahren. Werden sehr viele Klassen gebildet, verringert sich die Anzahl der Attributwerte nur wenig und der Suchaufwand zur Bestimmung von Intervallen bleibt hoch.

Ein weiteres Problem bei der Diskretisierung ist, wie aus Wertebereichen Klassen gebildet werden können, die eine vernünftige Größe haben und vernünftige Elemente enthalten. Dafür gibt es mehrere Ansätze:

- **Klassenblinde Diskretisierung [BB]**
Obiges Beispiel ist eine klassenblinde Diskretisierung. Hier wird nur der Wertebereich des numerischen Attributs betrachtet. Die Diskretisierung geschieht meist mit Hilfe von statistischen.
- **Klassensensitive Diskretisierung [BB]**
Hier wird die Diskretisierung in Abhängigkeit von der Klassenzugehörigkeit eines Objekts (z.B.: Klasse der negativen, bzw. positiven Beispiele) durchgeführt.
- **Multivariante Diskretisierung [RC]**
Hier wird nicht nur ein Attribut diskretisiert. Mehrere numerische Attributwerte werden als Vektor betrachtet und in Abhängigkeit voneinander diskretisiert.

Der Vorteil der Diskretisierung ist, daß sie als Vorverarbeitungsschritt vor Programmablauf durchgeführt werden kann. Das Ergebnis wäre dann eine Partition des numerischen Wertebereichs. Die Aufgabe des Algorithmus besteht dann darin, Klassen dieser Partition so miteinander zu kombinieren, daß das Akzeptanzkriterium erfüllt ist. Diese Aufgabe ist im hier beschriebenen System aufgrund der Nichtmonotonie der Confidence erschwert. Hillclimbing-Strategien sind kaum anwendbar, weil es keinen guten Schätzwert gibt, der auf eine Erhöhung der Confidence hindeutet. Vollständiges Durchsuchen aller Kombinationen von Klassen ist bei hoher Klassenanzahl ineffizient.

In dieser Diplomarbeit wurde ein Verfahren entwickelt, daß auf nicht diskretisierten reellwertigen Wertemengen arbeiten kann. Dabei wurde besonders großer Wert auf Effizienz und Anwendbarkeit gelegt.

Da Hillclimbing- und Greedy-Methoden aufgrund der Nichtmonotonie der Confidence kaum anwendbar sind wurde ein speziell auf das Problem zugeschnittenes Prinzip verwendet.

Es wird versucht, globale Bereiche mit großem Support und Confidence lokalen Bereichen mit großem Support und Confidence abzuleiten. Dies bedeutet, aus rechten und linken Grenzen mit großem Support/ Confidence werden Intervalle mit großem Support/ Confidence abgeleitet. Aus Intervallen für jeweils eine Variable (Dimension) werden durch

Kombination mehrere Intervalle in einer Klausel bestimmt. Dieses Verfahren werden in den nächsten Abschnitten ausführlich dargestellt.

6.2 Bestimmung von Intervallen

Dieses Kapitel beschreibt das im Rahmen dieser Diplomarbeit entwickelte und implementierte Verfahren zur Bestimmung von Intervallen in Klauseln mit numerischen Attributen. Ziel des Verfahrens ist, die Extension von Klauseln mit zu kleiner Confidence so einzuschränken, daß die Confidence steigt.

Beispiel 6.2

Angenommen, die Klausel K

$$\text{accent}(S) \leftarrow \text{inword}(S, W), \text{tag}(W, \text{'NN'}), \text{volume}(S, V)$$

hat ausreichenden Support, aber eine zu kleine Confidence. V sei ein numerisches Attribut mit reellwertigem Wertebereich. Ein Gleichverbund über V zu einem anderen Literal scheint in den meisten Fällen nicht sinnvoll. Deshalb ist es nicht sinnvoll, ein neues Literal mit der Variablen V an die Formel anzufügen. Es gibt jedoch eine Möglichkeit, die Confidence der Klausel zu erhöhen. Dies geschieht durch Anfügen einer Beschränkung. Eine Art von Beschränkungen ist ein Intervall. Dies führt zur Formel K':

$$\text{accent}(S) \leftarrow \text{inword}(S, W), \text{tag}(W, \text{'NN'}), \text{volume}(S, V), V \in [100..200]$$

Durch diese Beschränkung werden Werte aus der Extension von $\text{body}(K)$ und aus $\text{body} \wedge \text{head}(K)$ entfernt. K' hat genau dann eine höhere Confidence als K, wenn durch die Beschränkung viele Werte aus $\text{body}(K)$ und wenig Werte aus $\text{body} \wedge \text{head}(K)$ entfernt werden.

Die Aufgabe der numerischen Komponente besteht also darin, Intervalle zu finden, die die Confidence einer Klausel erhöhen und den Support einer Klausel nur so viel verringern, daß er noch über dem erforderlichen Support liegt.

6.2.1 Integration in den bestehenden Algorithmus

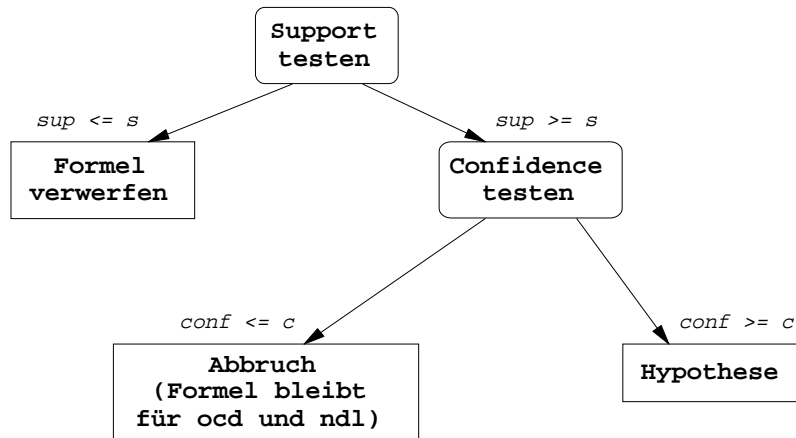
In diesem Abschnitt wird dargestellt, wie die numerische Komponente in das bestehende System integriert werden kann.

Die Intervallbestimmung wird immer dann aktiviert, wenn eine Formel zwar ausreichenden Support, jedoch eine zu kleine Confidence hat. Hat die Formel zu wenig Support, ist die Hinzunahme eines Intervalls nicht sinnvoll, weil der Support dadurch nur kleiner würde. Dies ist der Fall, weil eine Beschränkung eine Art von Spezialisierung einer Formel ist. Wegen der Monotonie des Support hat die speziellere Formel niemals einen größeren Support.

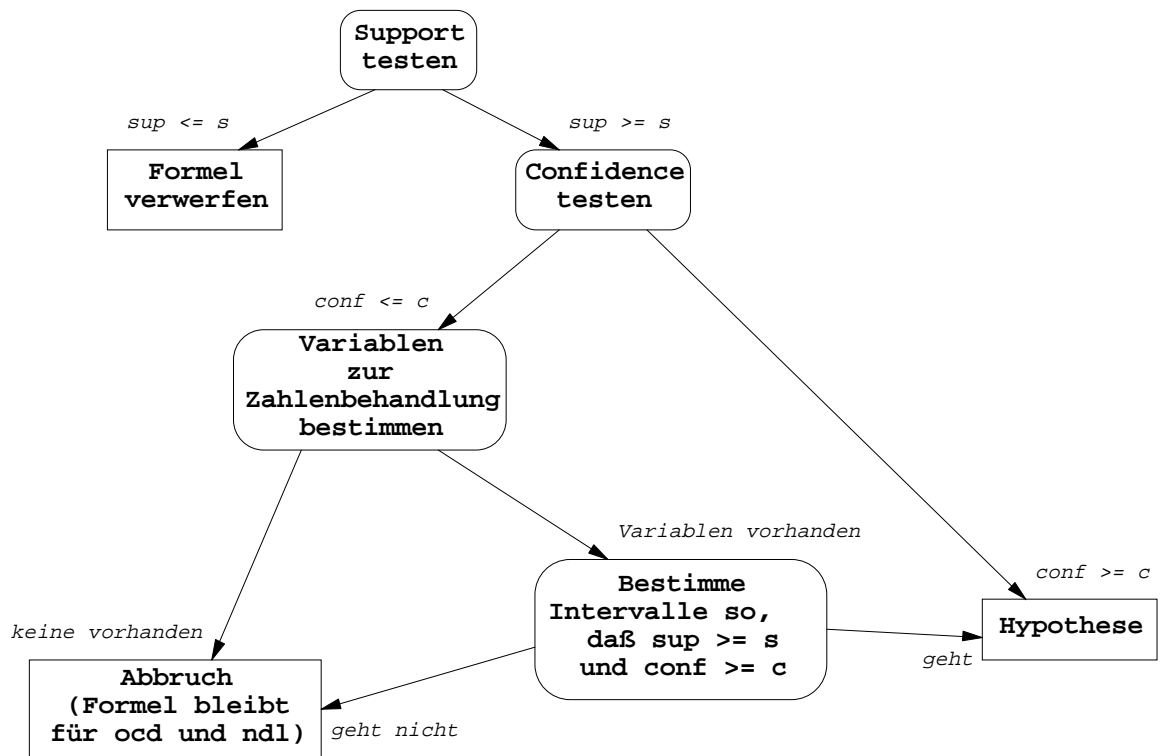
Im Gegensatz dazu kann die Confidence (die nicht monoton ist) steigen.

