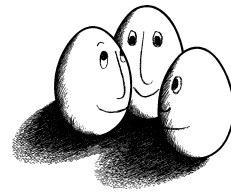


Diplomarbeit

Entdeckung häufiger Episoden und repräsentativer Episode-Regeln in Ereignis-Sequenzen

Nuhad Shaabani



Lehrstuhl für künstliche Intelligenz
Fachbereich Informatik
Universität Dortmund

Dortmund, 10.01.2007

Betreuer:
Prof. Dr. Katharina Morik
Dipl.-Inform. Timm Euler

Danksagung

Mein Dank gilt zuerst Prof. Dr. Katharina Morik, die mir die Aufgaben dieser Diplomarbeit gestellt hat und unter deren Betreuung die Arbeit stattgefunden hat.

Timm Euler danke ich sehr für seine unbürokratische und freundliche Betreuung und für seine Zahlreiche Tipps, die zur besseren Gestaltung der Diplomarbeit beigetragen haben.

Als ausländischer Student möchte ich der Universität Dortmund danken, die mich so ausgebildet hat, dass ich diese wissenschaftliche Arbeit leisten konnte.

Inhaltsverzeichnis

1	Einleitung	1
2	Entdeckung häufiger Mengen in Datenbanken	7
2.1	Problembeschreibung	7
2.2	Algorithmen	9
2.2.1	Apriori-Algorithmus	9
2.2.2	Eclat-Algorithmus	11
2.2.3	FP-growth -Algorithmus	13
3	Entdeckung häufiger Muster in zeitlichen Daten	23
3.1	Häufige Episoden in Ereignis-Sequenzen	24
3.1.1	Problembeschreibung	24
3.1.2	WINEPI-Algorithmus	29
3.2	Häufige Verkauf-Muster in Verkauf-Sequenzen	35
3.3	Häufige Intervall-Muster in Intervall-Sequenzen	36
3.3.1	Häufige Intervall-Muster nach [13]	37
3.3.2	Häufige Intervall-Muster nach [14]	38
4	Neuer Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen	43
4.1	Überblick	44
4.2	Datenstrukturen des Algorithmus	45
4.2.1	Datenstrukturen zur Repräsentation der Menge $WIN(\mathcal{S}, win)$	46
4.2.2	Datenstrukturen zur Repräsentation einer Episode	49
4.2.3	Datenstrukturen zur Repräsentation von $\mathcal{F}(WT, s)$ und $\mathcal{F}(\mathcal{S}, win, s)$ 50	
4.3	Formulierung des Algorithmus	55
4.4	Hauptschritte des Algorithmus	65
5	Kondensierte Repräsentationen der häufigen Mengen und der Assoziationsregeln	67
5.1	Kondensierte Repräsentationen der häufigen Mengen	69
5.1.1	Maximale häufige Mengen	69
5.1.2	Minimale nicht-häufige Mengen	69
5.1.3	Abgeschlossene Mengen	71
5.1.4	Freie Mengen	74

5.1.5	Disjunktion-freie Mengen	75
5.2	Assoziationsregeln	78
5.2.1	Problembeschreibung	78
5.2.2	Algorithmus zur Generierung der Assoziationsregeln	79
5.3	Kondensierte Repräsentationen der Assoziationsregeln	80
5.3.1	Inferenz-Methoden für Assoziationsregeln	81
5.3.2	Klassen der kondensierten Repräsentation der Assoziationsregeln	82
5.3.3	Repräsentative Assoziationsregeln	83
5.3.4	Minimale nicht redundante Assoziationsregeln	85
5.3.5	Die kondensierte Repräsentation der Assoziationsregeln nach [47]	88
6	Repräsentative Episode-Regeln	91
6.1	Episode-Regeln	91
6.2	Repräsentative Episode-Regeln	92
6.3	Algorithmus zur Berechnung repräsentativer Episode-Regeln	99
7	Implementierung und Experimente	103
7.1	FP-growth -Algorithmus	103
7.2	Entdeckung häufiger Episoden in Ereignis-Sequenzen	108
7.3	Repräsentative Episode-Regeln	114
7.4	Das YALE-Plugin „PatternDiscovery“	119
8	Zusammenfassung und Ausblick	121

1 Einleitung

Durch die Automatisierung und Digitalisierung der Vorgänge des alltäglichen, wirtschaftlichen und wissenschaftlichen Lebens werden immense Mengen von komplexen Daten gesammelt und gespeichert: Firmen speichern Daten über ihre Kunden, um ihre Profite zu maximieren, Wissenschaftler sammeln durch Beobachtungen und Messungen viele Daten über eine Erscheinung der Natur, um diese besser verstehen zu können und Regierungen vieler Länder verwalten große Mengen von Daten verschiedener Art, um die Sicherheit ihrer Bürger gewährleisten, was in Bezug auf den globalisierten Terrorismus als sehr ernsthaftes Problem erkannt wurde.

Um in den Daten verborgenes implizites Wissen um bereichsspezifische Zusammenhänge zu entdecken, müssen sie analysiert werden. Aber eine manuelle Analyse ist wegen des großen Umfangs der Daten so gut wie unmöglich. Also ist die Datenanalyse durch die Entwicklung geeigneter Verfahren zu automatisieren.

Die Verfahren, die sich mit der Automatisierung der Analyse von Daten beschäftigen und aus der künstlichen Intelligenz, der Theorie der Datenbanken und der Statistik stammen, sind mit den Begriffen Data Mining und Wissensentdeckung in Datenbanken (Knowledge Discovery in Databases, KDD) bezeichnet. Während sich der Begriff Data Mining auf den eigentlichen Vorgang der Datenanalyse bezieht, ist mit Wissensentdeckung im Allgemeinen der gesamte Prozess der automatisierten Analyse der Daten (KDD-Prozess) gemeint. Dieser Prozess umfasst außer der Analyse auch noch die Vorbereitung und Bereinigung der Daten und die Interpretation und Visualisierung der Resultate ([48]).

Unter Data Mining versteht man nach [48] den Schritt des KDD-Prozesses, in dem ein bestimmter Algorithmus auf die vorbereitete und bereinigte Datenmenge angewendet wird, um bei akzeptablem Ressourcen-Verbrauch interessante Muster aus ihr zu extrahieren. Mit anderen Worten ist das Ziel des Data Mining, interessante Muster, Strukturen, Abhängigkeiten usw. in Daten durch die Entwicklung geeigneter Algorithmen zu entdecken.

Der Begriff Muster ist kontextabhängig. Er hängt im Allgemeinen von dem Bereich und der Struktur der Daten und von der Zielsetzung der Analyse ab und beschreibt lokale Zusammenhänge in den Daten. Ein Muster in einer Datenmenge \mathcal{D} ist abstrakt nach [48] wie folgt zu verstehen: Ein Muster P ist ein Ausdruck in einer Sprache \mathcal{L} , der Fakten in einer Teilmenge \mathcal{D}_P von \mathcal{D} beschreibt und eine Zusammenfassung aller von ihm beschriebenen Fakten in \mathcal{D}_P darstellt. Im Kontext des maschinellen Lernens entspricht die Sprache \mathcal{L} dem Hypothesenraum \mathcal{H} , während ein Muster einer Hypothese

h in diesem Raum entspricht (s. [3], [49]). Eine intuitive Vorstellung über den Begriff Muster in einer Datenmenge sollten die folgenden Beispiele geben.

Beispiel 1.1 *Die Aussage:*

„Verheiratete Männer mit Pkws der Luxusklasse stellen nur drei Prozent der Kunden dar, erzeugen aber fünfzehn Prozent der Lebensversicherungsabschlusssumme“

repräsentiert ein interessantes Muster in der Kundendatenbank eines Versicherungsunternehmens.

Beispiel 1.2 *Die Regel:*

„80% der Kunden, die Bier und Sekt kaufen, kaufen auch Kartoffelchips“

stellt ein Muster in den Verkaufsdaten eines Supermarktes dar.

Beispiel 1.3 *Für eine Online-Buchhandlung repräsentiert die Aussage:*

„90% der Kunden, die das Buch B bestellt hatten, bestellten danach die Bücher B_1 und B_2 “

ein interessantes Muster in der Log-Datei ihres Servers.

Der wesentliche Unterschied zwischen den Mustern der letzten zwei Beispielen besteht darin, dass das Muster des Beispiels (1.3) eine zeitliche Struktur enthält, während die Zeit in dem Muster des Beispiels (1.2) überhaupt keine Rolle spielt. Das Beispiel (1.2) ist ein typisches Beispiel für das Problem der Entdeckung häufiger Mengen in Datenbanken, während das Beispiel (1.3) ein Beispiel für das Problem der Entdeckung häufiger Teilsequenzen bzw. Episoden in Sequenzen darstellt. Dabei ist eine Sequenz eine Datenmenge, zwischen deren Elementen zeitliche Relationen existieren.

Jeder Einkauf, den ein Kunde bei einem Besuch eines Supermarktes vorgenommen hat, wird als Transaktion bezeichnet. Die Menge aller von verschiedenen Kunden bei einem bestimmten Supermarkt durchgeführten Transaktionen werden in einer Datenbank namens Transaktionendatenbank gespeichert. Das Finden aller Mengen von Waren in einer Transaktionendatenbank, die häufig zusammen gekauft wurden, ist eine Instanz des Problems der Entdeckung häufiger Mengen in Datenbanken. Die Verwaltung eines Supermarktes kann durch die Ausnutzung des Wissens über die häufig zusammengekauften Waren Marketingstrategien gezielter entwickeln oder die Waren effizienter anordnen oder den Katalog des Supermarktes neu bzw. besser entwerfen.

Betrachtet man eine Datenbank als eine Menge von Tupeln, wobei jede Tupel wiederum als Menge von Objekten zu betrachten ist, dann ist also eine häufige Menge in dieser Datenbank diejenige Menge, deren Objekte häufig in den Tupeln zusammen vorkommen.

Eine Art von Daten, die zeitliche Strukturen enthalten, sind Sequenzen von Ereignissen. Beispiele für solche Daten sind Alarmmeldungen in Telekommunikationsnetzen, Aktionen der User beim Besuchen von Internetseiten eines Unternehmens, Bestellungen der Kunden bei einer Online-Buchhandlung usw. Dabei stellen Vorgänge wie „Das Buch B wurde im Monat M bestellt“, „Der Alarm A wurde gestern registriert“ und „Die Internetseite S wurde im Zeitpunkt t besucht“ Ereignisse dar.

Eine Ereignis-Sequenz ist also eine zeitliche Folge von Ereignissen, wobei ein Ereignis ein Objekt ist, dem ein Zeitpunkt zugeordnet ist, und einen bestimmten Vorgang in einem bestimmten Zusammenhang repräsentiert. Das Objekt selbst, aus dem ein Ereignis besteht, wird als Ereignis-Typ bezeichnet.

Bietet eine Online-Buchhandlung z.B. die Bücher $\{B_1, B_2, \dots, B_m\}$, dann kann man aus ihrer Kundendatenbank die folgende Ereignis-Sequenz extrahieren:

$[(B_{i,1}, t_1), (B_{i,2}, t_2), \dots, (B_{i,n}, t_n)], T_s, T_e$, wobei $B_{i,j}$ mit $1 \leq i \leq m$ und $1 \leq j \leq n$ die im Zeitraum $T_e - T_s$ Online verkaufte Bücher sind, während t_j mit $1 \leq j \leq n$ und $T_s \leq t_j < T_e$ die Zeitpunkte sind, in denen die Bücher verkauft wurden.

Ein Basisproblem in der Analyse einer Ereignis-Sequenz ist nach [9], häufige Episoden in der Ereignis-Sequenz zu finden. Eine Episode ist eine partiell geordnete Menge von Ereignis-Typen. Eine Episode kommt häufig in einer Ereignis-Sequenz vor, wenn ihre Ereignis-Typen als Ereignisse häufig in der Ereignis-Sequenz und mit der selben partiellen Ordnung vorkommen. Mit einfachen Worten ist eine häufige Episode eine Kollektion von Ereignis-Typen, die als Ereignisse häufig in der Ereignis-Sequenz zusammen vorkommen. Mithilfe der häufigen Episoden kann man z.B. im Bereich der Telekommunikationsnetzen Relationen zwischen den Alarms finden. Diese Relationen können zum besseren Verständnis der Probleme beitragen, die die Alarms verursachen. Sie können auch ausgenutzt werden, um redundante Alarms zu erkennen, oder um den Ausfall des Netzes hervorzusagen.

Die meisten Algorithmen zur Entdeckung häufiger Mengen in Datenbanken folgen der Methode der Kandidatenmengen-Generierung. Eine Kandidatenmenge hat die Eigenschaft, dass jede echte Teilmenge von ihr eine häufige Menge ist, während für sie die Datenbank komplett durchlaufen werden muss, um festzustellen, ob sie selbst eine häufige Menge ist. Obwohl diese Methode einfach zu verstehen ist, hat sie aber zwei Nachteile: Erstens steigt die Anzahl der Kandidatenmengen exponentiell in der Länge der häufigen Mengen und zweitens steigt die Anzahl der Durchläufe der Datenbank linear in der Länge der häufigen Mengen. Also beansprucht sie zu viel Rechenzeit und Speicherplatz. Der FP-growth-Algorithmus, einer der bekanntesten Algorithmen zur Entdeckung häufiger Mengen in Datenbanken, konnte durch kompakte Darstellung der Datenbank und durch effiziente Ausnutzung der Eigenschaften dieser Darstellung auf das Prinzip der Kandidatenmengen-Generierung verzichten. Der Preis dafür ist aber, dass seine Implementierung schwieriger als die Implementierung der Algorithmen ist, die das Prinzip der Kandidatenmengen-Generierung anwenden. Eines der Ziele dieser Diplomarbeit ist das Studieren und die Implementierung des FP-growth-Algorithmus. Dementsprechend wird das Kapitel 2 das Problem der Entdeckung häufiger Mengen in Datenbanken formal vorstellen und neben dem FP-growth-Algorithmus andere Algorithmen zur seiner Lösung

studieren und miteinander vergleichen.

Der WINEPI-Algorithmus ist ein Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen. Er folgt in seiner Arbeitsweise dem Prinzip der Kandidaten-Generierung. Die Anwendung des Prinzip der Kandidaten-Generierung bei der Entdeckung der häufigen Episoden in Ereignis-Sequenzen ist dadurch begründet, dass jede Subepisode einer häufigen Episode häufige Episode sein muss. Also hat eine Kandidatenepisode analog zu einer Kandidatenmenge die Eigenschaft, dass jede ihrer Subepisoden häufig ist. Im Kapitel 3 wird das Problem der Entdeckung häufiger Episoden mit dem WINEPI-Algorithmus ausführlich vorgestellt. Außerdem werden in diesem Kapitel andere Arten von Daten, die zeitliche Strukturen enthalten, studiert und miteinander verglichen.

Analog zum FP-growth-Algorithmus stellt diese Diplomarbeit die Frage, ob die Entdeckung häufiger Episoden in Ereignis-Sequenzen ohne die Anwendung des Prinzip der Kandidaten-Generierung möglich ist. Außerdem stellt sie die Frage, ob die Lösung des Problems der Entdeckung häufiger Mengen ein Teil der Lösung des Problems der Entdeckung häufiger Episoden darstellt. In der Tat wird im Kapitel 4 ein neuer Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen entwickelt, der positiv diese beiden gestellten Fragen beantwortet.

Eine Anwendung der in einer Ereignis-Sequenz entdeckten häufigen Episoden sind die Episode-Regeln. Diese beschreiben gewisse Zusammenhänge und Regelmäßigkeiten zwischen den Ereignissen der Sequenz und können zur Vorhersage des Verhalten der Sequenz genutzt werden. Sie repräsentieren beispielsweise im Bereich der Telekommunikationsnetzen Zusammenhänge zwischen Alarms und Ausfällen des Netzes wie folgt:

”Wird der Alarm A_2 nach dem Alarm A_1 gemeldet, dann wird das Netz sehr wahrscheinlich ausfallen.”

Die Episode-Regeln sind also als eine Art von interessanten Mustern in Ereignis-Sequenzen zu verstehen. Die Anzahl der Episode-Regeln ist aber in vielen Fällen zu groß, so dass das Verstehen und die Analyse des Verhaltens der Sequenz durch sie sehr schwierig wird. Deshalb stellt sich die Frage, ob die Anzahl der Episode-Regeln durch die Entwicklung einer kondensierten Repräsentation reduziert werden kann. Um eine Antwort auf diese Frage zu finden, wird im Kapitel 5 das Konzept der kondensierten Repräsentation der interessanten Muster in Datenbanken formal vorgestellt und seine Techniken im Bereich der häufigen Mengen und der Assoziationsregeln werden dort ausführlich studiert. Danach wird im Kapitel 6 ein neues Konzept zur kondensierten Repräsentation der Episode-Regeln, nämlich das Konzept „repräsentative Episode-Regeln“, entwickelt. Außerdem wird in diesem Kapitel ein Algorithmus zur Generierung der repräsentativen Episode-Regeln formuliert.