Der Programmablauf ändert sich folgendermaßen:

Bisheriger Ablauf:



Geänderter Ablauf:



Im folgenden wird die Komponente näher behandelt, die die Intervalle so bestimmt, daß Support und Confidence erfüllt sind.

Diese Komponente arbeitet nicht mit der Formel direkt, sondern mit den Extensionen von $\text{body}(K)$ und $\text{body}^{\wedge}\text{head}(K)$. Die Bestimmung erfolgt also in drei Schritten:

1. Berechnung der Extensionen
2. Bestimmung der Intervalle
3. Anfügen der gefundenen Intervalle an die Klausel

Die Schnittstelle zwischen numerischer Komponente und Data-Mining-Algorithmus besteht aus zwei Tabellen, die die Extension von $\text{Body}^{\wedge}\text{Head}$ (positive Beispiele), bzw. Body (negative Beispiele) der Klausel enthalten, an die numerischen Beschränkungen angefügt werden sollen. Jede dieser Tabellen hat Spalten, in denen die Kopfattribute stehen und Spalten mit numerischen Attributen.

Kopfattribut 1	Kopfattribut 2	num. Attribut 1	num. Attribut 2
...

Diese Tabellen müssen mit Hilfe von SQL-Anfragen erzeugt werden.

Die schwierigste Aufgabe stellt Punkt 2 (Bestimmung der Intervalle) dar. Deshalb ist dieser Punkt Thema des restlichen Kapitels.

Die Beschreibung des Algorithmus geschieht anhand eines Beispiels. Dieses Beispiel wird im nächsten Abschnitt vorgestellt.

6.2.2 Beispiel: Lautstärke und Frequenz

Eine Klausel K

$$\text{accent}(S) \leftarrow \text{inword}(S, W), \text{tag}(W, \text{'NN'}), \text{volume}(S, V), \text{frequency}(S, F)$$

erreicht den erforderlichen Support, hat aber zu wenig Confidence. Der erforderliche Support beträgt im Beispiel 0,22, die erforderliche Confidence 0,85. Der Benutzer hat angegeben, daß für die Variablen V und F eine Intervallbestimmung durchgeführt werden soll. Es sollen also Intervalle bestimmt werden, die aussagen, welche Lautstärke und welche Frequenz betonte Nomen haben.

Hier sollte angemerkt werden, daß dieses Beispiel nur zur Illustration des Algorithmus dient und keineswegs den Zusammenhängen in der Wirklichkeit entspricht.

Zuerst werden die Extensionen der Klausel berechnet:

Body \wedge Head

V	F
42	24
30	23
25	45
46	30
33	33
29	29
...	...

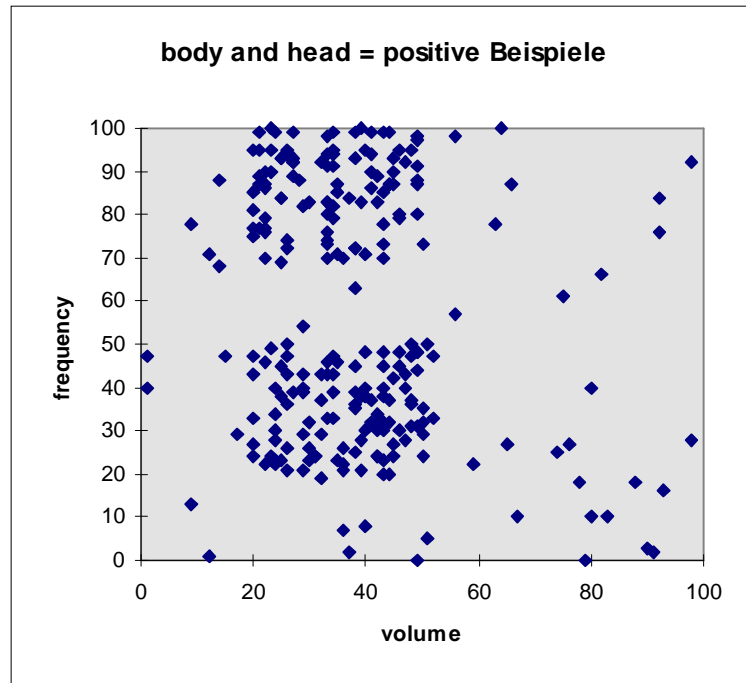
Body

V	F
42	24
30	23
25	45
46	30
33	33
29	29
...	...
71	17
69	5
91	15
51	70
57	18

Die komplette Extension des Beispiels enthält Anhang A.

Man erkennt, daß die Extension von $\text{body} \wedge \text{head}(K)$ eine Teilmenge der Extension von $\text{body}(K)$ ist. Die Extension von $\text{body} \wedge \text{head}(K)$ wird im folgenden auch als Menge der positiven Beispiele bezeichnet. Die Extension von $\text{body}(K)$ ist die Vereinigung von positiven und negativen Beispielen. Die negativen Beispiele sind demzufolge $\|\text{body}\| \setminus \|\text{body} \wedge \text{head}\|$. Die Berechnung der negativen Beispiele ist mit Hilfe von SQL sehr aufwendig, weil SQL keine Operation zur Bildung der Mengendifferenz zur Verfügung stellt. Die Berechnung ist nur mit Hilfe mehrerer „not in (select...)“ - Subanfragen möglich. Deshalb sollte die Intervallbestimmung ohne die Explizite Angabe der negativen Beispiele auskommen. Trotzdem wird im folgenden zur Erhöhung der Verständlichkeit oft von negativen Beispielen gesprochen.

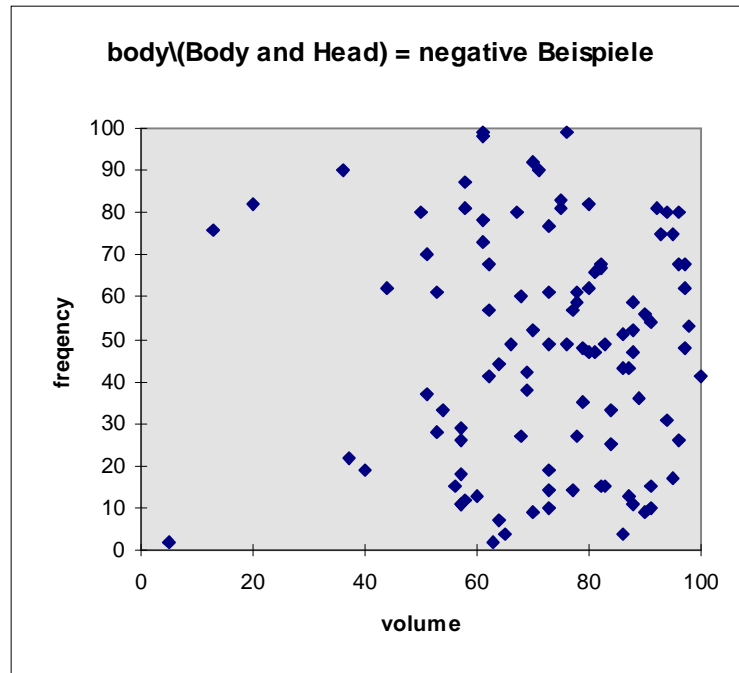
Die Verteilung der Lautstärke- und Frequenzwerte kann an Diagrammen illustriert werden.



Man erkennt, daß sich die positiven Beispiele, also die betonten Silben in drei Bereichen häufen. Diese Bereiche haben die Form von Rechtecken. Diese können auch jeweils durch zwei Intervalle dargestellt werden.

1. $V \in [20..50] \wedge F \in [20..50]$
2. $V \in [20..50] \wedge F \in [70..100]$
3. $V \in [20..50] \wedge F \in [20..100]$

Diese Intervalle kommen als Beschränkungen für die Klausel in Betracht. Jedoch sind auch die negativen Beispiele zu berücksichtigen:



Es werden also rechteckige Bereiche gesucht, die möglichst viele positive Beispiele (betonte Silben) und möglichst wenig negative Beispiele (unbetonte Silben) enthalten.

6.2.3 Mögliche Ansätze

Das größte Problem bei der Bestimmung von Beschränkungen liegt in der Effizienz. An die Mengen der positiven und negativen Beispiele müssen oft Anfragen, wie „wieviele betonte Silben liegen im Bereich $F \in [10..30] \wedge V \in [20..60]$ “. Diese Anfragen erfordern jeweils das Durchsuchen der kompletten Beispielmenge (die Beispielmenge liegt nicht sortiert vor). Deshalb gilt es die Menge von Anfragen zu reduzieren.

Die Frage ist also, wie man mit möglichst wenig Anfragen ein Rechteck (im mehrdimensionalen Raum ein Hyperrechteck) finden kann, das genug Support und Confidence hat. Folgende Möglichkeiten wurden in Erwägung gezogen:

1. Vollständiges Durchsuchen

Die einzige korrekte und vollständige Art der Intervallbestimmung ist das triviale Durchprobieren aller möglichen Intervalle des Wertebereichs. Für eine Variable mit n

Werten in den Beispielmengen kommen $\sum_{i=1}^n i \in O(n^2)$ Intervalle in Betracht (Jede linke

Intervallgrenze kann alle größeren Werte als rechte Intervallgrenze haben). Bei $|v|$ zu bestimmenden Variablen müssen $O(n^{|v|-2})$ Intervalle getestet werden. Bei einem

(durchaus realistischen) Wertebereich mit 100000 Elementen und den zwei Variablen Frequenz und Lautstärke ständen ungefähr $100000^{2 \cdot 2} = 10^{20}$ Intervalle zum Test an. Diese Komplexität ist bei großen Wertebereichen nicht akzeptabel.

2. Hillclimbing

Der zweite untersuchte Ansatz besteht darin, von einem Rechteck maximaler Größe ausgehend, dieses solange zu verkleinern, bis die Confidence groß genug ist. Dabei wird das Rechteck von der Seite gedrückt, die den maximalen Zuwachs an Confidence bietet. Dabei ergeben sich jedoch mehrere Probleme. Das größte Problem ist, daß man bei einem Rechteck ankommen kann, das den Mindestsupport nicht mehr erreicht. Dann gibt es nur eine Möglichkeit, nämlich Backtracking. Im schlimmsten Fall kann das Hillclimbing-Verfahren zum vollständigen Durchsuchen degenerieren. Ein anderes Problem besteht darin, zu entscheiden, wie weit das Rechteck von einer Seite her gedrückt werden soll. Die Schrittweite ist vom Wertebereich abhängig. Den nächsten Wert innerhalb des Rechtecks zu finden kostet jedoch $O(n)$ Schritte und ist damit nicht akzeptabel. Eine feste Schrittweite deckt den Wertebereich eventuell nicht gut genug ab. Die Folge ist, daß Hypothesenklauseln übersehen werden können.

Ein weiteres Problem liegt an der Entscheidung, in welche Richtung das Rechteck gedrückt werden soll. Wenn das Suchproblem $|v|$ Variablen zu bestimmen hat, kommen $2^{|v|}$ Richtungen in Betracht. Das bedeutet, daß für jede solche Entscheidung $2^{|v|}$ Durchläufe durch die Beispielmengen notwendig sind.

Alle diese Schwierigkeiten führten dazu, daß dieser Ansatz nicht implementiert wurde.

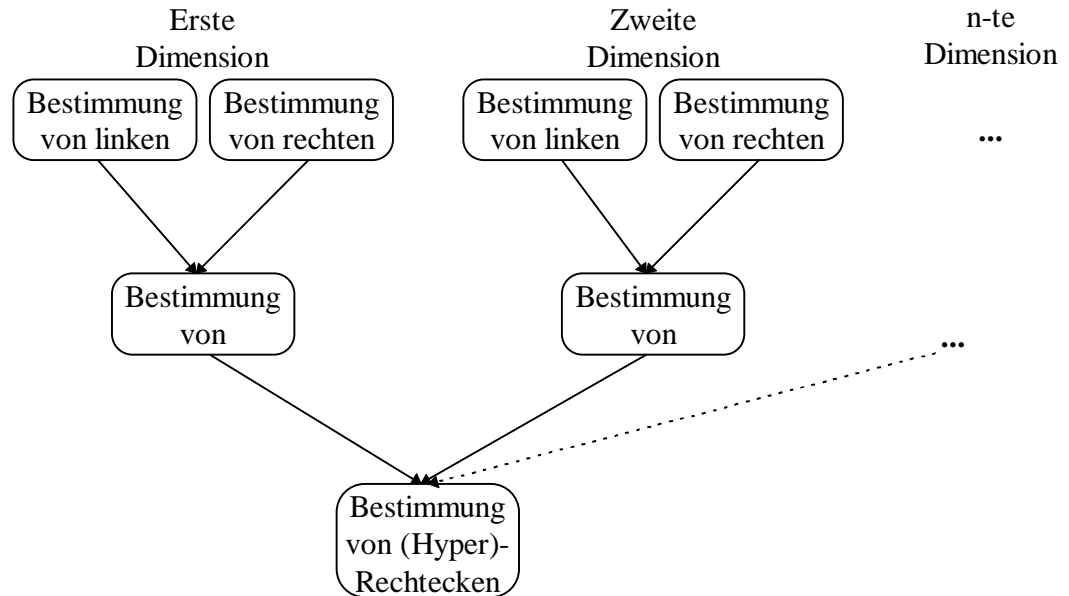
3. Heuristische Berechnung durch Übertragung lokaler Maxima auf globale Maxima

Dieser dritte Ansatz versucht einen möglichst guten Kompromiß aus Effizienz und Genauigkeit zu finden, wobei größerer Wert auf Effizienz gelegt wurde. Das Prinzip ist folgendes:

Zuerst werden linke und rechte Intervallgrenzen getrennt berechnet, die, wenn man sie miteinander kombiniert, eine hohe Confidence und hohen Support versprechen. Dies sind Grenzen, die schon allein einen großen Support haben, d.h. die linke Grenze v_l für die Variable V ist vielversprechend, wenn die Beschränkung $V > v_l$ eine hohe Confidence und hohen Support hat. Eine rechte Grenze v_r ist vielversprechend, wenn $V < v_r$ eine hohe Confidence hat. Aus allen vielversprechenden Grenzen werden dann durch Kombination Intervalle für eine Variable erzeugt. Für diese Intervalle wird jeweils ein Qualitätsmaß berechnet. Die besten Intervalle werden zur Bildung des (Hyper-) Rechtecks mit den besten Intervallen anderer Variablen kombiniert. Der nächste Abschnitt beschreibt das Verfahren genauer.