Die Diplomarbeit beschäftigt sich also mit den folgenden Aufgaben:

Aufgabe 1: Die Implementierung des FP-growth-Algorithmus zur Entdeckung häufiger Mengen in Datenbanken.

Aufgabe 2: Die Entwicklung und die Implementierung eines Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen ohne Kandidaten-Generierung.

Aufgabe 3: Die Entwicklung und die Implementierung eines Konzeptes zur kondensierten Repräsentation der Episode-Regeln.

2 Entdeckung häufiger Mengen in Datenbanken

Die Entdeckung von Mengen, deren Elemente häufig in einer Datenbank zusammenkommen, ist eine elementare Aufgabe in Data Mining, da solche Mengen in vielen Data Mining Aufgaben eine entscheidende Rolle spielen. Die ursprüngliche Motivation für das Finden häufiger Mengen ist durch den Bedarf an der Analyse der Verkaufsdaten eines Supermarktes entstanden, um das Kundenverhalten beim Einkaufen zu untersuchen ([1]). Nach der formalen Beschreibung des Problems im Abschnitt (3.1.1) werden im Abschnitt (2.2) Algorithmen zur Lösung des Problems vorgestellt, deren Unterschiede in der Datenstruktur bzw. in der Darstellung der zugrundeliegenden Datenbank liegen.

2.1 Problembeschreibung

Sei eine endliche und nicht leere Menge $\mathcal{I} = \{a_1, a_2, \dots, a_n\}$ von Objekten gegeben. Eine Transaktion über \mathcal{I} ist ein Tupel $T = (id, I)$, wobei $\emptyset \neq I \subseteq \mathcal{I}$ und $id \in \mathbb{N}$ ein Identifikator ist, der die Menge I eindeutig identifiziert. Eine Transaktionsdatenbank \mathcal{D} über \mathcal{I} ist eine endliche Menge von Transaktionen über \mathcal{I} .

Definition 2.1 (Support einer Menge in \mathcal{D}) *Der Support einer Menge $X \subseteq \mathcal{I}$ in \mathcal{D} ist die Anzahl der Transaktionen in \mathcal{D} , die X enthalten. Diese Anzahl wird durch $sup(X, \mathcal{D})$ gekennzeichnet. D.h.*

$$sup(X, \mathcal{D}) = |\{(id, I) \in \mathcal{D} \mid X \subseteq I\}|$$

Der Support einer beliebigen Menge $X \subseteq \mathcal{I}$ erfüllt $0 \leq sup(X, \mathcal{D}) \leq |\mathcal{D}|$, was unmittelbar aus der obigen Definition zu schließen ist. Die Häufigkeit einer Menge $X \subseteq \mathcal{I}$ in \mathcal{D} ist die Wahrscheinlichkeit dafür, dass X in einer Transaktion $T \in \mathcal{D}$ vorkommt. Diese Wahrscheinlichkeit hat den Wert $sup(X, \mathcal{D})/|\mathcal{D}|$. Die Häufigkeit einer Menge X wird als relativer Support dieser Menge bezeichnet. Nun kann der Begriff „häufige Menge“ formuliert werden.

Definition 2.2 (Häufige Menge in \mathcal{D}) *Sei $s \in \mathbb{N}$ gegeben. Eine Menge $X \subseteq \mathcal{I}$ ist häufig in \mathcal{D} genau dann, wenn $sup(X, \mathcal{D}) > s$ ist. Die Zahl s heißt minimaler Support ¹*

Für ein gegebenes $s \in \mathbb{N}$ wird die Menge aller häufigen Mengen in \mathcal{D} durch $\mathcal{F}(\mathcal{D}, s)$ gekennzeichnet. D.h.

$$\mathcal{F}(\mathcal{D}, s) = \{X \subseteq \mathcal{I} \mid sup(X, \mathcal{D}) > s\}$$

¹In der Literatur wird der minimale Support sehr oft durch *minSup* bezeichnet.

Basierend auf den obigen Begriffen kann jetzt das Problem „Entdeckung häufiger Mengen in einer Datenbank“ vorgestellt werden.

Definition 2.3 (Entdeckung häufiger Mengen) *Seien eine Objektmenge \mathcal{I} , eine Transaktionsdatenbank \mathcal{D} über \mathcal{I} und ein $s \in \mathbb{N}$ gegeben. Die Berechnung der Menge $\mathcal{F}(\mathcal{D}, s)$ heißt Entdeckung häufiger Mengen in der Datenbank \mathcal{D} .*

Also ist das Problem der Entdeckung häufiger Mengen ein Suchproblem, wobei der Suchraum die Potenzmenge $2^{\mathcal{I}}$ der Objektmenge \mathcal{I} ist. Weil die Größe des Suchraumes exponentiell in der Kardinalität der Menge \mathcal{I} ist, ist eine triviale Suchstrategie in akzeptierbarer Zeit undurchführbar für eine relativ große Anzahl von Objekten. Mit der trivialen Suchstrategie ist das Testen jeder Menge des Suchraumes, ob sie häufig in \mathcal{D} ist, gemeint. Man kann die Suche nur auf die Mengen einschränken, die nur in den Transaktionen vorkommen. Dies ist auch eine triviale Lösung, weil eine Transaktion gleich der Objektmenge \mathcal{I} sein kann. Also muss man die Suche anhand des Kriteriums charakterisieren, das die zu entdeckenden Mengen erfüllen müssen, um sie effizienter durchzuführen.

Lemma 2.1 *Sei \mathcal{D} eine Transaktionsdatenbank über der Menge \mathcal{I} gegeben. Es gilt folgendes:*

$$\forall X, Y \in 2^{\mathcal{I}} : X \subseteq Y \Rightarrow \text{sup}(X, \mathcal{D}) \geq \text{sup}(Y, \mathcal{D})$$

Beweis: Definieren wir die beiden folgenden Mengen:

$$\mathcal{T}_X = \{T \in \mathcal{D} \mid X \subseteq T\}, \mathcal{T}_Y = \{T \in \mathcal{D} \mid Y \subseteq T\}$$

dann haben wir:

$\forall T \in \mathcal{T}_Y : X \subseteq T$ weil $Y \subseteq T$ und $X \subseteq Y$ ist. Dies bedeutet $\mathcal{T}_Y \subseteq \mathcal{T}_X$. Daraus schließen wir: $\text{sup}(Y, \mathcal{D}) = |\mathcal{T}_Y| \leq |\mathcal{T}_X| = \text{sup}(X, \mathcal{D})$. \square

Das folgende Lemma zeigt Eigenschaften, die eine häufige bzw. nicht häufige Menge hat. Der Beweis dieses Lemmas ist direkt aus dem obigen Lemma zu schließen.

Lemma 2.2 *Für eine gegebene Transaktionsdatenbank \mathcal{D} und ein $s \in \mathbb{N}$ gilt folgendes:*

1. *Ist $X \subseteq \mathcal{I}$ eine häufige Menge in \mathcal{D} , dann ist jede Teilmenge von X eine häufige Menge in \mathcal{D} .*
2. *Ist $X \subseteq \mathcal{I}$ keine häufige Menge in \mathcal{D} , dann ist jede Übermenge von X keine häufige Menge in \mathcal{D} .*

Durch die Ausnutzung der in Lemma (2.2) formulierten Eigenschaften können Suchstrategien bzw. Algorithmen zur Berechnung der Menge $\mathcal{F}(\mathcal{D}, s)$ entworfen werden. Der Apriori-Algorithmus ([2]), der im nächsten Abschnitt vorgestellt wird, ist der erste Algorithmus, der das Lemma (2.2) angewendet hat. Ein anderer Punkt, der bei der Entwicklung einer effizienten Suchstrategie eine wesentliche Rolle spielt, ist die Darstellung der Menge \mathcal{D} . In ([1]) ist die Datenbank \mathcal{D} als binäre Matrix dargestellt. Bei dieser Darstellung verwendet man eine $|\mathcal{D}| \times |\mathcal{I}|$ -Matrix M , die nach der Durchnummerierung der Objekte in \mathcal{I} wie folgt definiert ist: Für jedes $a_j \in \mathcal{I}$ und für jede Transaktion $(id, I) \in \mathcal{D}$

ist $M(id, j) = 1$ genau dann, wenn $a_j \in I$ ist, und $M(id, j) = 0$ genau dann, wenn $a_j \notin I$ ist, wobei die Identifikatoren id so angesetzt sind, dass $id \in \{1, \dots, |\mathcal{D}|\}$ gilt. Das Beispiel (2.1) demonstriert die binäre Darstellung einer Transaktionendatenbank.

Beispiel 2.1 Für eine Objektmenge $\mathcal{I} = \{a, b, c, d, e\}$ sieht z.B. eine Transaktionendatenbank über \mathcal{I} so aus:

$\mathcal{D} = \{(1, \{a, b, c, e\}), (2, \{b, c, e\}), (3, \{a, c, d, e\}), (4, \{b, e\}), (5, \{a, b, c\})\}$. Die binäre Darstellung dieser Transaktionendatenbank ist in der Tabelle (2.1) gezeigt.

id	a	b	c	d	e
1	1	1	1	0	1
2	0	1	1	0	1
3	1	0	1	1	1
4	0	1	0	0	1
5	1	1	1	0	0

Tabelle 2.1: Die binäre Darstellung einer Transaktionendatenbank \mathcal{D} über der Menge $\mathcal{I} = \{a, b, c, d, e\}$.

Der nächste Abschnitt zeigt durch Vorstellung mehrerer Algorithmen zur Berechnung der Menge $\mathcal{F}(\mathcal{D}, s)$, welche entscheidende Rolle die Repräsentation der Datenbank bei der Entwicklung einer Suchstrategie spielt.

2.2 Algorithmen

2.2.1 Apriori-Algorithmus

Beim Apriori-Algorithmus ([2], [3], [4]) ist die Menge \mathcal{D} als Array der Größe $|\mathcal{D}|$ repräsentiert, wobei für jede Transaktion $T = (id, I)$ über \mathcal{I} gilt: $\mathcal{D}[id] = I$. Diese Darstellung heißt horizontale Darstellung der Transaktionendatenbank. Die Tabelle (2.2) zeigt beispielweise diese Repräsentation für die im Beispiel (2.1) gegebene Transaktionendatenbank. Für die Formulierung des Apriori-Algorithmus werden folgende Notationen vereinbart: \mathcal{F}_l bezeichnet die Menge aller häufigen Mengen der Länge l , wobei $1 \leq l \leq |\mathcal{I}|$ gilt. \mathcal{C}_l bezeichnet die Menge aller Mengen der Länge l , für die gilt, dass jede echte Teilmenge von jeder beliebigen Menge $C \in \mathcal{C}_l$ häufig ist. Außerdem werden die Elemente in jeder Menge $X \subseteq \mathcal{I}$ lexikographisch geordnet. Der Apriori-Algorithmus besteht wesentlich aus den folgenden drei Schritten, wobei der zweite Schritt und der dritte Schritt für alle $l \in \{1, \dots, |\mathcal{I}|\}$ zu wiederholen sind, solange die Menge \mathcal{C}_l keine leere Menge ist:

1. Schritt (\mathcal{C}_1 -Initialisierung): In diesem Schritt wird die Menge \mathcal{C}_1 wie folgt initialisiert:

$$\mathcal{C}_1 = \{\{a\} \mid a \in \mathcal{I}\}$$

id	$\mathcal{D}[id]$
1	$\{a, b, c, e\}$
2	$\{b, c, e\}$
3	$\{a, c, d, e\}$
4	$\{b, e\}$
5	$\{a, b, c\}$

Tabelle 2.2: Die horizontale Darstellung einer Transaktionsdatenbank über der Menge $\mathcal{I} = \{a, b, c, d, e\}$

- 2. Schritt (\mathcal{F}_l -Berechnung):** Die Menge \mathcal{F}_l wird aus der Menge \mathcal{C}_l erzeugt. Dazu wird für jede Transaktion $T = (id, I) \in \mathcal{D}$ und für jede Menge $X \in \mathcal{C}_l$ geprüft, ob $X \subseteq I$ gilt, um den Support von X zu berechnen. Danach werden alle häufig gefundenen Mengen in die Menge \mathcal{F}_l aufgenommen. D.h. $\mathcal{F}_l = \{X \in \mathcal{C}_l \mid \text{sup}(X, \mathcal{D}) > s\}$
- 3. Schritt (\mathcal{C}_{l+1} -Berechnung):** Die Menge \mathcal{C}_{l+1} wird aus der Menge \mathcal{F}_l erzeugt. Dazu wird für jede $X = \{a_1, \dots, a_{l-1}, a_l\}$, $Y = \{a_1, \dots, a_{l-1}, a'_l\}$ aus \mathcal{F}_l die Menge $Z = \{a_1, \dots, a_{l-1}, a_l, a'_l\}$ erzeugt, wobei a_l lexikographisch vor a'_l liegt. Danach wird die Menge Z in \mathcal{C}_{l+1} aufgenommen, falls jede echte Teilmenge der Länge l aus Z in \mathcal{F}_l enthalten ist. D.h. $\mathcal{C}_{l+1} = \{Z \subseteq \mathcal{I} \mid |Z| = l + 1 \wedge (\forall U \subset Z : |U| = l \Rightarrow U \in \mathcal{F}_l)\}$

Die Mengen von \mathcal{C}_l heißen Kandidaten der Länge l , weil aus denen die Häufigen der Länge l ausgewählt werden. Die Tatsache, dass jede Teilmenge einer häufigen Menge auch häufig sein muss (Lemma (2.2)), wird im dritten Schritt ausgenutzt, um die Anzahl der Kandidaten zu reduzieren. Mit anderen Worten braucht der Algorithmus den zweiten Schritt für eine Menge X nicht auszuführen, die eine nicht häufige Teilmenge hat. Alle Mengen, die im zweiten Schritt als nicht häufig klassifiziert sind, werden nicht mehr im dritten Schritt in Betracht gezogen, weil aus denen keine Kandidaten mehr generiert werden können.

Beispiel 2.2 Sei $s = 2$ und \mathcal{D} die Transaktionsdatenbank, die in der Tabelle (2.2) dargestellt ist. Der erste Schritt des Algorithmus liefert die Kandidaten der Länge 1 $\mathcal{C}_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. Dann wird im zweiten Schritt die Menge $\mathcal{F}_1 = \{\{a\}, \{b\}, \{c\}, \{e\}\}$ erzeugt. Die Mengen \mathcal{C}_2 und \mathcal{F}_2 sind in der Tabelle (2.3) aufgestellt. Die Anwendung des dritten Schrittes auf \mathcal{F}_2 liefert die Menge $\mathcal{C}_3 = \{\{b, c, e\}\}$. Da $\text{sup}(\{b, c, e\}) = 2 \leq s$ ist, liefert der zweite Schritt eine leere Menge, d.h. $\mathcal{F}_3 = \emptyset$. Daraus können keine Kandidaten mehr generiert werden. Der Algorithmus terminiert deshalb und liefert die Menge $\mathcal{F}(\mathcal{D}, 3) = \mathcal{F}_1 \cup \mathcal{F}_2$.

Wenn man sich den Suchraum als Graphen $G = (2^{\mathcal{I}}, \{(X, Y) \in 2^{\mathcal{I}} \times 2^{\mathcal{I}} \mid X \subseteq Y\})$ vorstellt, dann stellt man fest, dass der Apriori-Algorithmus eine Breitensuche in diesem Graphen ausführt, weil er zuerst alle Knoten der Ebene l und erst danach alle Knoten der Ebene $l + 1$ besucht. Dabei enthält die Ebene l alle Mengen der Länge l .

$X \in \mathcal{C}_2$	$sup(X, \mathcal{D})$	$X \in \mathcal{F}_2$
$\{a, b\}$	2	\times
$\{a, c\}$	3	$\{a, c\}$
$\{a, e\}$	2	\times
$\{b, c\}$	3	$\{b, c\}$
$\{b, e\}$	3	$\{b, e\}$
$\{c, e\}$	3	$\{c, e\}$

Tabelle 2.3: Die Kandidaten der Länge 2 und die daraus erzeugten häufigen Mengen der Länge 2, die Apriori aus der in der Tabelle (2.2) dargestellten Datenbank und für den minimalen Support $s = 2$ berechnet.