6.2.4 Heuristische Berechnung von Intervallen

Das im letzten Abschnitt beschriebene Verfahren läßt sich folgendermaßen darstellen:

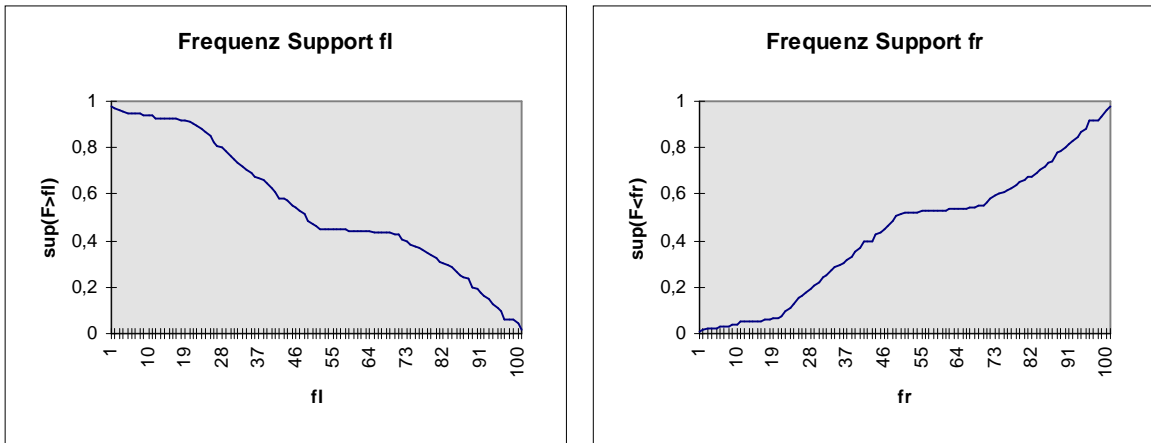


Die Heuristik, die Dabei verwendet wird ist folgende:

Beschränkungen der Form $f > f_l$, bzw. $f < f_r$ mit hohem Support und Confidence deuten auf Beschränkungen $f \in [f_l..f_r]$ mit hohem Support und Confidence hin. Beschränkungen (Intervalle) $f \in [f_l..f_r]$, bzw. $f \in [v_l..v_r]$ mit hohem Support/Confidence deuten auf Beschränkungen (Hyperrechtecke) $f \in [f_l..f_r] \wedge f \in [v_l..v_r]$ mit hohem Support/Confidence hin.

6.2.4.1 Berechnung von linken und rechten Grenzen

Der Abschnitt handelt davon, wie linke und rechte Intervallgrenzen getrennt voneinander berechnet werden können. Gesucht sind linke Grenzen v_l , so daß die Beschränkung $V > v_l$ hohen Support und Confidence hat und rechte Grenzen v_r , so daß die Beschränkung $V < v_r$ hohen Support und Confidence hat. Zur Veranschaulichung kann der Support in Diagrammen dargestellt werden, deren x-Achse alle Werte der zu bestimmenden Variablen enthält, und deren y-Achse der Support des Bereichs darstellt, der sich aus den kumulierten Beispielanzahlen berechnet. Hier sollen nur die Diagramme zur Berechnung der Grenzen für die Frequenz betrachtet werden.



Das linke Bild zeigt die Bestimmung der linken Intervallgrenze f_l für die Frequenz. $\text{Sup}(F>f_l)$ ist der Support, der sich ergibt, wenn die Klausel um die Beschränkung $F>f_l$ erweitert wird. Diese Kurve ist monoton fallend, weil die Anzahl der positiven Beispiele mit steigendem v_l immer kleiner wird, die Extension des Kopfes der Klausel aber konstant bleibt (im Beispiel ist die Head-Extension 250). Aus dieser Kurve kann ein Wert \bar{f}_l bestimmt werden, für den der Support gerade noch ausreichend ist. Alle größeren linken Grenzen haben den Support nicht. Alle kleineren linken Grenzen führen zu Extensionen, die groß genug sind, um den Support zu erreichen.

Beispiel 6.3

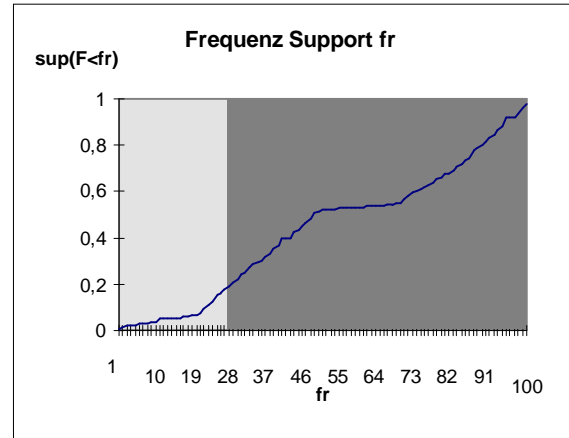
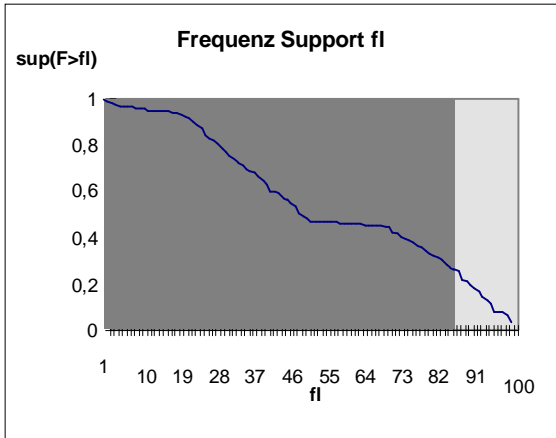
Im gegebenen Beispiel beträgt der erforderliche Support 0,22. Aus dem Diagramm ist entnehmbar, daß dieser Support bei $f_l=87$ erreicht wird. Das heißt, daß für die Beschränkung $F>87$ der Support gerade noch erfüllt ist. Für die Beschränkung $F>88$ ist der Support zu klein. Für alle Beschränkungen $F>f_l$ mit $f_l \leq 87$ ist der Support erfüllt.

Das rechte Bild zeigt die Bestimmung einer rechten Intervallgrenze. Hier verläuft die Kurve monoton steigend. Die Bestimmung der restlichen Werte verhält sich analog zur linken Grenze.

Beispiel 6.4

Der Schwellenwert für die rechte Grenze liegt hier bei $f_r=30$.

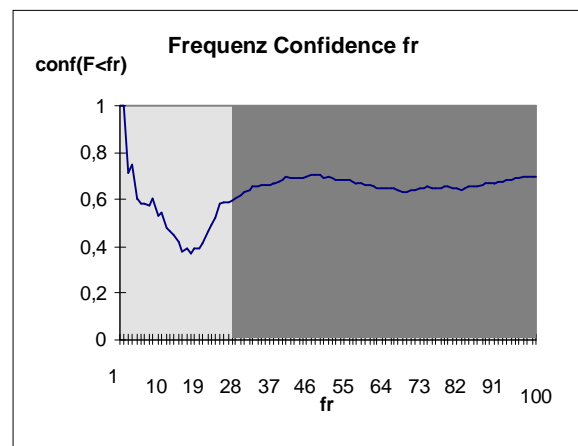
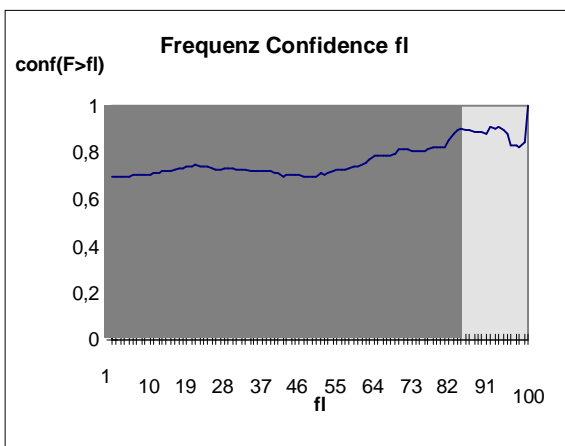
Durch diese Beobachtungen kann der Suchraum, in dem nach linken bzw. rechten Grenzen gesucht werden muß eingeschränkt werden. Das folgende Bild zeigt die Suchräume als Schraffierung.



Die Suchräume werden in der vorliegenden Implementierung mit Hilfe von binärer Suche ermittelt. Die Genauigkeit (also, wann die Suche abbrechen soll) kann vom Benutzer angegeben werden. Mit binärer Suche werden jedoch schon mit wenigen Schritten hohe Genauigkeiten erzielt. Die Bestimmung der Support-Grenze hat auf jeden Fall weniger als logarithmischen Aufwand (in der Anzahl der positiven Beispiele).

Nachdem der Suchraum mit obiger Methode eingeschränkt wurde, müssen linke, bzw. rechte Grenzen berechnet werden, die eine hohe Confidence versprechen. Die Problematik bei der Suche nach Grenzen mit hoher Confidence gestaltet sich ähnlich wie beim Support. Der größte Unterschied liegt darin, daß die Confidence keine Monotonie zeigt. Dies verhindert effiziente Methoden, wie binäre Suche oder Hillclimbing.

Gesucht sind linke, bzw. rechte Grenzen, die eine hohe Confidence versprechen. **Dies sind in der folgenden Kurve alle Punkte im Suchraum (schraffiert), die Hochpunkte sind und nicht weit unter der erforderlichen Confidence liegen.**



Es ist möglich, daß Punkte im Suchbereich gefunden werden, deren Confidence ausreichend ist. Das würde bedeuten, daß die komplette Intervallbestimmung beendet wäre, denn man hätte damit eine Beschränkung gefunden, die das Akzeptanzkriterium

erfüllt. In der Regel kann man jedoch nur Hochpunkte finden, deren Confidence bestenfalls nicht weit unter der erforderlichen Confidence liegt. Man hofft, daß durch Kombination von linken und rechten Grenzen die Confidence so weit ansteigt, daß sie groß genug wird, der Support aber nicht so weit sinkt, daß er kleiner als der erforderliche Support wird.

Beispiel 6.5

Aus den Schaubildern lassen sich folgende linke Grenzen ablesen:

$$f_l \in \{24, 50, 64, 70, 78\}$$

$$f_r \in \{41, 48, 76, 100\}$$

Keine dieser Grenzen hat allein die erforderliche Confidence von 0,85.

Wie können diese Maxima bestimmt werden, ohne daß die gesamte Confidence-Kurve aufgebaut werden muß (das hätte einen Aufwand von n^2 bei n Beispielwerten). In der vorliegenden Implementierung werden eine vom Benutzer bestimmbare Anzahl von Stützstellen in den Wertebereich von f gelegt. Für diese Stützstellen wird die Confidence berechnet. Damit erhält man eine Kurve, die die eigentliche Confidence-Kurve annähert. Die Stützwerte werden dabei einfach durch Geraden verbunden. Dies ist die einfachste und schnellste Interpolationsmethode, die es gibt. Andere Methoden, wie Spline-Interpolation wären sehr aufwendig und es ist zu bezweifeln, ob sie eine Verbesserung der Ergebnisse bringt (Die Kurve wird dabei eher geglättet, was bei der Suche nach Hochpunkten eher hinderlich ist).

Aus der angenäherten Kurve werden dann alle linke, bzw. rechten Grenzen bestimmt, deren Confidence nur einen (vom Benutzer bestimmbaren) Prozentsatz unter der erforderlichen Confidence liegen. Die Confidencewerte der bestimmten Grenzen zeigt die folgende Tabelle.

F_l	$\text{conf}(f > f_l)$	f_r	$\text{conf}(f < f_r)$
24	0,74	41	0,67
50	0,71	48	0,71
64	0,78	76	0,76
70	0,82	100	0,70
78	0,83		

Die erforderliche Confidence beträgt 0,85. Wäre der Prozentsatz bei 80%, so würden alle Grenzen zur Intervallberechnung zugelassen, die einen Confidence-Wert von mindestens 0,68 haben. Der Wert $f_r=41$ würde also entfallen.

Bei großen Wertebereichen besteht das Problem, daß eventuell sehr viele Hochpunkte mit hoher Confidence gefunden werden. Deshalb kann der Benutzer die Anzahl der Grenzen beschränken. Angenommen im Beispiel sei die Anzahl der Grenzen auf die besten 4 beschränkt, dann wäre die Ergebnismenge der Grenzenberechnung

F_l	$\text{conf}(f > f_l)$	f_r	$\text{conf}(f < f_r)$
24	0,74	48	0,71
64	0,78	76	0,76
70	0,82	100	0,70
78	0,83		

Im nächsten Schritt müssen aus diesen Grenzen Intervalle berechnet werden, die eventuell eine höhere Confidence als die Grenzen selbst haben. Dies wird im nächsten Abschnitt beschrieben.

6.2.4.2 Berechnung von Intervallen

Aus l linken und r rechten Grenzen können maximal $l \cdot r$ Intervalle geformt werden. Mit der Menge von Grenzen aus dem letzten Abschnitt ergeben sich für die Frequenz folgende Intervalle:

- [24..48], [24..76], [24..100]
- [64..76], [64..100]
- [70..76], [70..100]
- [78..100]

Diese Intervalle beschreiben decken jeweils kleinere Beispielmengen ab als die Grenzen aus denen sie zusammengesetzt sind. Das hat zur Folge, daß der Support kleiner wird. Deshalb muß der Support für alle Intervalle erneut getestet werden. Alle Intervalle mit zu kleinem Support können sofort entfallen.

Die nächste Tabelle zeigt den Support der Intervalle

Intervall	support
[24..48]	0,38
[24..76]	0,50
[24..100]	0,85
[64..76]	0,09
[64..100]	0,44
[70..76]	0,08
[70..100]	0,42
[78..100]	0,34

Die gestreiften Zeilen haben den erforderlichen Support nicht und werden deshalb verworfen.

Die übrigen Intervalle haben genug Support. Für diese Intervalle muß als nächstes die Confidence getestet werden. Falls Die Confidence eines Intervalls groß genug ist, wird dieses Intervall an die Klausel angehängt und die Klausel als Hypothesenklausel akzeptiert. Im Beispiel ist dies, wie die folgende Tabelle zeigt nicht der Fall.

Intervall	confidence
[24..48]	0,79
[24..76]	0,69
[24..100]	0,74
[64..100]	0,78
[70..100]	0,81
[78..100]	0,82

In diesem Fall werden die Intervalle zusammen mit Intervallen für andere Variablen in der Klausel (diese wurden genauso bestimmt) zur Berechnung von (Hyper-) Rechtecken bereitgestellt. Auch die Anzahl dieser Intervalle kann sehr hoch sein, deshalb kann der Benutzer die Anzahl beschränken. Falls der Benutzer nur die 5 besten Intervalle zur Berechnung von (Hyper-) Rechtecken zulassen will, ist die Ergebnismenge der Intervallbestimmung folgende:

- $f \in [24..48]$
- $f \in [24..100]$
- $f \in [64..100]$
- $f \in [70..100]$
- $f \in [78..100]$

Wie aus den Intervallen (Hyper-) Rechtecke berechnet werden, die vielleicht eine höhere Confidence haben, beschreibt der nächste Abschnitt.

6.2.4.3 Berechnung von Hyper-Rechtecken

Die Berechnung von Hyperrechtecken verläuft analog zur Berechnung von Intervallen. Zuerst werden die Intervalle aller Variablen (Dimensionen) der Klausel zu Hyperrechtecken kombiniert. Für alle diese Hyperrechtecke wird der Support berechnet. Die folgende Tabelle enthält alle Rechtecke mit ihren Support-Werten. Dabei wird angenommen, daß bei der Bestimmung von Intervallen für die Lautstärke (die zweite Variable der Formel) die beiden Intervalle

- $v \in [31..64]$
- $v \in [20..50]$

berechnet wurden.

Intervall	Support
$f \in [24..48] \wedge v \in [31..64]$	0,25
$f \in [24..100] \wedge v \in [31..64]$	0,51
$f \in [64..100] \wedge v \in [31..64]$	0,25
$f \in [70..100] \wedge v \in [31..64]$	0,25
$f \in [78..100] \wedge v \in [31..64]$	0,20
$f \in [24..48] \wedge v \in [20..50]$	0,34

$f \in [24..100] \wedge v \in [20..50]$	0,75
$f \in [64..100] \wedge v \in [20..50]$	0,39
$f \in [70..100] \wedge v \in [20..50]$	0,38
$f \in [78..100] \wedge v \in [20..50]$	0,31

Die gestreifte Zeile hat nicht genug Support und entfällt deshalb. Für die restlichen Rechtecke wird die Confidence berechnet. Erreicht ein Rechteck die erforderliche Confidence, wird dieses Rechteck als Beschränkung an die Formel angefügt und die Formel wird als Hypothesenklausel anerkannt. Dies ist hier zum Beispiel bei dem Rechteck

$$f \in [24..48] \wedge v \in [31..64]$$

der Fall. Die Confidence beträgt 0,9 und ist damit groß genug. Also wird die Klausel

$$\text{accent}(S) \leftarrow \text{inword}(S, W), \text{tag}(W, \text{'NN'}), \text{volume}(S, V), \text{frequency}(S, F), \\ f \in [24..48] \wedge v \in [31..64]$$

Als Hypothese anerkannt.

Findet der Algorithmus keine Klausel mit ausreichendem Support und Confidence, so wird die Klausel ohne Beschränkung zur Generierung neuer Klauseln bereitgestellt.