2.2.2 Eclat-Algorithmus

Zur Darstellung der Datenbank \mathcal{D} im Eclat-Algorithmus ([5], [4]) wird jedem Objekt $a_i \in \mathcal{I}$ eine Menge bzw. eine Liste \mathcal{T}_i zugeordnet, in der alle Identifikatoren id 's der das Objekt a_i enthaltenden Transaktionen gespeichert sind. D.h. $\forall a_i \in \mathcal{I} : \mathcal{T}_i := \{id \mid (id, I) \in \mathcal{D} \wedge a_i \in I\}$. Diese Darstellung wird als vertikale Darstellung bezeichnet. Die Tabelle (2.4) zeigt als Beispiel die vertikale Darstellung der im Beispiel (2.1) gegebene Transaktionsdatenbank. Für die Eclats Formulierung werden folgende Vereinbarungen getroffen: Die Objekte der Menge \mathcal{I} unterliegen einer lexikographischen Ordnung. Basierend darauf werden die Mengen aus $2^{\mathcal{I}}$ als Wörter interpretiert und verglichen. Die Menge \mathcal{I}^P ($P \subseteq \mathcal{I}$) bezeichnet die Menge aller häufigen Mengen, deren $|P|$ -Präfix gleich der Menge P ist. Es gilt $\mathcal{I}^\emptyset := \{\{a_i\} \mid a_i \in \mathcal{I} \wedge sup(\{a_i\}, \mathcal{D}) > s\}$. Für ein $X \in \mathcal{I}^P$ bezeichnet die Menge \mathcal{T}_X^P die Menge der Identifikatoren aller X enthaltenden Transaktionen. D.h. $\forall X \in \mathcal{I}^P : \mathcal{T}_X^P := \{id \mid (id, I) \in \mathcal{D} \wedge X \subseteq I\}$. Es gilt $\forall a_i \in \mathcal{I} : \mathcal{T}_{\{a_i\}}^\emptyset := \mathcal{T}_i$. Die Menge \mathcal{D}^P bezeichnet die Menge $\{\mathcal{T}_X^P \mid X \in \mathcal{I}^P\}$. Es gilt $\mathcal{D}^\emptyset := \mathcal{D}$.

a_i	\mathcal{T}_i
a	$\{1, 3, 5\}$
b	$\{1, 2, 4, 5\}$
c	$\{1, 2, 3, 5\}$
d	$\{3\}$
e	$\{1, 2, 3, 4\}$

Tabelle 2.4: Die vertikale Darstellung der im Beispiel (2.1) gegebene Datenbank

In **Algorithm (1)** ist eine Eclat's Formulierung angegeben. Der Algorithmus ist in dieser Form gestaltet, damit man die Gemeinsamkeiten zwischen ihm und dem im Abschnitt

(2.2.3) vorgestellten FP-growth-Algorithmus erkennen kann. Entsprechend dieser Formulierung wird der Algorithmus mit den Parametern $\mathcal{I}^\emptyset = \{\{a_i\} \mid a_i \in \mathcal{I} \wedge \text{sup}(\{a_i\}, \mathcal{D}) > s\}$, $\mathcal{D}^\emptyset = \{\mathcal{T}_{\{a_i\}}^\emptyset \mid \{a_i\} \in \mathcal{I}^\emptyset\}$, s und $\mathcal{F}(\mathcal{D}, s) = \{\{a_i\} \mid \{a_i\} \in \mathcal{I}^\emptyset\}$ aufgerufen. Also sind alle einelementigen häufigen Mengen in einem Initialisierungsschritt zu entdecken.

Algorithm 1 Eclat-Algorithmus

```

1: function ECLAT( $\mathcal{I}^P, \mathcal{D}^P, \mathcal{F}(\mathcal{D}, s), s$ )
2:   for all  $X \in \mathcal{I}^P$  do
3:      $\mathcal{I}^X \leftarrow \emptyset, \mathcal{D}^X \leftarrow \emptyset$ 
4:     for all  $Y \in \mathcal{I}^P : X < Y$  do
5:        $Z \leftarrow X \cup Y$ 
6:        $\mathcal{T}_Z^X \leftarrow \mathcal{T}_X^P \cap \mathcal{T}_Y^P$ 
7:       if  $|\mathcal{T}_Z^X| > s$  then
8:          $\mathcal{I}^X \leftarrow \mathcal{I}^X \cup \{Z\}, \mathcal{D}^X \leftarrow \mathcal{D}^X \cup \mathcal{T}_Z^X$ 
9:       end if
10:    end for
11:     $\mathcal{F}(\mathcal{D}, s) \leftarrow \mathcal{F}(\mathcal{D}, s) \cup \mathcal{I}^X$ 
12:    eclat( $\mathcal{I}^X, \mathcal{D}^X, \mathcal{F}(\mathcal{D}, s), s$ )
13:  end for
14:  return  $\mathcal{F}(\mathcal{D}, s)$ 
15: end function

```

Aus jedem Paar $X, Y \in \mathcal{I}^P$, wobei X vor Y lexikographisch liegt, wird eine neue Kandidatenmenge $Z = X \cup Y$ in der Zeile 5 erzeugt. Da Z nur in den Transaktionen vorkommt, in denen X und Y zusammen vorkommen, ist die Menge der Z enthaltenden Transaktionen der Durchschnitt von \mathcal{T}_X^P und \mathcal{T}_Y^P (Zeile 6). Da die Menge \mathcal{T}_Z^P alle Identifikatoren der Z enthaltenden Transaktionen ist, braucht man zur Feststellung der Häufigkeit von Z nur die Anzahl der Elemente dieser Menge zu ermitteln. Dadurch ist der Vorteil der vertikalen Darstellung gegenüber der horizontalen Darstellung hervorgehoben. Dies bedeutet, dass man das Durchlaufen aller Transaktionen in \mathcal{D} zur Feststellung der Häufigkeit einer Menge, was der Fall bei Apriori ist, bei Eclat nicht braucht. Nach der Berechnung des Supports von Z (Zeile 7) wird sie in \mathcal{I}^X aufgenommen (Zeile 8), falls sie häufig ist. Gleichzeitig wird die Menge \mathcal{D}^X um \mathcal{T}_Z^X erweitert. Nach der Berechnung von \mathcal{I}^X wird diese in die Menge $\mathcal{F}(\mathcal{D}, s)$ eingefügt, da alle ihre Elemente häufige Mengen sind. Die Arbeitsweise von Eclat-Algorithmus ist im Beispiel (2.3) demonstriert.

Beispiel 2.3 Sei der minimale Support $s = 1$. Es wird die im Beispiel (2.1) gegebene Datenbank mit ihrer in der Tabelle (2.4) gezeigten vertikalen Darstellung betrachtet. Die häufigen Objekte sind $\{a, b, c, e\}$. Also ist Eclat mit den Parametern:

$$\begin{aligned}
s &= 1, \\
\mathcal{I}^\emptyset &= \{\{a\}, \{b\}, \{c\}, \{e\}\}, \\
\mathcal{D}^\emptyset &= \{\{1, 3, 5\}_{\{a\}}^\emptyset, \{1, 2, 4, 5\}_{\{b\}}^\emptyset, \{1, 2, 3, 5\}_{\{c\}}^\emptyset, \{1, 2, 3, 4\}_{\{e\}}^\emptyset\} \text{ und} \\
\mathcal{F}(\mathcal{D}, 1) &= \{\{a\}, \{b\}, \{c\}, \{e\}\}
\end{aligned}$$

aufzurufen. Die Tabelle (2.5) zeigt die vom Algorithmus berechneten Mengen $X, \mathcal{I}^X, \mathcal{D}^X$.

Z.B. für $X = \{a, b\}$ und $Y = \{a, e\}$ wird in der Zeile (5) die Menge $Z = \{a, b, e\}$ und in der Zeile (6) die entsprechende Menge $\mathcal{T}_{abe}^{ab} = \mathcal{T}_{ab}^a \cap \mathcal{T}_{ae}^a = \{1, 5\} \cap \{1, 3\} = \{1\}$ berechnet. Da $|\mathcal{T}_{abe}^{ab}| \not\geq s$ ist, wird die Menge Z in die Menge \mathcal{I}^{ab} nicht eingefügt. Der Algorithmus liefert nach seiner Terminierung die Menge:

$$\mathcal{F}(\mathcal{D}, 1) = \{a, b, c, e, ab, ac, ae, abc, ace, bc, be, bce, ce\}$$

X	\mathcal{I}^X	\mathcal{D}^X
a	$\{ab, ac, ae\}^a$	$\{\{1, 3, 5\}_{ab}^a, \{1, 3, 5\}_{ac}^a, \{1, 3\}_{ae}^a\}$
ab	$\{abc\}^{ab}$	$\{\{1, 5\}_{abc}^{ab}\}$
abc	\emptyset	\emptyset
ac	$\{ace\}^{ac}$	$\{\{1, 3\}_{ace}^{ac}\}$
ace	\emptyset	\emptyset
ae	\emptyset	\emptyset
b	$\{bc, be\}^b$	$\{\{1, 2, 5\}_{bc}^b, \{1, 2, 4\}_{be}^b\}$
bc	$\{bce\}^{bc}$	$\{\{1, 2\}_{bce}^{bc}\}$
bce	\emptyset	\emptyset
be	\emptyset	\emptyset
c	$\{ce\}^c$	$\{\{1, 2, 3\}_{ce}^c\}$
ce	\emptyset	\emptyset
e	\emptyset	\emptyset

Tabelle 2.5: Die Tabelle zeigt den Verlauf des Eclat-Algorithmus für die im Beispiel (2.1) gegebene Datenbank. Dabei ist der minimale Support $s = 1$

Der Eclat-Algorithmus hat die Eigenschaft: $\forall P, Q \subseteq \mathcal{I} : P \neq Q \Rightarrow \mathcal{I}^P \cap \mathcal{I}^Q = \emptyset$. Dies bedeutet, dass die Generierung der Menge \mathcal{I}^P unabhängig von der Generierung der Menge \mathcal{I}^Q ausgeführt werden kann. Diese Tatsache kann man zur Lösung des Problems ausnutzen, bei dem die Datenbank \mathcal{D} nicht in den Hauptspeicher passt. Dabei wird für jede Menge $\{a_i\} \in \mathcal{I}^0$ nur die Datenbank $\mathcal{D}^{\{a_i\}}$ in den Hauptspeicher geladen und auf sie wird der Eclat-Algorithmus separat angewendet. Falls sich $\mathcal{D}^{\{a_i\}}$ in den Hauptspeicher wiederum nicht einpasst, wird sie als selbstständige Datenbank betrachtet und das selbe Verfahren auf sie angewendet, das auf \mathcal{D} anzuwenden ist.

2.2.3 FP-growth -Algorithmus

Ein Nachteil des Apriori-Algorithmus ist das Verfahren, das er für die Berechnung des Supports der Kandidaten anwendet. Denn für jede Kollektion \mathcal{C}_l von Kandidaten muss bei diesem Verfahren die Datenbank komplett durchgelaufen werden, was zu viel Laufzeit in Anspruch nimmt. Die Generierung, Speicherung und die Berechnung des Supports von großen Anzahl von Kandidaten beansprucht viele Ressourcen. Dies ist ein anderer Nachteil von Apriori. Zum Beispiel, ist die Anzahl der einelementigen häufigen Mengen

$\mathcal{F}_1 = 1000$, dann generiert Apriori $\binom{|\mathcal{F}_1|}{2} \approx 10^6/2 = 500000$ Kandidaten der Länge 2. Ein anderes Beispiel, für eine häufige Menge der Länge n müssen nach dem Lemma (2.2) mindestens $\binom{n}{2}$ Kandidaten der Länge 2, mindestens $\binom{n}{3}$ Kandidaten der Länge 3, \dots , mindestens $\binom{n}{l}$ Kandidaten der Länge l ($2 \leq l \leq n$), \dots , und mindestens ein Kandidat der Länge n generiert werden. Also beträgt die Anzahl der Kandidaten mindestens $2^n - n - 1$ Kandidaten, um eine häufige Menge der Länge n zu finden. Dies bedeutet, dass die Anzahl der Kandidaten exponentiell in der Länge der häufigen Mengen steigt. Der Eclat-Algorithmus folgt auch dem Prinzip der Kandidaten-Generierung (Zeile 5 in **Algorithm (1)**) jedoch nicht in der Form und dem Umfang, wie bei Apriori. Deshalb stellt sich die Frage, ob man durch eine geeignete Darstellung der Datenbank \mathcal{D} auf das Prinzip der Kandidaten-Generierung verzichten kann. Der FP-growth-Algorithmus ([6], [7]) hat diese Frage durch den Entwurf einer neuen Datenstruktur namens FP-tree positiv beantwortet. Für die Konstruktion der FP-tree-Struktur muss die Datenbank in einem als Vorbereitung zu betrachtenden Schritt nur zwei Mal durchgelaufen werden. Danach wird die Datenbank nicht mehr gebraucht, da alle notwendigen Informationen zur Entdeckung der häufigen Mengen in kompakter Form in der FP-tree-Struktur gespeichert sind. Das heißt, dass FP-growth das Problem des mehrfachen Durchlaufes der Datenbank behoben hat. Durch Ausnutzung der Eigenschaften der FP-tree-Struktur berechnet FP-growth-Algorithmus alle häufigen Mengen, ohne Kandidaten zu generieren. FP-growth-Algorithmus besteht aus drei Stufen, die in den folgenden drei Unterabschnitten ausführlich erklärt werden.

Vorbereitung der Datenbank

Zuerst wird der Support jedes Objektes $a_i \in \mathcal{I}$ berechnet. Dies verlangt einen Durchlauf der Datenbank \mathcal{D} . Im zweiten Durchlauf werden alle nicht häufigen Objekte $a_i \in \mathcal{I}$ aus der Datenbank entfernt und in jeder Transaktion alle übriggebliebenen häufigen Objekte nach deren Support absteigend sortiert. Der Grund für die Entfernung aller nicht häufigen Objekte aus der Datenbank ist der, dass keine häufige Menge $X \in \mathcal{F}(\mathcal{D}, s)$ eine nicht häufige Menge der Länge 1 enthalten kann. Sobald die FP-tree-Struktur in dem nächsten Unterabschnitt definiert wird, wird der Grund für die Sortierung der häufigen Objekte in den Transaktionen verständlich. In der Tabelle (2.6) ist die Vorbereitung einer Datenbank über der Menge $\mathcal{I} = \{a, b, c, d, e, f, g\}$ demonstriert, wobei der minimale Support s gleich 2 ist. Dabei ist die Menge I in jeder nach der Entfernung aller nicht häufigen Objekte und der absteigenden Sortierung der häufigen Objekte nach deren Support neu entstehenden Transaktion $T = (id, I)$ als eine geordnete Liste betrachtet, weil die Position jedes Objektes in der jeweiligen Transaktion jetzt eine Bedeutung hat. Um dies zu betonen, werden die geschweiften Klammern $\{ , \}$ durch die eckigen Klammern $[,]$ ersetzt. Das i -Element in der Transaktion (id, I) wird deshalb durch $I[i]$ gekennzeichnet. In der „Vorbereitung der Datenbank“-Phase wird auch eine neue aus drei Spalten bestehende Tabelle namens Header-Tabelle (HT) konstruiert. Die erste Spalte enthält alle häufigen Objekte in einer absteigenden Sortierung nach deren Supports, die entsprechend in der zweiten Spalte gespeichert sind. Jedes Objekt aus der ersten Spalte wird mit einer leeren Liste assoziiert, die in der dritten Spalte gespeichert ist. Die Header-Tabelle wird

id	$\mathcal{D}[id]$
1	$\{a, d, f\}$
2	$\{a, c, d, e\}$
3	$\{b, d\}$
4	$\{b, c, d\}$
5	$\{b, c\}$
6	$\{a, b, d\}$
7	$\{b, d, e\}$
8	$\{b, c, e, g\}$
9	$\{c, d, f\}$
10	$\{a, b, d\}$

a_i	$sup(\{a_i\}, \mathcal{D})$
d	8
b	7
c	5
a	4
e	3
f	2
g	1

id	$\mathcal{D}[id]$
1	$[d, a]$
2	$[d, c, a, e]$
3	$[d, b]$
4	$[d, b, c]$
5	$[b, c]$
6	$[d, b, a]$
7	$[d, b, e]$
8	$[b, c, e]$
9	$[d, c]$
10	$[d, b, a]$

Tabelle 2.6: Die **linke Tabelle** zeigt eine Datenbank über der Menge $\mathcal{I} = \{a, b, c, d, e, f, g\}$ in vertikaler Darstellung. In der **mittleren Tabelle** sind alle Objekte mit deren Support eingetragen. Die **rechte Tabelle** ist die Datenbank nach der Entfernung aller nicht häufigen Objekte ($s = 2$) und der absteigenden Sortierung der häufigen Objekte in jeder neu entstehenden Transaktion.

ein Bestandteil der im nächsten Unterabschnitt vorzustellenden FP-tree-Struktur sein. Um die Konstruktion der FP-tree-Struktur zu ermöglichen, wird jedem häufigen Objekt ein Wert zugeordnet, der seine Position in der Header-Tabelle angibt. Die Header-Tabelle HT kann man als Liste $HT = [(a_1, s_1, L_1), \dots, (a_i, s_i, L_i), \dots, (a_h, s_h, L_h)]$ auffassen, die folgende Eigenschaften hat:

- $\forall 1 \leq i \leq h : s_i = sup(\{a_i\}, \mathcal{D})$
- $\forall 1 \leq i \leq h : L_i$ ist die mit a_i assoziierte Liste.
- $s_1 \geq \dots \geq s_i \geq \dots \geq s_h > s$

Die Tabelle (2.7) zeigt als Beispiel die mit der im rechten Teil der Tabelle (2.6) als vorbereitet gezeigten Datenbank assoziierte Header-Tabelle.

In den nächsten Unterabschnitten wird immer angenommen, dass die Datenbank vorbereitet und mit einer Header-Tabelle assoziiert ist.

FP-tree-Struktur

Wenn man sich die vorbereitete Datenbank ansieht, die im rechten Teil der Tabelle (2.6) gezeigt ist, bemerkt man, dass das Objekt d in 80% der Transaktionen vorkommt und das Objekt b in 70% der Transaktionen enthalten ist. Um dieses redundante Speichern der Objekte zu beseitigen, wurde in ([6]) eine neue Datenstruktur namens FP-tree-Struktur entworfen. Wir werden hier aber die in ([6]) gegebene Definition dieser Datenstruktur

i	$HT[i] \cdot a_j$	$HT[i] \cdot sup$	$HT[i] \cdot list$
1	d	8	$[\]$
2	b	7	$[\]$
3	c	5	$[\]$
4	a	4	$[\]$
5	e	3	$[\]$

$a_j \in \mathcal{I}$	$a_j \cdot pos$
a	4
b	2
c	3
d	1
e	5

Tabelle 2.7: **Die linke Tabelle** ist die Header-Tabelle, die mit der im rechten Teile der Tabelle (2.6) gezeigten Datenbank assoziiert ist. **Die rechte Tabelle** gibt an, in welcher Zeile der Header-Tabelle jedes Objekt zu finden ist.

nicht übernehmen, sondern zuerst einpaar neue Begriffe einfügen und dann definieren wir basierend auf diesen neuen Begriffen die FP-tree-Struktur.

Definition 2.4 (Portion einer Datenbank) Eine Portion \mathcal{P} von der Datenbank \mathcal{D} ist die Menge aller Transaktionen in \mathcal{D} , für die gilt:

$$\forall T_1 = (id_1, I_1), T_2 = (id_2, I_2) \in \mathcal{P} : I_1[1] = I_2[1]$$

Definition 2.5 (Präfix einer Portion) Sei \mathcal{P} eine Portion von einer Datenbank \mathcal{D} . Das Präfix von \mathcal{P} ist das Objekt $I[1]$ in irgendeiner Transaktion $(id, I) \in \mathcal{P}$. Dieses Objekt wird durch $pre(\mathcal{P})$ gekennzeichnet. Sind $T_1, T_2, \dots, T_{|\mathcal{P}|}$ alle in \mathcal{P} enthaltenen Transaktionen, dann gilt gemäß der Definition (2.4): $pre(\mathcal{P}) = I_1[1] = I_2[1] = \dots = I_{|\mathcal{P}|}[1]$.

Definition 2.6 (Präfix-Baum einer Portion) Ein Präfix-Baum einer Portion \mathcal{P} von einer Datenbank \mathcal{D} ist ein Baum, der rekursiv wie folgt definiert ist:

- Wenn gilt: $\forall T = (id, I) \in \mathcal{P} : |I| = 1$, dann besteht der Präfix-Baum aus einem einzigen Knoten, in dem das Präfix $pre(\mathcal{P})$ und der Wert $|\mathcal{P}|$ gespeichert sind.
- Existiert eine Transaktion $T = (id, I) \in \mathcal{P}$ mit $|I| > 1$, dann sind in der Wurzel des Baumes das Präfix der Portion \mathcal{P} und der Wert $|\mathcal{P}|$ zu speichern und die Kinder der Wurzel sind alle Präfix-Bäume aller Portionen der Menge: $\{(id, I') \mid \exists (id, I) \in \mathcal{P} : I' = I \setminus \{I[1]\}\}$. Also ist diese die Menge derjenigen Transaktionen, die dadurch entstehen, indem man das Präfix aus jeder Transaktion in \mathcal{P} entfernt.

Jedem Knoten eines Präfix-Baumes ist das Paar $(a, count)$ zugeordnet, wobei $a \in \mathcal{I}$ das gespeicherte Objekt und $count \in \mathbb{N} \setminus \{0\}$ die gespeicherte Zahl in diesem Knoten ist. Gemäß der Definition (2.6) hat jeder Pfad $P = [(a_1, count_1), \dots, (a_l, count_l)]$ in einem Präfix-Baum die Eigenschaft: $count_1 \geq count_2 \geq \dots \geq count_l$.

Das Beispiel (2.4) zeigt alle Portionen der im rechten Teil der Tabelle (2.6) dargestellten Datenbank. Die entsprechenden Präfix-Bäume sind in Abbildung (2.1) anzusehen.

Beispiel 2.4 Betrachtet man die Datenbank, die im rechten Teil der Tabelle (2.6) dargestellt ist, dann sind ihre Portionen:

$$\mathcal{P}_1 = \{[d, a], [d, c, a, e], [d, b], [d, b, c], [d, b, e], [d, c], [d, b, a]\}$$

$$\mathcal{P}_2 = \{[b, c], [b, c, e]\}$$

Das Präfix der ersten Portion ist $\text{pre}(\mathcal{P}_1) = d$ und das Präfix der zweiten Portion ist $\text{pre}(\mathcal{P}_2) = b$. Die Abbildung (2.1) zeigt den Präfix-Baum von jeweils \mathcal{P}_1 und \mathcal{P}_2 .

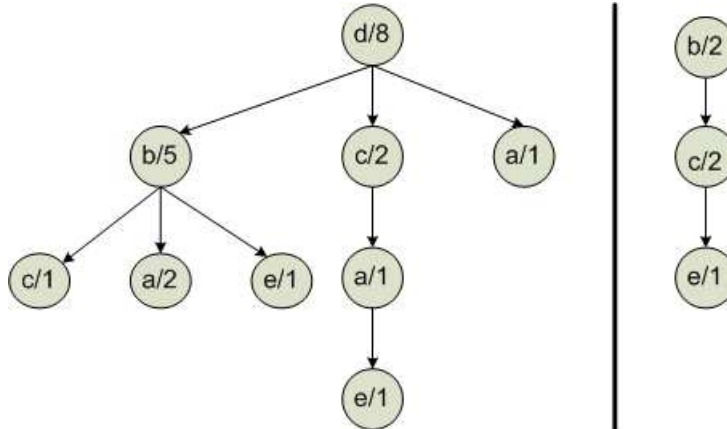


Abbildung 2.1: **Der linke Baum** ist der Präfix-Baum der Portion \mathcal{P}_1 aus dem Beispiel (2.4). **Der rechte Baum** ist der Präfix-Baum der Portion \mathcal{P}_2 aus dem Beispiel(2.4).

Mithilfe des Begriffes „Präfix-Baum“ kann die FP-tree-Struktur in der folgenden Definition formuliert werden.

Definition 2.7 (FP-tree-Struktur) Eine FP-tree-Struktur (a frequent-pattern tree) der Datenbank \mathcal{D} ist ein Baum mit den folgenden Eigenschaften:

- Die Wurzel ist mit dem Wort „root“ beschriftet.
- Die Kinder der Wurzel sind die Präfix-Bäume aller Portionen der Datenbank \mathcal{D}
- Alle Knoten, in denen das selbe Objekt gespeichert ist, sind in derjenigen Liste gespeichert, mit der in der Header-Tabelle dieses Objekt assoziiert ist. Mit anderen Worten, ist das Objekt $a_j \in \mathcal{I}$ in der Header-Tabelle mit der Liste $list$ assoziiert ist, dann sind alle dieses Objekt a_j enthaltenden Knoten in der Liste $list$ zu speichern.

Die Abbildung (2.2) zeigt die FP-tree-Struktur der im rechten Teil der Tabelle (2.6) dargestellten Datenbank. Wenn man sich diese Abbildung ansieht, erkennt man, dass die absteigende Sortierung der häufigen Objekte nach deren Häufigkeit zu einer effizienten Speicherung der Transaktionen in der FP-tree-Struktur führt. Der Grund dafür liegt darin, dass häufigere Mengen mehr Transaktionen gemeinsam teilen als weniger häufige Mengen.

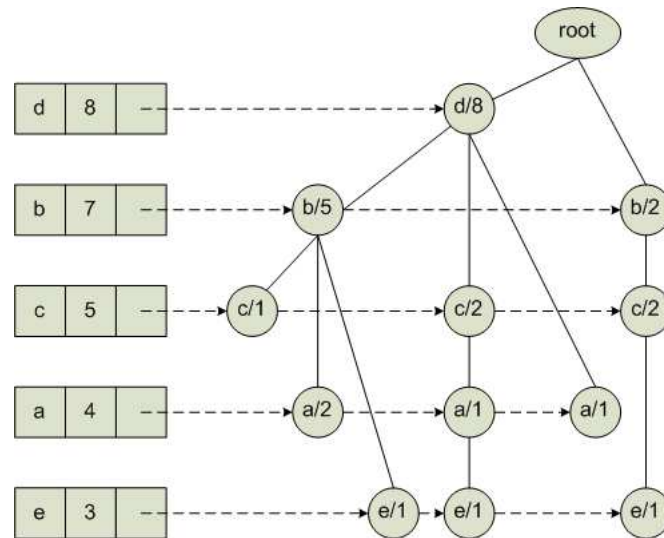


Abbildung 2.2: Die FP-tree-Struktur der im rechten Teil der Tabelle (2.6) dargestellten Datenbank.

Die Vorbereitung der Datenbank und die Konstruktion der FP-tree-Struktur sind in den folgenden zwei Schritten zusammenzufassen:

1. **Schritt (Konstruktion der Header-Tabelle):** In diesem Schritt wird die Datenbank durchgelaufen, um den Support jedes Objektes $a \in \mathcal{I}$ zu berechnen. Gleichzeitig wird jedes häufige Objekt mit seinem Support in die Header-Tabelle HT eingefügt und mit einer leeren Liste $list$ assoziiert. Danach werden die Objekte in der Header-Tabelle nach deren Support absteigend sortiert.
2. **Schritt (Konstruktion der FP-tree-Struktur):** In diesem Schritt wird die Datenbank \mathcal{D} zum zweiten und gleichzeitig zum letzten Mal durchgelaufen. Für jede Transaktion $\mathcal{T} = (id, I) \in \mathcal{D}$ werden alle häufigen Objekte in der Menge I in eine Liste L eingefügt und in dieser nach deren Support sortiert. Danach wird die Liste L in die FP-tree-Struktur eingefügt, wobei diese Struktur am Anfang, also vor dem zweiten Durchlauf, als ein einziger mit „root“ beschrifteter Knoten zu initialisieren ist. Für das Einfügen einer Liste L in die FP-tree-Struktur wird die Prozedur `insert` aufgerufen, die in **Algorithm (2)** formuliert ist.

FP-growth

Basierend auf der FP-tree-Struktur wird der FP-growth- Algorithmus in diesem Unterabschnitt theoretisch untersucht und entsprechend formuliert.

Algorithm 2 Eine Prozedur zum Einfügen einer Liste (Transaktion) L in die FP-tree-Struktur

```

1: procedure INSERT( $L, HT, node$ )
2:   if  $L \neq \emptyset$  then
3:      $a \leftarrow L[1]$ 
4:     if  $a \in node$  then
5:        $node \cdot count \leftarrow node \cdot count + 1$ 
6:        $L \leftarrow L \setminus [a]$ 
7:       insert( $L, HT, node$ )
8:     else if  $\exists node' : node' \in node \cdot children \wedge a \in node'$  then
9:       insert( $L, HT, node'$ )
10:    else
11:       $newNode \leftarrow \text{new Node}()$ 
12:       $newNode \cdot a \leftarrow a ; newNode \cdot count \leftarrow 1$ 
13:       $node.children \leftarrow node.children \cup \{newNode\}$ 
14:       $HT[a \cdot pos] \cdot list \leftarrow HT[a \cdot pos] \cdot list + [newNode]$ 
15:       $L \leftarrow L \setminus [a]$ 
16:      insert( $L, HT, newNode$ )
17:    end if
18:  end if
19: end procedure

```

Definition 2.8 (Präfix-Pfad eines Objektes) Sei $P = [(a_1, count_1) \cdots, (a_i, count_i), \cdots, (a_l, count_l)]$ ein kompletter Pfad in einem Präfix-Baum einer FP-tree-Struktur. Dann ist ein Präfix-Pfad des Objektes a_i ($1 < i \leq l$) das Paar $([a_1, \cdots, a_{i-1}], count_i)$, welches durch $P|a_i$ gekennzeichnet wird.

Definition 2.9 (Bedingte Datenbank) Sei eine FP-tree-Struktur einer Datenbank \mathcal{D} gegeben. Die Menge aller Präfix-Pfade eines Objektes $a \in HT$ heißt die bedingte Datenbank gegeben a , die durch $\mathcal{D}|a$ gekennzeichnet wird. D.h.

$$\mathcal{D}|a = \{([a_1, \cdots, a_k], count) \mid ([a_1, \cdots, a_k], count) \text{ ist ein Präfix-Pfad von } a\}$$

Aus der obigen Definition ist aufzufassen, dass die Menge $\mathcal{D}|a$ eine Transaktionsdatenbank über der Menge $\mathcal{I}|a = (\bigcup_{(I, \cdot) \in \mathcal{D}|a} I)$ ist. Der Support einer beliebigen Menge $X \in \mathcal{I}|a$ in $\mathcal{D}|a$ ist nach der folgenden Vorschrift zu berechnen: $sup(X, \mathcal{D}|a) = (\sum_{(L, count): X \subseteq L} count)$. Die aus $\mathcal{D}|a$ konstruierte FP-tree-Struktur heißt entsprechend die bedingte FP-tree-Struktur gegeben a , die durch $FP\text{-tree}|a$ bezeichnet wird. Das Beispiel (2.5) verdeutlicht die Definitionen (2.8) und (2.9).

Beispiel 2.5 Es wird die in Abbildung (2.2) gezeigte FP-tree-Struktur betrachtet. $([d, b], 2)$ ist der Präfix-Pfad von a , der aus dem Pfad $[(d, 8), (b, 5), (a, 2)]$ erzeugt ist. Die Menge aller Präfix-Pfade von a ist:

$$\mathcal{D}|a = \{([d, b], 2), ([d, c], 1), ([d], 1)\}.$$

Es gilt in dieser Datenbank $sup(\{d\}, \mathcal{D}|a) = 4$, $sup(\{b\}, \mathcal{D}|a) = 2$ und $sup(\{c\}, \mathcal{D}|a)$

= 1. Dies bedeutet für den minimalen Support $s = 1$, dass die Mengen $\{d\}$ und $\{b\}$ häufig in $\mathcal{D}|\{a\}$ sind, aber die Menge $\{c\}$ nicht häufig. $\mathcal{D}|\{a\}$ sieht also nach dem Vorbereitungsschritt so aus: $\mathcal{D}|\{a\} = \{([d, b], 2), ([d], 1), ([d], 1)\}$. Falls das Einfügen von (L, count) in die FP-tree $|\{a\}$ die Erzeugung eines neuen Knoten verlangt, dann muss in der Zeile (12) in **Algorithm (2)** dem Zähler des neu erzeugten Knoten der Wert count zugewiesen werden. Dementsprechend besteht FP-tree $|\{a\}$ nur aus einem einzigen Pfad, nämlich dem Pfad $[(d, 4), (b, 2)]$.