6.2.4.4 Parameter der numerischen Komponente

Die beschriebene numerische Komponente hat mehrere Parameter. Damit kann die Bestimmung von Beschränkungen in Bezug auf Effizienz oder Genauigkeit manipuliert werden. Die Einstellung dieser Parameter ist abhängig von der Problemstellung, insbesondere vom Wertebereich der numerischen Attribute.

Folgende Tabelle enthält alle Parameter des Systems:

Qualitätsmaß für Beschränkungen	Formel, mit der berechnet wird, wie gut eine Beschränkung bei der Kombination mit anderen Beschränkungen wieder hohe Support- und Confidence-Werte ergibt. Das heißt, wie gut die Heuristik auf diese Beschränkung zutrifft. Zum Beispiel, wie stark eine Grenze auf ein Intervall mit hohen Support- und Confidence-Werten hindeutet. Mögliche Qualitätsmaße sind: <ul style="list-style-type: none"> • Die Summe aus Support und Confidence • Nur die Confidence (diese ist schwieriger zu erreichen als der Support, der bei den gegebenen Formeln ohne Beschränkungen ja schon groß genug ist) • evtl. andere
Anzahl der Grenzen	Absolute Anzahl der linken, bzw. rechten Grenzen, die zur Bildung von Intervallen verwendet werden sollen. Der

	Algorithmus berechnet alle Hochpunkte der Confidence und gibt die Besten (siehe Qualitätsmaß für Beschränkungen) zurück.
Anzahl der Intervalle	Absolute Anzahl der Intervalle, die zur Bildung von Hyperrechtecken verwendet werden sollen. Der Algorithmus bestimmt für alle Intervalle das Qualitätsmaß (s.o.) und gibt die Besten zurück.
Unterschied zwischen erforderlicher und erreichter Confidence	Die Confidence von Grenzen und Intervallen ist meist kleiner als die erforderliche Confidence. Beschränkungen mit sehr kleiner Confidence führen durch Kombination mit anderen Beschränkungen kaum zu genügenden Confidence-Werten. Deshalb kann der Benutzer einen Faktor (<i>conf_relax</i>) <i>f</i> angeben. Alle Beschränkungen, die kleinere Confidence als <i>erf. Conf. * conf_relax</i> sind, werden verworfen.
Anzahl der Stützstellen zur Bestimmung von Grenzen	Der Benutzer kann angeben, an wievielen Stellen die Confidence-Werte zur Ermittlung von Hochpunkten berechnet werden sollen. Mehr Stützstellen bedeuten höhere Genauigkeit bei größerem Aufwand.
Differenz zwischen erforderlichem und erreichtem Support	Zur Bestimmung der Support-Grenze mit Binärer Suche wird als Abbruchkriterium die Differenz zwischen erforderlichem Support und dem Support an der Stützstelle verwendet. Kleine Differenzen bedeuten höhere Genauigkeit bei größerem Aufwand.

6.2.4.5 Algorithmus

Diese Prozedur bestimmt ein Hyperrechteck, so daß Support und Confidence für die Klausel erfüllt sind. Die Klausel mit dem gefundenen Hyperrechteck wird zurückgegeben. Falls kein Hyperrechteck gefunden wird KEINE_KLAUSEL zurückgegeben.

```

Klausel bestimme_hyperrechteck (Klausel K)
{
  FOR dimension = 1 TO ANZAHL_DIMENSIONEN(K)
  {
    intervalliste(dimension) =
      bestimme_intervalle(K, dimension);
    IF HYPOTHESE_GEFUNDEN
      RETURN intervalliste(dimension)
  }
  rechteckliste = kombiniere_intervalle_zu_rechtecken
  FOR EACH rechteck IN rechteckliste
  {
    IF sup(rechteck) > s THEN
      IF conf(rechteck) > c THEN
        RETURN KLAUSEL(rechteck)
  }
}

```

```

    // Wenn man hier ankommt, wurde kein Rechteck
    gefunden
    RETURN KEINE_KLAUSEL
}

```

Diese Prozedur bestimmt eine Menge von Intervallen, jedoch nicht mehr als $|I|$ (vom Benutzer angegebener Parameter). `Conf_relax` ist der Prozentsatz, um die die Confidence unter der erforderlichen Confidence liegen darf, damit das Intervall akzeptiert wird.

```

Intervallliste  bestimme_intervalle  (Klausel  K,
Dimension dim)
{
  linke_grenzen = bestimme_grenzen(K, dim, LINKS);
  IF HYPOTHESE_GEFUNDEN
    RETURN intervall [linke_grenze..max(dim)];
  rechte_grenzen = bestimme_grenzen(K, dim, RECHTS);
  IF HYPOTHESE_GEFUNDEN
    RETURN intervall [min(dim)..rechte_grenze];
  intervalle = kombiniere_grenzen_zu_intervallen
  FOR EACH intervall IN intervalle
    IF sup(intervall) > s THEN
      IF conf(intervall) > c THEN
        HYPOTHESE_GEFUNDEN = TRUE
        RETURN intervall;
      IF conf(intervall) > c*conf_relax THEN
        intervalliste = intervalliste + intervall
  RETURN best_intervalls(intervalliste, |I|)
}

```

Diese Prozedur berechnet die linken bzw. rechten Grenzen eines Intervalls. Die maximale Anzahl ist $|G|$. Zur Bestimmung werden $|st|$ Stützstellen verwendet. `Conf_relax` ist der Prozentsatz, um die die Confidence unter der erforderlichen Confidence liegen darf, damit die Grenze akzeptiert wird.

```

Grenzenliste  bestimme_grenzen(Klausel  K,  Dimension
dim,
                               Seite l/r)
support_grenze = bestimme_support_grenze;
stützstellenliste = bestimme_stützstellen(|st|);
confidence_maxima = bestimme_confidence_maxima(|G|)
FOR EACH grenze IN confidence_maxima
  IF sup(grenze) > s THEN
    IF conf(grenze) > c THEN
      HYPOTHESE_GEFUNDEN = TRUE
      RETURN grenze;
    IF conf(grenze) > c*conf_relax THEN
      grenzenliste = grenzenliste + grenze
RETURN best_grenzen(grenzenliste, |G|)

```

6.2.4.6 Laufzeit

Die Bestimmung einer Liste von Grenzen benötigt zum Aufbau der Confidence-Kurve $|st|$ Durchläufe durch die Datenbank. Die Bestimmung von Grenzen hat also bei n Werten im Attributwertbereich die Komplexität

$$|st| \cdot n$$

Die Berechnung der Intervalle benötigt bei jeweils $|G|$ linken und rechten Grenzen maximal

$$|G|^2 \cdot n$$

Schritte zum Test der Intervalle.

Da für jedes Intervall linke und rechte Grenzen berechnet werden müssen, ist die Gesamtkomplexität der Intervallberechnung

$$2 \cdot |st| \cdot n + |G|^2 \cdot n = (2 \cdot |st| + |G|^2) \cdot n$$

Dieser Aufwand wird für jede Dimension benötigt. Zusätzlich müssen die Rechtecke getestet werden, die sich aus der Anzahl der Intervalle jeder Dimension ergeben. Dies sind

$$|I|^{|\text{dim}|}$$

Rechtecke, die getestet werden müssen.

Die Gesamtkomplexität der Beschränkungsbestimmung ist also im schlechtesten Fall

$$|\text{dim}| \cdot (2 \cdot |st| + |G|^2 + |I|^{|\text{dim}|}) \cdot n$$

Wie man sieht, ist die Komplexität nur noch linear in n . Bei vielen Dimensionen kann die Anzahl der Datenbankdurchläufe sehr groß werden. Bei wenigen Dimensionen ist die Komplexität durchaus akzeptabel.

6.3 Versuch mit einer Diabetes-Datenbank

Die in diesem Kapitel beschriebene numerische Komponente wurde auf eine Datenbank angewendet, die verschiedene numerische Meßwerte von weiblichen Patienten enthält, die einen Verdacht auf Diabetes hatten¹. Die Datenbank enthält 768 Datensätze, davon stammen 500 von Patienten, bei denen der Befund positiv war (positive Beispiele) und 268 von Patienten mit negativem Befund. Die Datenbank hat folgende Attribute:

¹ Es handelt sich um die Pima Indians Diabetes Database

schw	Anzahl der bisherigen Schwangerschaften
pk	Plasma Glucose-Konzentration 2 Stunden nach einem oralen Glucose-Toleranz-Test
db	Diastolischer Blutdruck (mm Hg)
th	Trizeps-Hautdicke (mm)
si	2 Stunden Serum Insulin (mu U/ml)
gg	(Gewicht des Patienten / Größe des Patienten) ²
dp	Diabetes-Stammbaum-Funktion
al	Alter

Mit dieser Datenbank wurde eine Laufzeitanalyse durchgeführt. Folgende Tabelle enthält alle Parameter, die sich auf die Laufzeit auswirken.