Um die Bedingte Datenbank $\mathcal{D}|\{a_i\}$ zu konstruieren, braucht man nach der Definition (2.9) alle Präfix-Pfade von a_i zu berechnen. Dies verlangt wiederum, dass man alle das Objekt a_i enthaltenden Pfade zu besuchen hat. Mithilfe der Header-Tabelle $HT = [(a_1, s_1, L_1), \dots, (a_i, s_i, L_i), \dots, (a_h, s_h, L_h)]$ kann man alle a_i enthaltenden Pfade erreichen, indem man die Liste L_i durchläuft. Die Begründung dieses Verfahrens liegt in der Tatsache, dass alle a_i enthaltenden Knoten des Baumes der FP-tree-Struktur in L_i gespeichert sind. Dieses Verfahren zur Konstruktion einer bedingten Datenbank gegeben irgendein Objekt und die folgenden Lemmata (2.3) und (2.4) sind die Gerüste der FP-growths Formulierung.

Lemma 2.3 Sei eine FP-tree-Struktur einer Datenbank \mathcal{D} über \mathcal{I} gegeben. $X \subseteq \mathcal{I}|\{a\}$ ist eine häufige Menge in $\mathcal{D}|\{a\}$ genau dann, wenn die Menge $X \cup \{a\}$ eine häufige Menge in \mathcal{D} ist.

Außerdem gilt $\text{sup}(X, \mathcal{D}|\{a\}) = \text{sup}(X \cup \{a\}, \mathcal{D})$.

Beweis: Zuerst werden wir $\mathcal{D}|\{a\}$ in die Form

$$\mathcal{D}^*|\{a\} = \{ \underbrace{(id_{i_1}, L), \dots, (id_{i_{\text{count}}}, L)}_{\text{count Mal}} \mid (L, \text{count}) \in \mathcal{D}|\{a\} \}$$

umschreiben, wobei folgendes gilt:

$$\forall (id_i, L_i), (id_j, L_j) \in \mathcal{D}^*|\{a\} : (id_i, id_j \in \mathbb{N}) \wedge (L_i \neq L_j \Leftrightarrow id_i \neq id_j)$$

Die Datenbank $\mathcal{D}|\{a\}$ ist äquivalent zur Datenbank $\mathcal{D}^*|\{a\}$ im folgenden Sinn:

$$\forall X \in \mathcal{I}|\{a\} : \text{sup}(X, \mathcal{D}|\{a\}) = \text{sup}(X, \mathcal{D}^*|\{a\})$$

was offensichtlich ist. Für eine beliebige $X \subseteq \mathcal{I}|\{a\}$ definieren wir:

$$\mathcal{T}_{X \cup \{a\}} = \{T \in \mathcal{D} \mid X \cup \{a\} \subseteq T\}$$

$$\mathcal{T}_X|\{a\} = \{T \in \mathcal{D}^*|\{a\} \mid X \subseteq T\}$$

Die Menge $\mathcal{T}_{X \cup \{a\}}$ ist die Menge aller Transaktionen in \mathcal{D} , die die Menge $X \cup \{a\}$ enthalten. Die Menge $\mathcal{T}_X|\{a\}$ ist die Menge aller Transaktionen in $\mathcal{D}^*|\{a\}$, die X enthalten. Nach der Definitionen (2.8) und (2.9) und der Konstruktion von $\mathcal{D}^*|\{a\}$ entspricht jede Transaktion T aus $\mathcal{T}_X|\{a\}$ genau und nur genau einer Transaktion aus $\mathcal{T}_{X \cup \{a\}}$, aus der die Transaktion T konstruiert ist. Daraus und aus der Tatsache, dass die Mengen $\mathcal{T}_{X \cup \{a\}}, \mathcal{T}_X|\{a\}$ endlich sind, ergibt sich:

$$s < \text{sup}(X, \mathcal{D}^*|\{a\}) = |\mathcal{T}_X|\{a\}| = |\mathcal{T}_{X \cup \{a\}}| = \text{sup}(X \cup \{a\}, \mathcal{D})$$

Dies bedeutet: $X \in \mathcal{F}(\mathcal{D}|\{a\}, s) \Leftrightarrow X \cup \{a\} \in \mathcal{F}(\mathcal{D}, s)$. □

Lemma 2.4 Sei $HT = [(a_1, s_1, L_1), \dots, (a_i, s_i, L_i), \dots, (a_h, s_h, L_h)]$ die Header-Tabelle einer FP-tree-Struktur einer Datenbank \mathcal{D} . Die Menge $\bigcup_{a \in \{a_i, \dots, a_1\}} \mathcal{H}_{\{a\}}$ ist die Menge aller häufigen Mengen, die a_i ($1 \leq i \leq h$) enthalten.

Dabei ist $\mathcal{H}_{\{a\}} = \{X \cup \{a\} \mid X \in \mathcal{F}(\mathcal{D}|\{a\}, s)\}$.

Beweis: Nach dem Lemma (2.3) stellt die Menge $\mathcal{H}_{\{a_i\}}$ ($1 \leq i \leq h$) die Menge aller häufigen Mengen dar, die sich nur aus a_i und Elementen aus $\{a_{i-1}, \dots, a_1\}$ konstruieren lassen, weil $\mathcal{I}\{a_i\} \subseteq \{a_{i-1}, \dots, a_1\}$ ist. Für ein bestimmtes $a_i \in HT$ haben wir also:

$\mathcal{H}_{\{a_h\}}$ ist die Menge aller häufigen Mengen, die nur aus a_h und Elementen aus $\{a_h, \dots, \underline{a_i}, \dots, a_1\}$ bestehen,

$\mathcal{H}_{\{a_{h-1}\}}$ ist die Menge aller häufigen Mengen, die nur aus a_{h-1} und Elementen aus $\{a_{h-1}, \dots, \underline{a_i}, \dots, a_1\}$ bestehen,

⋮

$\mathcal{H}_{\{a_{i+1}\}}$ ist die Menge aller häufigen Mengen, die nur aus a_{i+1} und Elementen aus $\{a_{i+1}, \underline{a_i}, \dots, a_1\}$ bestehen,

$\mathcal{H}_{\{a_i\}}$ ist die Menge aller häufigen Mengen, die nur aus a_i und Elementen aus $\{\underline{a_i}, \dots, a_1\}$ bestehen,

$\mathcal{H}_{\{a_{i-1}\}}$ ist die Menge aller häufigen Mengen, die nur aus a_{i-1} und Elementen aus $\{a_{i-1}, \dots, a_1\}$ bestehen. Also gibt es keine häufige Menge aus $\mathcal{H}_{\{a_{i-1}\}}$, die a_i enthalten kann. Dies gilt für alle $a_{i-1}, a_{i-2}, \dots, a_1$. Also ist $\mathcal{H}_{\{a_h\}} \cup \mathcal{H}_{\{a_{h-1}\}} \cup \dots \cup \mathcal{H}_{\{a_i\}}$ die Menge aller häufigen Mengen, die a_i enthalten. \square

Die FP-growths Formulierung ist in **Algorithm (3)** gezeigt. Jeder Durchlauf der Hauptschleife in der Zeile (3) erzeugt die Menge $\mathcal{H}_{\{a\}}$ für ein $a \in HT$. Dies passiert indirekt in den Zeilen (7), (8), (9) und (10). Für die Generierung aller häufigen Mengen, die ein bestimmtes Objekt a enthalten, spielt die Reihenfolge, in der die Header-Tabelle HT durchgelaufen wird, überhaupt keine Rolle. Zur Konstruktion der bedingten Datenbank $\mathcal{D}\{a\}$ in der Zeile (5) wird, wie schon vorher erwähnt wurde, die mit dem Objekt a in der Header-Tabelle assoziierte Liste durchgelaufen, um alle das Objekt a enthaltenden Pfade zu erreichen und danach die entsprechenden Präfix-Pfade von a zu erzeugen. Zur Konstruktion der bedingten FP-tree-Struktur von a in der Zeile (6) werden die im Unterabschnitt „FP-tree-Struktur“ auf der Seite (18) vorgestellten Verfahren auf die Datenbank $\mathcal{D}\{a\}$ angewendet.

Die Eigenschaft: $\forall a_i, a_j \in HT : i \neq j \Rightarrow \mathcal{H}_{\{a_i\}} \cap \mathcal{H}_{\{a_j\}} = \emptyset$ kann zur Lösung des Problems ausgenutzt werden, welches dadurch entsteht, dass die Datenbank \mathcal{D} nicht in den Hauptspeicher passt. Die Lösung dieses Problems ist ähnlich der bei Eclat-Algorithmus vorgeschlagenen Lösung. Für jedes Objekt $a \in HT$ wird also die Menge aller a enthaltenden Transaktionen als selbstständige Datenbank betrachtet und nach deren Ladung in den Hauptspeicher wird FP-growth separat auf sie angewendet. Dabei ist es zu erkennen, dass die Menge $\mathcal{H}_{\{a\}}$ in FP-growth-Algorithmus genau die Rolle der Menge $\mathcal{I}\{a\}$ in Eclat-Algorithmus spielt. Die wesentliche Gemeinsamkeit zwischen den beiden Algorithmen ist aber die, dass diese beiden Algorithmen der „teile und herrsche“-Strategie zur Berechnung der Menge $\mathcal{F}(\mathcal{D}, s)$ folgen. Diese Strategie führt dazu, dass die beiden Algorithmen durch die rekursiven Aufrufe viel Hauptspeicher konsumieren. Um dieses Problem bei FP-growth-Algorithmus zu mindern, wird von der folgenden Eigenschaft der FP-tree-Struktur in FP-growths Implementierung Gebrauch gemacht.

Lemma 2.5 *Besteht die FP-tree-Struktur einer Datenbank \mathcal{D} aus einem einzigen Pfad $P = [(a_1, count_1), \dots, (a_i, count_i), \dots, (a_l, count_l)]$, dann ist jede Kombination $X \subseteq$*

Algorithm 3 FP-growth-Algorithmus

```

1: function FP-GROWTH(FP-tree,s)
2:    $\mathcal{F}(\mathcal{D}, s) \leftarrow \emptyset$ 
3:   for all  $a \in HT$  do
4:      $\mathcal{F}(\mathcal{D}, s) \leftarrow \mathcal{F}(\mathcal{D}, s) \cup \{\{a\}\}$ 
5:     konstruiere  $\mathcal{D}|\{a\}$ 
6:     konstruiere FP-tree $|\{a\}$ 
7:      $\mathcal{F}(\mathcal{D}|\{a\}, s) \leftarrow \text{FP-growth}(\text{FP-tree}|\{a\}, s)$ 
8:     for all  $X \in \mathcal{F}(\mathcal{D}|\{a\}, s)$  do
9:        $(X \cup \{a\}) \cdot \text{sup} \leftarrow X \cdot \text{sup}$ 
10:       $\mathcal{F}(\mathcal{D}, s) \leftarrow \mathcal{F}(\mathcal{D}, s) \cup \{X \cup \{a\}\}$ 
11:    end for
12:  end for
13:  return  $\mathcal{F}(\mathcal{D}, s)$ 
14: end function

```

$\{a_1, \dots, a_i, \dots, a_l\}$ eine häufige Menge in \mathcal{D} , wobei gilt: $\text{sup}(X, \mathcal{D}) = \min\{\text{count}_i \mid a_i \in X\}$.

Der Beweis ergibt sich direkt aus der Konstruktion der FP-tree-Struktur. Diese Eigenschaft ist besonders nützlich, wenn die Datenbank sehr lange häufige Mengen enthält.

Techniken zur Implementierung des FP-growth werden wir im Abschnitt (7.1) diskutieren.

3 Entdeckung häufiger Muster in zeitlichen Daten

Eine der Hauptaufgaben im Data Mining ist die Entdeckung von zeitlichen kausalen Zusammenhängen in Daten, die zeitliche Informationen enthalten. Eine Form solcher Daten sind Sequenzen, zeitliche Folgen von qualitativen Beobachtungswerten eines bestimmten Vorgangs. Sequenzen kann man mit Zeitreihen vergleichen, wobei eine Zeitreihe eine zeitlich geordnete Folge quantitativer Beobachtungswerte ist ([8]). Beide stellen also zeitliche Folgen von Beobachtungswerten dar. Allerdings handelt es sich bei Zeitreihen um numerische und somit geordnete Beobachtungswerte. Während in Zeitreihen Trends und typische Formen gesucht werden können, können Sequenzen nur nach Mustern durchsucht werden, ähnlich wie Zeichenketten.

In diesem Kapitel werden verschiedene Formen von Sequenzen vorgestellt und jeweils eine Data Mining-Aufgabe entsprechend formuliert.

Nach einer ausführlichen Beschreibung von Ereignis-Sequenzen im Abschnitt (3.1) wird der WINEPI-Algorithmus ([9]) vorgestellt, der alle häufigen Muster (Episoden) in solchen Sequenzen findet. Ein Beispiel für Ereignis-Sequenzen sind Alarmmeldungen in Telekommunikationsnetzen. In diesen Alarmmeldungen kann man nach Ereignissen suchen, die größeren Ausfällen des Netzwerkes vorausgehen, um diesen vorzubeugen.

Im Abschnitt (3.2) werden Verkauf-Sequenzen ([10], [11]) vorgestellt. Ein Beispiel dafür sind die Einkäufe, die ein Kunde nacheinander bei einem Online-Händler oder einem Supermarkt macht. In diesen Einkauf-Transaktionen könnte man nach typischen Folgen von Einkäufen suchen, die von vielen Kunden gemacht werden, um anderen Kunden entsprechende Angebote machen zu können und den Umsatz zu steigern.

Eine andere Form von Sequenzen sind Intervall-Sequenzen ([13], [14]), die im Abschnitt (3.3) vorgestellt sind. Ein Beispiel für eine Intervall-Sequenz ist der Krankheitsverlauf bei einem chronisch Kranken. Eine Zeit lang geht es dem Kranken gut, während es später vielleicht eine Periode mit einer Krankheit gibt. Hier wäre interessant, in allen Patientenakten chronisch Kranker nach häufigen Intervall-Mustern zu suchen, um vielleicht zu einem besseren Verständnis der Krankheit zu kommen und die Patienten besser behandeln zu können.

Es ist beim Lesen dieses Kapitels zu beachten, dass jede Menge definierter Symbole nur in demjenigen Abschnitt gilt, in dem sie definiert ist.

3.1 Häufige Episoden in Ereignis-Sequenzen

3.1.1 Problembeschreibung

Ereignis-Sequenz

Sei eine endliche und nicht leere Menge $E = \{e_1, e_2, \dots, e_m\}$ von Symbolen gegeben, wobei jedes Symbol $e_i \in E$ Ereignis-Typ heißt. Für ein $e \in E$ und ein $t \in \mathbb{Z}$ heißt das Paar (e, t) ein Ereignis. Eine Ereignis-Sequenz über E ist ein Tripel $\mathcal{S} = (S, T_s, T_e)$ mit der folgenden Interpretation:

- $T_s \in \mathbb{Z}$ ist der Anfang der Ereignis-Sequenz und $T_e \in \mathbb{Z}$ ist das Ende der Ereignis-Sequenz, wobei $T_s < T_e$ gilt.
- $S = [(e_1, t_1), (e_2, t_2), \dots, (e_n, t_n)]$ eine Folge von n Ereignissen, wobei $e_i \in E$, $T_s \leq t_i < T_e$ für alle $i = 1, \dots, n$ und $t_i \leq t_{i+1}$ für alle $i = 1, \dots, n-1$ ist.

Ein Fenster auf einer Ereignis-Sequenz $\mathcal{S} = (S, T_s, T_e)$ ist eine Ereignis-Sequenz $\mathcal{W} = (W, T_{ws}, T_{we})$, die folgende Bedingungen erfüllt:

- $T_{ws} < T_e$ und $T_{we} > T_s$
- Die Folge W besteht nur aus allen Ereignissen (e, t) , die in der Folge S vorkommen und für die $T_{ws} \leq t < T_{we}$ gilt.

Der Wert $width(\mathcal{W}) = T_{we} - T_{ws}$ heißt die Breite des Fensters \mathcal{W} . Die Menge aller Fenster auf einer Ereignis-Sequenz \mathcal{S} , die die selbe Breite win haben, wird durch $WIN(\mathcal{S}, win)$ bezeichnet. Das folgende Lemma gibt die Kardinalität der Menge $WIN(\mathcal{S}, win)$ an.

Lemma 3.1 *Für eine gegebene Ereignis-Sequenz $\mathcal{S} = (S, T_s, T_e)$ und ein gegebenes $win \in \mathbb{N} \setminus \{0\}$ ist $|WIN(\mathcal{S}, win)| = T_e - T_s + win - 1$.*

Beweis: Jedes $\mathcal{W} = (W, T_{ws}, T_{we}) \in WIN(\mathcal{S}, win)$ muss die Bedingungen $T_{ws} < T_e$ und $T_{we} > T_s$ erfüllen. Wir betrachten das erste und das letzte Fenster. Wegen $T_{we} > T_s$ muss das Ende des ersten Fensters gleich $T_s + 1$ sein, da $T_s + 1$ die kleinste Zahl $\in \mathbb{Z}$ ist, die größer als T_s ist. Also ist der Anfang dieses Fensters gleich $T_s + 1 - win$. Und wegen $T_{ws} < T_e$ muss der Anfang des letzten Fensters gleich $T_e - 1$ sein, da $T_e - 1$ die größte Zahl $\in \mathbb{Z}$ ist, die kleiner als T_e ist. Da man jedes Fenster mit seinem Anfang eindeutig identifizieren kann, gilt:

$$|WIN(\mathcal{S}, win)| = |[T_s + 1 - win, T_e - 1] \cap \mathbb{Z}|$$

$$|WIN(\mathcal{S}, win)| = (T_e - 1) - (T_s + 1 - win) + 1 = T_e - T_s + win - 1. \quad \square$$

Beispiel 3.1 *Eine Ereignis-Sequenz über $E = \{a, b, c, e, f\}$ ist zum Beispiel:*

$$\mathcal{S} = ([[(b, 2), (f, 3), (c, 3), (b, 4), (a, 5), (c, 8), (a, 9), (e, 10)], 1, 11]).$$

Es gibt 14 Fenster der Breite $win = 5$ auf \mathcal{S} . Das erste Fenster ist $\mathcal{W}_1 = (\emptyset, -3, 2)$. Also kann ein Fenster keine Ereignisse enthalten. Das letzte Fenster ist $\mathcal{W}_{14} = ([[(e, 10)], 10, 15])$. Das fünfte Fenster ist $\mathcal{W}_5 = ([[(b, 2), (f, 3), (c, 3), (b, 5)], 1, 6)$ und das sechste Fenster ist $\mathcal{W}_6 = ([[(b, 2), (f, 3), (c, 3), (b, 5), (a, 6)], 2, 7)$.

Episode

Wenn man die in der Abbildung (4.1) dargestellte Ereignis-Sequenz \mathcal{S} über $E = \{a, b, c, d, e, f\}$ betrachtet, erkennt man, dass die Ereignis-Typen a, b, c in der Reihenfolge $[b, c, a]$ in \mathcal{S} vorkommen. In dieser Ereignis-Sequenz ist auch zu erkennen, dass der Ereignis-Typ d nach den Ereignis-Typen e und f vorkommt. Dabei spielt die Reihenfolge, in der e und f vorkommen, keine Rolle. Dieser Sachverhalt wird durch $[\{e, f\}, d]$ ausgedrückt. Die Ereignis-Typen a und d kommen auch in \mathcal{S} vor, was durch $\{a, d\}$ ausgedrückt wird, falls man sich für die Reihenfolge nicht interessiert, in der a und d in \mathcal{S} vorkommen. $[b, c, a], [\{e, f\}, d]$ und $\{a, d\}$ werden als Episoden bezeichnet. Eine Episode ist also eine geordnete Kollektion von Ereignis-Typen. In diesem Unterabschnitt werden die Begriffe „Episode“, „Unterepisode“ und „Vorkommen einer Episode in einer Ereignis-Sequenz“ formal beschrieben. Außerdem wird die Aufgabe der Entdeckung häufiger Episoden in einer Ereignis-Sequenz formuliert. Zur Formulierung des Begriffes „Episode“ sind zuerst

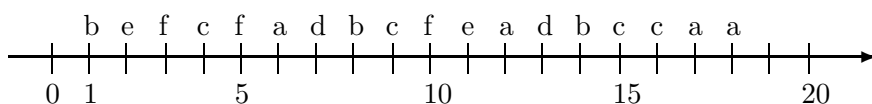


Abbildung 3.1: Eine Ereignis-Sequenz $\mathcal{S} = (S, 1, 20)$ über $E = \{a, b, c, d, e, f\}$

die Definition einer Ordnungsrelation und die Definition einer injektiven Funktion in Erinnerung zu rufen.

Definition 3.1 (Ordnungsrelation) Eine Menge $R \neq \emptyset$ ist eine binäre Relation in der Menge $X \neq \emptyset$ genau dann, wenn $R \subseteq X \times X$ gilt. Eine binäre Relation R in X heißt Ordnungsrelation in X genau dann, wenn R folgendes erfüllt:

1. $\forall x \in X : (x, x) \in R$
2. $\forall x, y \in X : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$
3. $\forall x, y, z \in X : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Gilt $\forall x, y \in X : ((x, y) \in R \vee (y, x) \in R)$ für eine Ordnungsrelation R in X , dann heißt R totale Ordnungsrelation in X . Existiert eine totale Ordnungsrelation in X , dann heißt X total geordnete Menge. Die Ordnungsrelation R in X heißt trivial genau dann, wenn $R = \{(x, x) \mid x \in X\}$ ist.

Zur Vereinfachung der Schreibweise wird $(x, y) \in R$ als $x \leq y$ bezeichnet, falls die Ordnungsrelation R aus dem Kontext hervorgeht.

Definition 3.2 (Injektive Funktion) Eine Funktion $f : X \rightarrow Y$ ist eine injektive Funktion, genau dann, wenn sie folgendes erfüllt:

$$\forall x_1, x_2 \in X : f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

Definition 3.3 (Episode) Sei V eine endliche, nicht leere Menge von Knoten gegeben. Für eine Funktion $f : V \rightarrow E$, die jeden Knoten $x \in V$ mit einer Ereignis-Type $e \in E$ beschriftet, und für eine Ordnungsrelation R in V heißt das Tripel $\alpha = (V, R, f)$ Episode. Die Größe der Episode α , die durch $|\alpha|$ bezeichnet ist, ist die Kardinalität der Knotenmenge V .

Um die Notation einer Episode $\alpha = (V, R, f)$ vereinfachen zu können, wird oft jeder Knoten $x \in V$ durch $f(x)$ ersetzt. Es ist dabei zu beachten, dass für alle $x_1 \dots x_k \in V$ mit $x_1 \neq \dots \neq x_k \wedge f(x_1) = \dots = f(x_k) = e$ der Ereignis-Typ e k - Mal in V vorkommt. Dies bedeutet, dass die Funktion f in dieser Notation impliziert angegeben ist. Es soll aus dem Kontext hervorgeht, um welche Notation es sich handelt.

Anhand der Relation R kann man zwei besondere Arten von Episoden spezifizieren. Falls die Ordnungsrelation R eine totale Ordnungsrelation in V ist, dann heißt die Episode $\alpha = (V, R, f)$ **serielle Episode**. Eine serielle Episode mit $V = \{x_1, \dots, x_{|V|}\}$ wird durch $[x_1, \dots, x_{|V|}]$ ausgedrückt, wobei $\forall i, j \in \{1, \dots, |V|\} : i < j \Rightarrow (x_i, x_j) \in R$ gilt. Falls die Relation R eine triviale Ordnungsrelation in V ist, dann bezeichnet man die Episode $\alpha = (V, R, f)$ als **parallele Episode**. Eine parallele Episode wird mit ihrer Knotenmenge V identifiziert. Ist eine Episode keine parallele und keine serielle Episode, dann wird sie oft **generelle Episode** genannt. Auch ist eine andere Art von Episoden anhand der Funktion f zu spezifizieren. Eine Episode $\alpha = (V, R, f)$ heißt **injektive Episode**, falls die Funktion f injektiv ist. In einer injektiven Episode kommt jeder Ereignis-Typ also höchstens einmal vor. Zur Rechtfertigung der Bezeichnung der Menge V als Knotenmenge stellt man sich eine Episode $\alpha = (V, R, f)$ als einen Graphen $G = (V, R)$ vor, wobei jeder Knoten $x \in V$ mit dem Ereignis-Typ $f(x) \in E$ beschriftet ist. Die folgende Definition liegt fest, wann eine Episode in einer Ereignis-Sequenz vorkommt.

Definition 3.4 (Vorkommen einer Episode) Eine Episode $\alpha = (V, R, f)$ kommt in der Ereignis-Sequenz $\mathcal{S} = ((e_1, t_1), \dots, (e_n, t_n)), T_s, T_e$ genau dann vor, wenn eine injektive Funktion $h : V \rightarrow \{1, \dots, n\}$ existiert, die folgendes erfüllt:

- $\forall x \in V : f(x) = e_{h(x)}$
- $\forall x, y \in V : x \neq y \wedge x \leq y \Rightarrow t_{h(x)} < t_{h(y)}$

Beispiel 3.2 Es wird die in der Abbildung (4.1) dargestellte Ereignis-Sequenz \mathcal{S} über $E = \{a, b, c, d, e, f\}$ betrachtet. Die Episode $\alpha = (\{b, c, e, f\}, R, f)$, wobei $\forall x \in V : f(x) = x$ und $R = \{(b, b), (c, c), (e, e), (f, f), (b, c), (b, e), (b, f), (e, c), (f, c)\}$ gilt, kommt in \mathcal{S} vor, da die Funktion $h : V \rightarrow \{1, \dots, 18\}$ mit $h(b) = 8, h(f) = 10, h(e) = 11, h(c) = 15$ existiert, die die Bedingungen der Definition (3.4) erfüllt. Diese Episode lässt sich durch $\alpha = [b, \{e, f\}, c]$ ausdrücken. Die Abbildung (3.2) stellt α als gerichteten Graphen dar.

Definition 3.5 (Support einer Episode) Seien eine Ereignis-Sequenz \mathcal{S} und ein $win \in \mathbb{N} \setminus \{0\}$ gegeben. Der Support einer Episode $\alpha = (V, R, f)$ ist die Anzahl der Fenster in $WIN(\mathcal{S}, win)$, in denen α vorkommt. Diese Anzahl wird durch $sup(\alpha, \mathcal{S}, win)$ gekennzeichnet. D.h. $sup(\alpha, \mathcal{S}, win) = |\{\mathcal{W} \in WIN(\mathcal{S}, win) \mid \alpha \text{ kommt in } \mathcal{W} \text{ vor}\}|$.

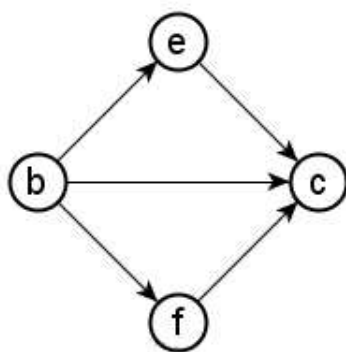


Abbildung 3.2: Die Darstellung der Episode $\alpha = [b, \{e, f\}, c]$ als gerichteter Graph.

Der Support einer beliebigen Episode α erfüllt $0 \leq \text{sup}(\alpha, \mathcal{S}, \text{win}) \leq T_e - T_s + \text{win} - 1$, was direkt aus der obigen Definition und dem Lemma (3.1) zu schließen ist. Die Wahrscheinlichkeit dafür, dass eine Episode α in einem Fenster $\mathcal{W} \in \text{WIN}(\mathcal{S}, \text{win})$ vorkommt, ist $\text{sup}(\alpha, \mathcal{S}, \text{win}) / (T_e - T_s + \text{win} - 1)$. Diese Wahrscheinlichkeit heißt die Häufigkeit des Vorkommens von α bzw. die Häufigkeit von α in der Ereignis-Sequenz \mathcal{S} . Die Häufigkeit einer Episode in \mathcal{S} wird als relativer Support bezeichnet.

Definition 3.6 (Häufige Episode in \mathcal{S}) Für ein gegebenes $s \in \mathbb{N}$ und ein gegebenes $\text{win} \in \mathbb{N} \setminus \{0\}$ ist eine Episode α häufig in \mathcal{S} genau dann, wenn $\text{sup}(\alpha, \mathcal{S}, \text{win}) > s$ ist. Die Zahl s heißt minimaler Support.

Die Menge aller häufigen Episoden in \mathcal{S} wird durch $\mathcal{F}(\mathcal{S}, \text{win}, s)$ gekennzeichnet. D.h. $\mathcal{F}(\mathcal{S}, \text{win}, s) = \{(V, R, f) \mid \text{sup}((V, R, f), \mathcal{S}, \text{win}) > s\}$. Die Berechnung dieser Menge heißt Entdeckung häufiger Episoden in der Ereignis-Sequenz \mathcal{S} . Zur Erledigung dieser Aufgabe werden Eigenschaften häufiger Episoden im Rest dieses Abschnitts untersucht. Dazu wird der Begriff „Unterepisode“ im Folgenden definiert.

Definition 3.7 (Unterepisode) Eine Episode $\beta = (V', R', f')$ ist eine Unterepisode (oder Subepisode) von einer Episode $\alpha = (V, R, f)$, was durch $\beta \preceq \alpha$ gekennzeichnet ist, genau dann, wenn eine injektive Funktion $g: V' \rightarrow V$ existiert, die folgendes erfüllt:

- $\forall x' \in V' : f'(x') = f(g(x'))$
- $\forall x', y' \in V' : x' \leq' y' \Rightarrow g(x') \leq g(y')$

Die Episode α wird als Oberepisode (oder Superepisode) von β bezeichnet.

Beispiel 3.3 Betrachten wir die Episode $\alpha = [b, \{e, f\}, c]$, dann sind $\beta_1 = \{e, f\}$, $\beta_2 = [b, c]$, $\beta_3 = [b, \{e, f\}]$ und $\beta_4 = [\{e, f\}, c]$ Unterepisoden von α .

Lemma 3.2 *Kommt $\alpha = (V, R, f)$ in \mathcal{S} vor, dann kommt jede Unterepisode β von α in \mathcal{S} vor.*

Beweis: Sei $\beta = (V', R', f')$ eine beliebige Unterepisode von α . Dann gibt es nach der Definition(3.7) eine injektive Funktion $g : V' \rightarrow V$ mit den Eigenschaften:

$$\forall x' \in V' : f'(x') = f(g(x')) \quad (1)$$

$$\forall x', y' \in V' : x' \leq' y' \Rightarrow g(x') \leq g(y') \quad (2)$$

Da α in \mathcal{S} vorkommt, existiert nach der Definition(3.4) eine injektive Funktion $h : V \rightarrow \{1, \dots, n\}$ mit den Eigenschaften:

$$\forall x \in V : f(x) = e_{h(x)} \quad (3)$$

$$\forall x, y \in V : x \neq y \wedge x \leq y \Rightarrow t_{h(x)} < t_{h(y)} \quad (4)$$

Wir definieren die Funktion $h' : V' \rightarrow \{1, \dots, n\}$ wie folgt:

$$\forall x' \in V' : h'(x') = h(g(x'))$$

Die Funktion h' ist injektiv, weil die beiden Funktionen h und g injektiv sind. Außerdem haben wir:

$$\begin{aligned} \forall x' \in V' : f'(x') &= f(g(x')) && \text{(wegen 1)} \\ &= e_{h(g(x'))} && \text{(wegen 3)} \\ &= e_{h'(x')} && \text{(Definition von } h') \end{aligned}$$

$$\forall x', y' \in V' :$$

$$\begin{aligned} x' \neq y' \wedge x' \leq' y' &\Rightarrow g(x') \neq g(y') \wedge g(x') \leq g(y') && (g \text{ injektiv und wegen 2)} \\ &\Rightarrow t_{h(g(x'))} < t_{h(g(y'))} && \text{(wegen 4)} \\ &\Rightarrow t_{h'(x')} < t_{h'(y')} && \text{(Definition von } h') \end{aligned}$$

Also erfüllt die Funktion h' die Definition (3.4) und damit kommt die Episode β in \mathcal{S} vor. \square

Das folgende Lemma zeigt eine Eigenschaft, die eine häufige Episode bzw. eine nicht häufige Episode hat. Der Beweis dieses Lemmas ergibt sich direkt aus dem Lemma (3.2).