Support	Confidence	Anz. Grenzen	Anz. Stützwerte	Anz. Intervalle	Laufzeit (s)
0,6	0,85	10	50	3	165
0,6	0,86	10	50	1	13
0,6	0,86	30	200	1	13
0,6	0,86	10	50	2	30
0,6	0,86	10	50	3	166
0,6	0,86	10	50	4	914
0,6	0,87	10	50	4	922

Auffallend ist, daß ab einer Confidence von 0,87 keine Formeln mehr gefunden werden. Leider wurde die Datenbank bisher nur zur Klassifikation verwendet. Deshalb war eine Kontrolle der Ergebnisse nicht möglich. Bei allen Versuchen wurde folgende Beschränkung gefunden

$$\text{schw} > 0 \wedge \text{pk} \in [0..127,4] \wedge \text{gg} \in [0..40,8]$$

Der für die Laufzeit relevante Parameter ist die Anzahl der Intervalle pro Dimension. Allerdings liefert der Algorithmus schon bei einem Intervall pro Dimension zwei Ergebnisse. Da der Algorithmus abbricht, wenn er eine Klausel gefunden hat, würde in diesem Fall diese schnelle Konfiguration ausreichen.

6.4 Zusammenfassung

In diesem Kapitel wurde eine Komponente entwickelt, die automatisch Beschränkungen für eine Klausel ermittelt, so daß Support und Confidence der Klausel erfüllt sind. Dabei wurde großer Wert auf Effizienz gelegt. Dadurch wird die Suche nach

Hypothesenklauseln unvollständig. Der Algorithmus kann eventuell Beschränkungen übersehen, die das Akzeptanzkriterium erfüllen würden. Dies liegt zum einen daran, daß bei der Berechnung von linken und rechten Grenzen die Kurve für die Confidence nur angenähert vorliegt. Zum anderen trifft die angewandte Heuristik nicht in allen Fällen zu. Die Unvollständigkeit des Algorithmus muß jedoch in Kauf genommen werden, wenn man die Größe des Hypothesenraums und die Größe der Attributwertemengen in Betracht zieht. Ein großes Effizienzproblem liegt auch darin, daß jede Klausel, die den Support nicht erfüllt und Variablen besitzt, für die Beschränkungen bestimmt werden sollen, durch den ganzen Bestimmungsalgorithmus laufen muß. Dabei ist oft der Fall, daß keine passende Beschränkung gefunden wird. Dies bedeutet, daß die Klausel durch OCD oder NDL um ein Literal erweitert wird. Die neue Klausel steht oft wieder zur Beschränkungsbestimmung an. Wieder wird keine gefunden, usw.

Die Bestimmung von Beschränkungen erfordert viele Durchläufe durch die Datenbank. Da SQL-Anfragen für eine solche Anwendung zu langsam sind, werden in der aktuellen Implementierung die Extensionen der positiven und negativen Beispielmengen der Klausel ohne Beschränkung mit einem SQL-Statement ermittelt. Diese Beispielmengen werden dann in einer eigenen Datenstruktur im Hauptspeicher gehalten. Auf dieser Datenstruktur können Extensionen von Beschränkungen schnell ermittelt werden.

7

Zusammenfassung und Ausblick

Für den von Irene Weber entwickelten Algorithmus wurde von Thomas Klenk, Irene Weber und mir ein System entworfen und in C implementiert. Dabei kam die relationale Datenbank Ingres zum Einsatz. Das System wird über TCP/IP an diese Datenbank gekoppelt. Das bedeutet, daß der Algorithmus nicht auf dem selben Rechner laufen muß, wie die Datenbank. Die Implementierung beinhaltet die Verwaltung von Klauseln, die Erzeugung von SQL-Anfragen zur Berechnung der Support- und Confidence-Werte für eine Klausel, die Sortierung von Kandidatenklauseln mit Hilfe der Subsumption und die Generierung neuer Klauseln aus Klauseln, die zwar genug Support, aber zu wenig Confidence erreichen. Die Architektur dieses Systems wurde in Kapitel 4 beschrieben.

Die Aufgabe der Diplomarbeit bestand in der Anwendung des Algorithmus auf einer Datenbank, die phonetische Daten von Radiosendungen enthält. Dieser Schritt sollte Schwächen und Verbesserungsmöglichkeiten des Algorithmus aufzeigen. Dazu wurden aus der phonetischen Datenbank Prädikate konstruiert, die dem Algorithmus als Eingabe dienen.

Das Ergebnis des ersten Versuchs mit dem Algorithmus war, daß er in dieser Form nur bedingt anwendbar ist. Die Gründe dafür und Verbesserungsvorschläge wurden in Kapitel 5.3 dargestellt. Sinnvolle Erweiterungen wären die automatische Ermittlung von symbolischen Konstanten, der Integration von eingebauten Prädikaten, die automatische Ermittlung von numerischen Beschränkungen und eine komfortablere Möglichkeit zur Definition des Hypothesenraums. Es wäre sinnvoll, wenn sich zukünftige Arbeiten mit einigen dieser Verbesserungen beschäftigen würden.

Einer der Verbesserungsvorschläge wurde im Rahmen dieser Diplomarbeit umgesetzt. Es handelt sich dabei um die automatische Bestimmung von numerischen Beschränkungen. Diese Komponente soll an eine Formel, die numerische Variablen enthält, Beschränkungen in Form von Intervallen so anfügen, daß Support und Confidence groß genug werden. Dazu wurde ein heuristisches Verfahren entwickelt, das ohne Vorverarbeitung auskommt und möglichst effizient arbeiten soll. Das Problem der numerischen Komponente ist, daß sie nicht vollständig ist. Das heißt, daß gültige Formeln eventuell „übersehen“ werden können.

Zur numerischen Komponente des Systems ist zu sagen, daß sie bei getesteten kleinen Beispielanwendungen bessere Ergebnisse liefert, als ich sie von Hand ermitteln konnte. Eine große Schwierigkeit ist, daß viel Aufwand getrieben wird, um eine Klausel mit Beschränkungen zu versehen. Der größte Aufwand ist dann notwendig, wenn keine

Beschränkung gefunden werden kann. Dann müssen sehr viele Möglichkeiten betrachtet werden, um am Schluß festzustellen, daß es keine Beschränkung gibt, die das Akzeptanzkriterium erfüllt.

Ein ausführlicher Versuch mit der numerischen Komponente konnte nicht durchgeführt werden, weil die Anbindung an den bestehenden Algorithmus fehlt. Diese wurde nicht implementiert, weil die Benutzung der relationalen Datenbank im Lauf der Arbeit in Frage gestellt wurde. Die numerische Komponente arbeitet auf einer eigenen Datenstruktur im Hauptspeicher. Die Schnittstelle zum restlichen System geht momentan über eine Textdatei. Es wurde erwogen, den ganzen Algorithmus in einer zukünftigen Version auf Textdateien operieren zu lassen.

Grundsätzlich ist anzuzweifeln, ob die Definition des Hypothesenraums durch Angabe von Rumpfliteralen sinnvoll ist. Hier liegt meiner Meinung ein großes Problem bei der Anwendung des Verfahrens. Der Benutzer muß komplexe Eingaben erarbeiten, was oft nur möglich ist, wenn man die Hypothesenklauseln, die der Algorithmus finden soll schon kennt. Das größte Problem dabei stellt die Spezifikation von Verbindungen von Literalen über Variablen dar. Das Problem der Definition des Hypothesenraums scheint jedoch grundsätzlich im ILP vorhanden sein. Meiner Meinung muß der Benutzer zur Definition von Metaprädikaten in RDT/DB oder DLAB-Formeln in CLAUDIEN auch schon viel über die zu erwartenden Hypothesenklauseln wissen.

Eine andere wichtige Beobachtung ist, daß die hier verwendete INGRES-Datenbank auch schon mit kleinen Relationen an ihre Grenzen stößt. Es war beispielsweise nicht möglich, einen 8-fach Join durchzuführen. Kleinere Joins wurden zwar korrekt ausgeführt, hatten aber recht lange Laufzeiten. Wichtig für (wahrscheinlich jede Data-Mining-Anwendung) wäre ein Anfrageinterpreter mit einer sehr guten Anfrageoptimierung, die eventuell Joins reduziert oder effizient ausführt¹.

In Bezug auf Effizienz gab es während der Diplomarbeit noch eine weitere Beobachtung: Bei großen Klauselmengen wird der Aufbau der Testhierarchie mit Hilfe der Subsumption so langsam, daß es besser ist, wenn man sie abschaltet. Dies ist ein grundsätzliches Problem des Algorithmus, eine schnellere Implementierung für die Erstellung der Testhierarchie ist kaum möglich.

Zusammenfassend kann festgestellt werden, daß der Algorithmus in seiner derzeitigen Version noch im Anfangsstadium steht. Es gibt jedoch viele Ansatzpunkte, an denen weitere Arbeit geleistet werden kann.

¹ Durch Verwendung von Hash-Joins oder Merge-Joins unter Einbeziehung von Schlüsselattributen

Glossar

Akzeptanzkriterium	Berechnungsvorschrift, die entscheidet, ob eine Formel als Hypothese anerkannt wird. Hier ist das Akzeptanzkriterium, daß Support und Confidence über den vom Benutzer spezifizierten Werten liegen
Argument eines Prädikats	Variable oder Konstante, von der der Wert eines Prädikats abhängt X und Y sind Argumente des Prädikats p im Literal $p(X, Y)$
Bereichsbeschränktheit	Eine Klausel, deren Kopfvariablen alle auch im Rumpf vorkommen heißt Bereichsbeschränkt
Beschränkung	Beschränkungen sind Aussagen der Form $x > 3$, $x \in [1..4]$, usw. Beschränkungen werden von der numerischen Komponente automatisch an Klauseln angehängt, die ohne die Beschränkung zu wenig Confidence hätten.
Bias	Bedingungen, die den Hypothesenraum einschränken. Ein deklarativer Bias ist ein Bias, der durch Angabe des Hypothesenraums durch den Benutzer spezifiziert wird.
Confidence	Teil des Akzeptanzkriteriums. Drückt aus, wie stark eine Implikation ist.
Data-Mining	Teil des KDD-Prozesses. Data-Mining wird unterschieden in Clustering, Regression, Klassifikation und Association Rules.
Hornklausel	Prädikatenlogische Formel der Form $H \leftarrow B_1, \dots, B_n$, wobei H der Kopf der Klausel und B_1, \dots, B_n positive Rumpfliterale sind.
Hypothesenraum	Menge aller möglichen Hypothesenklauseln
ILP	Inductive Logic Programming
KDD	Knowledge Discovery in Databases. Prozeß zur Ableitung von Wissen aus Daten.
Kopf einer Klausel	Die Klausel $K \leftarrow R$ hat den Kopf K
Rumpf	Die Klausel $K \leftarrow R$ hat den Rumpf R
Subsumption	Syntaktisch ermittelbare Beziehung zwischen zwei Hornklauseln, die ausdrückt, welche Formel allgemeiner ist.
Support	Teil des Akzeptanzkriteriums. Der Support gibt an, auf wieviele Tupel der Datenbank die Formel zutrifft.
Test	Bestimmung der Meßwerte Support und Confidence und Prüfung, ob sie groß genug sind.
Verbundene Literale	Literale sind verbunden, wenn sie entweder gemeinsame Variablen besitzen oder wenn es ein Literal gibt, das die beiden Literale verbindet.
Where-Klausel	Teil einer SQL-Anfrage

Literatur

- [1] Irene Weber: Approximate nonmonotonic ILP - a parametrized approach to clause discovery in databases. Universität Stuttgart, unveröffentlicht.
- [2] Irene Weber: Discovery of first order regularities in a relational database using offline candidate generation, unveröffentlicht.
- [3] Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo: Improved methods for finding association rules. University of Helsinki, Department of Computer Science, Technical Report, Publication No. C-1993-65
- [4] Guido Lindner: Logikbasiertes Lernen in relationalen Datenbanken, Technical Report, LS-8 Report 12, Universität Dortmund, Lehrstuhl Künstliche Intelligenz.
- [5] J.R. Quinlan: Foil: A Midterm Report, Basser Department of Computer Science, University of Sydney, Machine Learning: ECML-93, European Conference on Machine Learning 1993, Publisher: Springer.
- [6] L. De Raedt, L. Dehaspe: Clausal Discovery, Department of Computer Science, Katholieke Universiteit Leuven, Machine Learning 1996.
- [7] H. Bloekel, L. De Raedt: Relational Knowledge Discovery in Databases, Katholieke Universiteit Leuven, Technical Report, November 1996
- [8] P. Brockhausen, K. Morik: Direct Access of an ILP Algorithm to a Database Management System, Univ. Dortmund, Computer Science Department, ML96WS Pages 95-110, 1996
- [9] L. Dehaspe, L. De Raedt: DLAB, A declarative language bias for concept learning and knowledge discovery engines, Katholieke Universiteit Leuven, Technical Report CW214, oct. 1995
- [10] L. De Raedt, M. Bruynooghe: A Theory of Clausal Discovery, Department of Computing Science, Katholieke Universiteit Leuven, Technical Report.
- [FP] Usama M. Fayad, Gregory Piatetsky-Shapiro, Padhraic Smyth: From Data Mining to Knowledge Discovery: An Overview. In: Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, Ramasamy Uthurusamy: Advances in Knowledge Discovery and Data-Mining, The AAAI Press, Distributed by MIT Press, Massachusetts Institute of Technology, ISBN 0-262-56097-6
- [SD] Saso Dzeroski: Inductive Logic Programming and Knowledge Discovery in Databases. In: Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, Ramasamy Uthurusamy: Advances in Knowledge Discovery and Data-Mining, The AAAI Press, Distributed by MIT Press, Massachusetts Institute of Technology, ISBN 0-262-56097-6
- [LS] Lockemann/Schmidt: Datenbank-Handbuch, Springer-Verlag, Berlin, ISBN 0-387-10741-X
- [AR] Ashwin Shrinivasan, Rui Camacho: Numerical Reasoning in ILP, Oxford University Computing Laboratory, Foliensatz
- [MH] Fumio Mizoguchi, Hayato Ohwada: Using Inductive Logic Programming for Constraint Acquisition in Constraint Based Problem Solving, Science University of Tokyo, Noda, Chiba 278, Japan, ILP 1992

-
- [SA] Ramakrishnan Srikant, Rakesh Agrawal: Mining Quantitative Association Rules in Large Transactional Tables, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, Proc. Of the (ACM) Sigmod Conference on Management of Data 1996.
- [BB] Petr Berka, Ivan Bruha: Various discretisation procedures of numerical attributes: Empirical Comparisons, Workshop Statistics, Machine Learning and Knowledge Discovery in Databases 1995, Edited by Y Kodratoff, G Nakhaeizadeh, C: Taylor.
- [RC] Thomas W. Rauber, Dinu Coltuc, Adolfo S Steiger-Garcia: Multivariate Discretisation of continuous Attributes for Machine Learning

[1] Irene Weber: Approximate nonmonotonic ILP - a parametrized approach to clause discovery in databases. Universität Stuttgart.

[9] L. Dehaspe, L. De Raedt: DLAB, A declarative language bias for concept learning and knowledge discovery engines, Katholieke Universiteit Leuven, Report CW214, oct. 1995

[LS] Lockemann/Schmidt: Datenbank-Handbuch, Springer-Verlag, Berlin, ISBN 0-387-10741-X

[AR] Ashwin Srinivasan, Rui Camacho: Numerical Reasoning in ILP, Oxford University Computing Laboratory

[MH] Fumio Mizoguchi, Hayato Ohwada: Using Inductive Logic Programming for Constraint Acquisition in Constraint Based Problem Solving, Science University of Tokyo, Noda, Chiba 278, Japan

[SA] Ramakrishnan Srikant, Rakesh Agrawal: Mining Quantitative Association Rules in Large Transactional Tables, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120

[BB] Petr Berka, Ivan Bruha: Various discretisation procedures of numerical attributes: Empirical Comparisons

[BB] Petr Berka, Ivan Bruha: Various discretisation procedures of numerical attributes: Empirical Comparisons

[RC] Thomas W. Rauber, Dinu Coltuc, Adolfo S Steiger-Garcia: Multivariate Discretisation of continuous Attributes for Machine Learning