Lemma 3.3 *Für eine beliebige Unterepisode β von α gilt:*

$$1. \alpha \in \mathcal{F}(\mathcal{S}, win, s) \Rightarrow \beta \in \mathcal{F}(\mathcal{S}, win, s)$$

$$2. \beta \notin \mathcal{F}(\mathcal{S}, win, s) \Rightarrow \alpha \notin \mathcal{F}(\mathcal{S}, win, s)$$

Der erste Teil des Lemmas (3.3) besagt, dass die Menge der häufigen Episoden in Bezug auf die Unterepisoden-Beziehung abwärts abgeschlossen ist. Während der zweite Teil besagt, dass die Menge der nicht häufigen Episoden in Bezug auf die Unterepisoden-Beziehung aufwärts abgeschlossen ist. Diese Eigenschaften sind die Basis für den Entwurf des WINEPI-Algorithmus ([9]) zur Berechnung der Menge $\mathcal{F}(\mathcal{S}, win, s)$. Er folgt, wie der im Kapitel (2) vorgestellte Apriori-Algorithmus, dem Prinzip der Kandidaten-Generierung. In der Tat ist die Arbeitsweise vom WINEPI-Algorithmus ähnlich der Arbeitsweise vom Apriori-Algorithmus, was im nächsten Abschnitt festzustellen ist. Außerdem kann man jetzt die Ähnlichkeit zwischen den Konzepten des Problems „Entdeckung

häufiger Mengen in einer Datenbank” und den Konzepten des Problems „Entdeckung häufiger Episoden in einer Ereignis-Sequenz” erkennen. Die Objektmenge \mathcal{I} entspricht der Ereignis-Sequenz \mathcal{S} über E . Die Rolle, die die Datenbank \mathcal{D} bei den häufigen Mengen spielt, ist die selbe Rolle, die die Menge $WIN(\mathcal{S}, win)$ bei den häufigen Episoden spielt. Bei der Entdeckung häufiger Mengen geht es um die Generierung der Menge $\mathcal{F}(\mathcal{D}, s)$, während es bei der Entdeckung häufiger Episoden um die Generierung der Menge $\mathcal{F}(\mathcal{S}, win, , s)$ geht.

3.1.2 WINEPI-Algorithmus

WINEPI behandelt nur die Generierung der Menge aller häufigen parallelen Episoden $\mathcal{F}^p(\mathcal{S}, win, s)$ und die Generierung der Menge aller häufigen seriellen Episoden $\mathcal{F}^s(\mathcal{S}, win, s)$. In ([9]) ist erwähnt, dass sich die Berechnung der Menge $\mathcal{F}(\mathcal{S}, win, s)$ auf die Berechnung der beiden Mengen $\mathcal{F}^p(\mathcal{S}, win, s)$ und $\mathcal{F}^s(\mathcal{S}, win, s)$ zurückführen lässt. Zur Formulierung vom WINEPI-Algorithmus vereinbaren wir folgende Notationen: \mathcal{F}_l^r mit $r = p, s$ bezeichnet die Menge aller häufigen parallelen, falls $r = p$, bzw. seriellen, falls $r = s$, Episoden der Länge l , wobei $1 \leq l \leq |E|$ gilt. \mathcal{C}_l^r mit $r = p, s$ bezeichnet die Menge aller parallelen, falls $r = p$, bzw. seriellen, falls $r = s$, Episoden der Länge l , für die gilt, dass jede echte Unterepisode von jeder beliebigen Episode $\alpha \in \mathcal{C}_l^r$ häufig ist. Es gilt $\mathcal{C}_1^p = \{\{e\} \mid e \in E\}$ und $\mathcal{C}_1^s = \{[e] \mid e \in E\}$. Dementsprechend ist das Gerüst des WINEPI-Algorithmus in **Algorithm (4)** dargestellt.

Algorithm 4 WINEPI-Algorithmus

```

1: function WINEPI( $\mathcal{S}, \mathcal{C}_1^r, win, s$ )
2:    $\mathcal{F}^r(\mathcal{S}, win, s) \leftarrow \emptyset$  ;  $l \leftarrow 1$ 
3:   while  $\mathcal{C}_l^r \neq \emptyset$  do
4:      $\mathcal{F}_l^r \leftarrow \{\alpha \in \mathcal{C}_l^r \mid sup(\alpha, \mathcal{S}, win) > s\}$ 
5:      $\mathcal{F}^r(\mathcal{S}, win, s) \leftarrow \mathcal{F}^r(\mathcal{S}, win, s) \cup \mathcal{F}_l^r$  ;  $l \leftarrow l + 1$ 
6:      $\mathcal{C}_l^r \leftarrow \{\alpha \mid |\alpha| = l \wedge (\forall \beta \preceq \alpha : |\beta| = l - 1 \Rightarrow \beta \in \mathcal{F}_{l-1}^r)\}$ 
7:   end while
8:   return  $\mathcal{F}^r(\mathcal{S}, win, s)$ 
9: end function

```

Also besteht WINEPI aus zwei Hauptaufgaben. In der ersten Hauptaufgabe (Zeile 4) geht es darum, aus der Menge \mathcal{C}_l^r diejenigen Episoden auszuwählen, die häufig sind. Dies verlangt das Durchlaufen der Ereignis-Sequenz \mathcal{S} bzw. der Menge $WIN(\mathcal{S}, win)$, um den Support jeder Episode in \mathcal{C}_l^r zu ermitteln. Dieser Schritt stellt den Kern von WINEPI dar. In der zweiten Hauptaufgabe (Zeile 6) handelt es sich um die Generierung der Kandidaten der Länge $l + 1$ aus den häufigen Episoden der Länge l . Im Folgenden wird eine Lösung für jede dieser Aufgaben beschrieben.

Kandidaten-Generierung

1. Parallele Episoden: Es wird vorausgesetzt, dass die Elemente jeder parallelen Episode $\beta = \{e_1, e_2, \dots, e_l\}$ lexikographisch geordnet sind. Aus jedem Paar $\beta_1 = \{e_1, \dots, e_{l-1}, e_l\}$,

$\beta_2 = \{e_1, \dots, e_{l-1}, e'_l\} \in \mathcal{F}_l^p$, wobei $e_l = e'_l$ oder e_l lexikographisch vor e'_l ist, wird die Episode $\alpha = \{e_1, \dots, e_{l-1}, e_l, e'_l\}$ erzeugt. Dann wird geprüft, ob jede Teilmenge der Länge l von α in der Menge \mathcal{F}_l^p enthalten ist. Falls dies der Fall ist, wird die Episode in \mathcal{C}_{l+1}^p aufgenommen. Zum Beispiel wird aus $\mathcal{F}_1^p = \{\{a\}, \{b\}, \{c\}\}$ die Kandidaten $\mathcal{C}_2^p = \{\{a, a\}, \{a, b\}, \{a, c\}, \{b, b\}, \{b, c\}, \{c, c\}\}$ generiert.

2. Serielle Episoden: Zur Generierung der Menge \mathcal{C}_{l+1}^s aus der Menge \mathcal{F}_l^s werden aus jedem Paar $\beta_1 = [e_1, \dots, e_{l-1}, e_l], \beta_2 = [e_1, \dots, e_{l-1}, e'_l] \in \mathcal{F}_l^s$ die Episoden $\alpha_1 = [e_1, \dots, e_{l-1}, e_l, e'_l]$ und $\alpha_2 = [e_1, \dots, e_{l-1}, e'_l, e_l]$ erzeugt. Danach wird für jede der erzeugten Episoden geprüft, ob jede ihrer Unterepisoden der Länge l häufig ist. Diejenige Episode, die diesen Test besteht, wird in die Menge \mathcal{C}_{l+1}^s eingefügt. Zum Beispiel wird aus $\mathcal{F}_1^s = \{\{d\}, \{e\}, \{f\}\}$ die Menge $\mathcal{C}_2^s = \{\{d, d\}, \{d, e\}, \{d, f\}, \{e, d\}, \{e, e\}, \{e, f\}, \{f, d\}, \{f, e\}, \{f, f\}\}$ generiert. Aus der Menge $\mathcal{F}_2^s = \{\{d, e\}, \{d, f\}\}$ kann keine Kandidaten generiert werden, weil weder $[e, f]$ noch $[f, e]$ in \mathcal{F}_2^s enthalten ist. Also ist $\mathcal{C}_3^s = \emptyset$.

Nach ([9]) ist die Laufzeit zur Generierung von \mathcal{C}_{l+1}^r $\mathcal{O}(l^2 |\mathcal{F}_l^r|^2 \log |\mathcal{F}_l^r|)$.

Durchlaufen der Ereignis-Sequenz

Beim Durchlaufen der Ereignis-Sequenz \mathcal{S} geht es um die Berechnung des Supports jeder Episode $\alpha \in \mathcal{C}_l^r$. Dies bedeutet, dass man für jede Episode $\alpha \in \mathcal{C}_l^r$ die Fenster $\mathcal{W} \in WIN(\mathcal{S}, win)$ zählen muss, in denen α vorkommt. Daher stellt sich die Frage, wie man die Menge $WIN(\mathcal{S}, win)$ erzeugen kann. WINEPI folgt dem Weg nicht, in dem die Menge $WIN(\mathcal{S}, win)$ explizit berechnet wird, sondern nutzt die Eigenschaft aus, die zwei direkt aufeinander folgende Fenster haben. Dabei folgt ein Fenster $\mathcal{W}' = (W', T'_{ws}, T'_{we})$ direkt auf das Fenster $\mathcal{W} = (W, T_{ws}, T_{we})$ genau dann, wenn $T'_{ws} = T_{ws} + 1$ und $T'_{we} = T_{we} + 1$ gilt. Für die beiden Fenster \mathcal{W} und \mathcal{W}' gilt die Eigenschaft:

$$W' = (W \setminus \{(e, T_{ws}) \mid (e, T_{ws}) \in S\}) \cup (W \cup \{(e, T_{we}) \mid (e, T_{we}) \in S\}).$$

Also lässt sich die Ereignis-Folge W' aus der Ereignis-Folge W erzeugen, indem man alle Ereignisse (e, T_{ws}) aus W entfernt und gleichzeitig alle Ereignisse (e, T_{we}) in W einfügt. Basierend darauf ordnet WINEPI jeder Episode $\alpha \in \mathcal{C}_l^r$ bestimmte Datenstrukturen zu, deren Zustand sich immer beim Übergang von einem Fenster \mathcal{W} in das direkt darauf folgende Fenster \mathcal{W}' ändert. Bei diesem Übergang sind zwei Fragen zu beantworten. Die Erste ist, wie man die Datenstrukturen einer Episode α ändern muss, wenn man die Ereignisse (e, T_{ws}) aus W entfernt. Die zweite Frage ist, wie man die Datenstrukturen der selben Episode α modifizieren muss, wenn man die Ereignisse (e, T_{we}) in W einfügt. Im Rest dieses Unterabschnittes werden Datenstrukturen jeweils für parallele Episoden und serielle Episoden vorgestellt und die Antworten auf die gerade gestellten Fragen gegeben. Entsprechend der obigen Diskussion ist das Skelett des Algorithmus zum Durchlaufen der Ereignis-Sequenz \mathcal{S} in **Algorithm (5)** angegeben. Es ist bei dieser Darstellung implizit zu verstehen, dass $\mathcal{W} = (W, start - 1, start + win - 1)$ und direkt darauf folgende Fenster $\mathcal{W}' = (W', start, start + win)$ ist. Im folgenden Text wird das Fenster $\mathcal{W} = (W, start - 1, start + win - 1)$ als aktuelles Fenster bezeichnet. Es ist zu beachten, dass **Algorithm (5)** mit einem leeren Fenster $(\emptyset, T_s - win, T_s) \notin WIN(\mathcal{S}, win)$ gestartet

und einem leeren Fenster $(\emptyset, T_e, T_e + win) \notin WIN(\mathcal{S}, win)$ beendet wird.

Algorithm 5 Durchlaufen der Ereignis-Sequenz zur Erzeugung von \mathcal{F}_l^r

```

1: function COMPUTE $\mathcal{F}_l^r(\mathcal{S}, \mathcal{C}_l^r, win, s)$ 
2:   /* Initialisierung der Datenstrukturen jeder Episode  $\alpha \in \mathcal{C}_l^r$  */
3:   for  $start \leftarrow T_s - win + 1$  to  $T_e$  do
4:     /* Einfügung der Ereignisse  $(e, T_{we})$  in  $W$  */
5:     for all  $(e, start + win - 1) \in S$  do
6:       /* Änderung der Datenstrukturen jeder Episode  $\alpha \in \mathcal{C}_l^r$  */
7:     end for
8:     /* Entfernung der Ereignisse  $(e, T_{ws})$  aus  $W$  */
9:     for all  $(e, start - 1) \in S$  do
10:      /* Änderung der Datenstrukturen jeder Episode  $\alpha \in \mathcal{C}_l^r$  */
11:    end for
12:  end for
13:   $\mathcal{F}_l^r \leftarrow \emptyset$ 
14:  for all  $\alpha \in \mathcal{C}_l^r$  do
15:    if  $\alpha \cdot sup > s$  then
16:       $\mathcal{F}_l^r \leftarrow \mathcal{F}_l^r \cup \{\alpha\}$ 
17:    end if
18:  end for
19:  return  $\mathcal{F}_l^r$ 
20: end function

```

1. Parallele Episoden: Jeder Episode $\alpha \in \mathcal{C}_l^p$ ist folgendes zugeordnet:

- $\alpha \cdot event_count$: In dieser Variable wird gespeichert, wieviele von den in α enthaltenen Ereignis-Typen in dem aktuellen Fenster bisjetzt gefunden wurden. α kommt in dem aktuellen Fenster genau dann vor, wenn $\alpha \cdot event_count = |\alpha|$ ist.
- $\alpha \cdot inwindow$: In dieser Variable wird der Wert T_{ws} des aktuellen Fensters, für das gerade $|\alpha| = \alpha \cdot event_count$ erfüllt wurde, gespeichert.
- $\alpha \cdot sup$: In dieser Variable wird gespeichert, in wievielen Fenstern die Episode α bisjetzt vorgekommen ist. α ist in \mathcal{F}_l^p genau dann einzufügen, wenn $\alpha \cdot sup > s$ gilt.

Jede Episode, die ein Ereignis-Typ $e \in E$ genau k -Mal enthält, wird in der Liste namens $contains(e, k)$ gespeichert. Mit anderen Worten enthält die Liste $contains(e, k)$, ($1 \leq k \leq |E|$) alle Episoden, in denen der Ereignis-Typ e k -Mal vorkommt. Außerdem wird jedem Ereignis-Typ $e \in E$ eine Variable $e \cdot count$, in der zu speichern ist, wieviele Male der Ereignis-Typ e in dem aktuellen Fenster bisjetzt gefunden wurde.

Die Initialisierung der oben genannten Variablen sieht folgendermaßen aus:

```

for all  $\alpha \in \mathcal{C}_l^p$  do
  for all  $e \in \alpha$  do
     $e \cdot \text{count} \leftarrow 0$ 
    for  $i \leftarrow 1, l$  do
       $\text{contains}(e, i) \leftarrow \emptyset$ 
    end for
  end for
end for
for all  $\alpha \in \mathcal{C}_l^p$  do
  for all  $e \in \alpha$  do
     $k \leftarrow$  Anzahl des Vorkommens von  $e$  in  $\alpha$ 
     $\text{contains}(e, k) \leftarrow \text{contains}(e, k) \cup \{\alpha\}$ 
  end for
   $\alpha \cdot \text{event-count} \leftarrow 0 ; \alpha \cdot \text{sup} \leftarrow 0$ 
end for

```

Für jedes Ereignis (e, t) , das in die Ereignis Folge W des aktuellen Fensters einzufügen ist, sind Variablen durch folgende Befehle zu modifizieren:

```

 $e \cdot \text{count} \leftarrow e \cdot \text{count} + 1$ 
for all  $\alpha \in \text{contains}(e, e \cdot \text{count})$  do
   $\alpha \cdot \text{event-count} \leftarrow \alpha \cdot \text{event-count} + e \cdot \text{count}$ 
  if  $\alpha \cdot \text{count} = l$  then
     $\alpha \cdot \text{inwindow} \leftarrow \text{start}$ 
  end if
end for

```

Diese Befehle stellen den Körper der **for all**-Schleife in der Zeile (5) der in **Algorithm (5)** formulierten Prozedur dar.

Für jedes Ereignis (e, t) , das aus der Ereignis-Folge des aktuellen Fensters zu entfernen ist, sind folgende Befehle auszuführen, aus denen der Körper der **for all**-Schleife in der Zeile (9) der in **Algorithm (5)** formulierten Prozedur besteht:

```

for all  $\alpha \in \text{contains}(e, e \cdot \text{count})$  do
  if  $\alpha \cdot \text{event-count} = l$  then
     $\alpha \cdot \text{sup} \leftarrow \alpha \cdot \text{sup} - \alpha \cdot \text{inwindow} + \text{start}$ 
  end if
   $\alpha \cdot \text{event-count} \leftarrow \alpha \cdot \text{event-count} - e \cdot \text{count}$ 
end for
 $e \cdot \text{count} \leftarrow e \cdot \text{count} - 1$ 

```

Nach ([9]) ist die Laufzeit zur Generierung von $\mathcal{F}_l^p \mathcal{O}((T_e - T_s + l^2) |\mathcal{C}_l^p|)$.

2. Serielle Episoden: Jeder seriellen Episode $\alpha = [e_{1,\alpha}, \dots, e_{l,\alpha}] \in \mathcal{C}_l^s$ wird ein deterministischer endlicher Automat $A_\alpha = (\{q_0, q_1, \dots, q_l\}, E, u, q_0, \{q_l\})$ zugeordnet, wobei die Übergangsfunktion $u : E \times \{q_0, \dots, q_l\} \rightarrow \{q_0, \dots, q_l\}$ wie folgt definiert ist:

$$u(e, q_i) = \begin{cases} q_{i+1} & : (0 \leq i < l - 1) \wedge (e = e_{i+1,\alpha}) \\ q_i & : \text{sonst} \end{cases}$$

Zur Illustration ist das Übergangsdiagramm von A_α in der Abbildung (3.3) dargestellt.

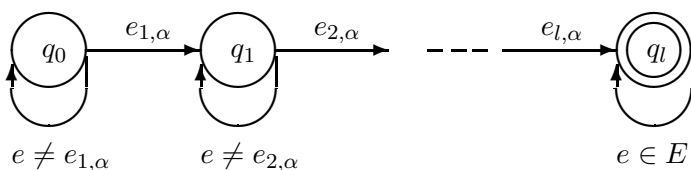


Abbildung 3.3: Ein deterministischer endlicher Automat, der die serielle Episode $\alpha = [e_{1,\alpha}, e_{2,\alpha}, \dots, e_{l,\alpha}]$ akzeptiert. Die Eingabesymbole sind die Ereignis-Typen $e \in E$

Dementsprechend kommt eine serielle Episode α in einem Fenster $\mathcal{W} = ([e_1, e_2, \dots, e_k], T_{ws}, T_{we})$ genau dann vor, wenn die Zeichenreihe $e_1 e_2 \dots e_k$ vom Automaten A_α akzeptiert wird. Zur Umsetzung dieser Idee stellt sich die Frage, wie der Automat A_α implementiert werden kann. Es wird jedem Ereignis-Typ $e \in E$ eine Liste namens $waits(e)$ zugeordnet, die Einträge der Form (α, j) enthält. Der Eintrag $(\alpha, j) \in waits(e)$ bedeutet, dass sich der Automat A_α im Zustand q_{j-1} befindet und auf die Eingabe e (besonders auf $e_{j,\alpha}$) wartet. Der aktuelle Übergang, den der Automat A_α vom Zustand q_{j-1} in den Zustand q_j auszuführen hat, wird als Eintrag der Form (α, j, t) in einer Liste namens $transitions$ gespeichert, wobei der Wert t ist der Zeitpunkt der Erzeugung von A_α . Dabei wird A_α im aktuellen Fenster $\mathcal{W} = (W, T_{ws}, T_{we})$ jedes Mal erzeugt, wenn $(e_{1,\alpha}, t) \in S$ in die Ereignis-Folge W einzufügen ist. Der Wert t in $(e_{1,\alpha}, t)$ heißt Zeitpunkt der Erzeugung von A_α in \mathcal{W} . Jede Instanz von A_α wird durch den Zeitpunkt ihrer Erzeugung identifiziert. Weil der Ereignis-Typ $e_{1,\alpha}$ maximal $|\alpha|$ -Mal in α vorkommen kann, kann A_α maximal $|\alpha|$ -Mal in einem Fenster erzeugt werden. Die Zeitpunkte der Erzeugungen der Instanzen von A_α werden in einem Array namens $\alpha \cdot initialized$ gespeichert. Dabei $\alpha \cdot initialized[i]$ enthält den Zeitpunkt der Erzeugung derjenigen Instanz von A_α , die sich im Zustand q_i befindet. Wenn eine Instanz von A_α den akzeptierenden Zustand q_l erreicht hat und keine andere Instanz von A_α diesen Zustand erreicht hatte, was durch die Abfrage von $\alpha \cdot initialized[l]$ festzustellen ist, wird der Anfang T_{ws} des aktuellen Fensters in $\alpha \cdot inwindow$ gespeichert. Dies bedeutet, dass α im aktuellen Fenster vorkommt. Wenn ein Ereignis (e, t) aus dem aktuellen Fenster zu entfernen ist, muss diejenige Instanz von A_α , die im Zeitpunkt t erzeugt wurde, auch von diesem Fenster entfernt werden. Um dies zu bewerkstelligen, werden alle Instanzen aller A_α ($\alpha \in \mathcal{C}_l^s$), die im Zeitpunkt t erzeugt wurden, in einer Liste namens $beginsat(t)$ gespeichert. Jeder Eintrag der Liste $beginsat(t)$ hat die Form (α, j) , was bedeutet, dass sich die im Zeitpunkt t erzeugte Instanz von A_α im Zustand q_j befindet.

Die Initialisierung der oben vorgestellten Datenstrukturen sieht folgendermaßen aus:

```

for all  $\alpha \in \mathcal{C}_l^s$  do
  for  $i \leftarrow 1, l$  do
     $\alpha \cdot \text{initialized}[i] \leftarrow 0$ 
     $\text{waits}(\alpha[i]) \leftarrow \emptyset$  /*  $\alpha[i] = e_{i,\alpha}$  */
  end for
end for
for all  $\alpha \in \mathcal{C}_l^s$  do
   $\text{waits}(\alpha[1]) \leftarrow \text{waits}(\alpha[1]) \cup \{(\alpha, 1)\}$ 
   $\alpha \cdot \text{sup} \leftarrow 0$ 
end for
for  $t \leftarrow T_s - \text{win}, T_s - 1$  do
   $\text{beginsat}(t) \leftarrow \emptyset$ 
end for

```

Vor jedem Durchlauf der Hauptschleife in der Zeile (3) von **Algorithm (5)** sind $\text{beginsat}(start + win - 1)$ und transitions mit der leeren Menge zu initialisieren. Für jedes ins aktuelle Fenster einzufügende Ereignis (e, t) werden alle Übergänge, die jeder A_α auszuführen hat, im folgenden Block berechnet, der den Körper der **for all**-Schleife in Zeile (5) von **Algorithm (5)** darstellt:

```

for all  $(\alpha, j) \in \text{waits}(e)$  do
  if  $j = l \wedge \alpha \cdot \text{initialized}[l] = 0$  then
     $\alpha \cdot \text{inwindow} \leftarrow start$ 
  end if
  if  $j = 1$  then
     $\text{transitions} \leftarrow \text{transitions} \cup \{(\alpha, 1, start + win - 1)\}$ 
  else
     $\text{transitions} \leftarrow \text{transitions} \cup \{(\alpha, j, \alpha \cdot \text{initialized}[j - 1])\}$ 
     $\text{beginsat}(\alpha \cdot \text{initialized}[j - 1]) \leftarrow \text{beginsat}(\alpha \cdot \text{initialized}[j - 1]) \setminus \{(\alpha, j - 1)\}$ 
     $\alpha \cdot \text{initialized}[j - 1] \leftarrow 0$ 
     $\text{waits}(e) \leftarrow \text{waits}(e) \setminus \{(\alpha, j)\}$ 
  end if
end for

```

Zur Ausführung der Übergänge von jedem A_α ist der folgende Block von Befehlen direkt nach der Zeile (7) von **Algorithm (5)** einzufügen:

```

for all  $(\alpha, j, t) \in \text{transitions}$  do
   $\alpha \cdot \text{initialized}[j] \leftarrow t$ 
   $\text{beginsat}(t) \leftarrow \text{beginsat}(t) \cup \{(\alpha, j)\}$ 
  if  $j < l$  then
     $\text{waits}(\alpha[j + 1]) \leftarrow \text{waits}(\alpha[j + 1]) \cup \{(\alpha, j + 1)\}$ 
  end if
end for

```

Zur Entfernung aller Automaten, die nicht mehr im aktuellen Fenster vorhanden sind, ist die **for all**-Schleife in der Zeile (9) durch die folgende Schleife zu ersetzen:


```

for all  $(\alpha, j) \in \text{beginsat}(start - 1)$  do
  if  $j = l$  then
     $\alpha \cdot sup \leftarrow \alpha \cdot sup - \alpha \cdot inwindow + start$ 
  else
     $waits(\alpha[j + 1]) \leftarrow waits(\alpha[j + 1]) \setminus \{(\alpha, j + 1)\}$ 
  end if
   $\alpha \cdot initialized[j] \leftarrow 0$ 
end for
    
```

Nach ([9]) ist die Laufzeit zur Generierung der Menge $\mathcal{F}_l^s \mathcal{O}((T_e - T_s)|\mathcal{C}_l^s|l)$.

3.2 Häufige Verkauf-Muster in Verkauf-Sequenzen

Sei eine Objektmenge $\mathcal{I} = \{a_1, a_2, \dots, a_{|\mathcal{I}|}\}$ gegeben. Eine Sequenz über \mathcal{I} ist eine Folge $S = [X_1, X_2, \dots, X_n]$, wobei gilt: $\forall i \in \{1, 2, \dots, n\} : X_i \subseteq \mathcal{I}$. Eine Sequenz $S = [X_1, \dots, X_n]$ über \mathcal{I} ist in der Sequenz $S' = [Y_1, \dots, Y_m]$ über \mathcal{I} ($n \leq m$) enthalten genau dann, wenn es $i_1, i_2, \dots, i_n \in \{1, 2, \dots, m\}$ mit $i_1 < i_2 < \dots < i_n$ gibt, so dass $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, \dots, X_n \subseteq Y_{i_n}$ gilt. In einer Menge von Sequenzen über \mathcal{I} ist eine Sequenz S maximal genau dann, wenn S in keiner anderen Sequenz aus dieser Menge enthalten ist.

Beispiel 3.4 *Sie $\mathcal{I} = \{a, b, c, d, e, f, g, h\}$. Die Sequenz $S = [\{b\}, \{e, f\}, \{c\}]$ ist in der Sequenz $S' = [\{g\}, \{b, c\}, \{d\}, \{e, f, h\}, \{c\}]$ enthalten, weil $\{b\} \subseteq \{b, c\}, \{e, f\} \subseteq \{e, f, h\}$ und $\{c\} \subseteq \{c\}$ gilt.*

In ([10]) sind die Konzepte „Transaktion“ und „Transaktionendatenbank“, die im Kapitel (2) vorgestellt wurden, wie folgt erweitert. Eine Transaktion über \mathcal{I} ist ein Tripel $T = (id, t, I)$ mit $id, t \in \mathbb{N}$ und $I \subseteq \mathcal{I}$. Der Wert id identifiziert eindeutig einen Kunden, der in der Zeit t die Waren I bei einem Supermarkt gekauft hat. Es gilt für zwei beliebige Transaktionen $T_1 = (id_1, t_1, I_1), T_2 = (id_2, t_2, I_2)$ über \mathcal{I} folgendes: $id_1 = id_2 \wedge t_1 = t_2 \Rightarrow I_1 = I_2$. Dies bedeutet, dass das Paar (id, t) einen Identifikator der Transaktion $T = (id, t, I)$ darstellt. Eine endliche, nicht leere Menge \mathcal{D} von Transaktionen über \mathcal{I} heißt Transaktionendatenbank über \mathcal{I} . Sind $T_1 = (id, t_1, I_1), T_2 = (id, t_2, I_2), \dots, T_n = (id, t_n, I_n)$ alle Transaktionen in \mathcal{D} , die den selben Wert id besitzen und aufsteigend nach den t_i 's Werten ($1 \leq i \leq n$) sortiert sind, dann heißt die Sequenz $S_{id} = [I_1, I_2, \dots, I_n]$ über \mathcal{I} Sequenz des Kunden id bzw. Kunde-Sequenz in \mathcal{D} . Die Menge aller Verkauf-Sequenzen in \mathcal{D} wird durch \mathcal{CS} gekennzeichnet. D.h. $\mathcal{CS} = \{S_{id} \mid S_{id} \text{ Kunde-Sequenz in } \mathcal{D}\}$. Der Support einer Sequenz S ist die Anzahl der Verkauf-Sequenzen in \mathcal{CS} , in denen S enthalten ist. Diese Anzahl wird durch $sup(S, \mathcal{CS})$ gekennzeichnet. In Bezug auf eine gegebene Zahl $s \in \mathbb{N}$ ist eine Sequenz über \mathcal{I} häufig genau dann, wenn $sup(S, \mathcal{CS}) > s$ gilt. Jede maximale Sequenz in der Menge aller häufigen Sequenzen über \mathcal{I} heißt Verkauf-Muster in \mathcal{D} . Die Berechnung der Menge aller Verkauf-Muster in \mathcal{D} , die durch $SM(\mathcal{D}, s)$ bezeichnet ist, heißt Entdeckung der Verkauf-Muster in der Datenbank \mathcal{D} .

Beispiel 3.5 Sei $\mathcal{I} = \{a, b, c, d, e, f, g, h\}$. Eine Transaktionsdatenbank \mathcal{D} über \mathcal{I} ist im linken Teil der Tabelle (3.1) gegeben. Die Menge aller Verkauf-Sequenzen in \mathcal{D} ist im rechten Teil dieser Tabelle berechnet. In Bezug auf $s = 1$ sind die Verkauf-Muster $\mathcal{SM}(\mathcal{D}, 1) = \{[\{c\}, \{h\}], [\{c\}, \{d, g\}]\}$.

Das Verkauf-Muster $[\{c\}, \{d, g\}]$ deutet darauf hin, dass über 20% der Kunden, die das Produkt c gekauft hatten, später die Produkte d und g gekauft haben.

id	t	I
1	5	$\{c\}$
1	6	$\{h\}$
2	1	$\{a, b\}$
2	3	$\{c\}$
2	4	$\{d, f, g\}$
3	5	$\{c, e, g\}$
4	5	$\{c\}$
4	6	$\{d, g\}$
4	7	$\{h\}$
5	2	$\{h\}$

id	S_{id}
1	$[\{c\}, \{h\}]$
2	$[\{a, b\}, \{c\}, \{d, f, g\}]$
3	$[\{c, e, g\}]$
4	$[\{c\}, \{d, g\}, \{h\}]$
5	$[\{h\}]$

Tabelle 3.1: Im **rechten Teil** sind alle Verkauf-Sequenzen der im **linken Teil** gegebenen Datenbank \mathcal{D} dargestellt

Der in ([10]) vorgestellte Algorithmus zur Berechnung der Menge $\mathcal{SM}(\mathcal{D}, s)$ folgt dem Prinzip der Kandidaten-Generierung. Dies lässt sich mit der Tatsache begründen, dass alle in einer häufigen Sequenz über \mathcal{I} enthaltenden Sequenzen auch häufig sein müssen. In ([11]) werden die Konzepte bzw. die Verfahren aus ([10]) erweitert. Es wird in ([11]) zum einen die Definition einer Taxonomie auf der Objektmenge \mathcal{I} erlaubt, so dass zum Beispiel aussagekräftigere oder allgemeinere Ergebnisse gefunden werden können. Zum anderen werden Zeitfenster und andere zeitliche Nebenbedingungen eingefügt. So können z.B. Transaktionen, die innerhalb eines Fensters einer festen Länge auftreten, zu einer einzigen Transaktion zusammengefasst werden. Damit werden Kunden, die viele kleine Transaktionen kurz hintereinander machen, genauso gut erfasst wie solche Kunden, die eine große Transaktion durchführen.

3.3 Häufige Intervall-Muster in Intervall-Sequenzen

Sei eine endliche, nicht leere Menge $M = \{m_1, m_2, \dots, m_{|M|}\}$ von Merkmalen gegeben. Für ein $m \in M$ und $t_s, t_e \in \mathbb{N}$ mit $t_s < t_e$ heißt das Tripel $X = (m, t_s, t_e)$ zeitliches Intervall, wobei m Markierung des Intervalls heißt und t_s, t_e den Beginn bzw. das Ende des Intervalls bezeichnen. Die Menge aller zeitlichen Intervalle über M wird durch \mathcal{I} gekennzeichnet. D.h. $\mathcal{I} = \{(m, t_s, t_e) \mid (m \in M) \wedge (t_s, t_e \in \mathbb{N}) \wedge (t_s < t_e)\}$.

Die Rolle, die ein zeitliches Intervall beim Suchen nach häufigen Intervall-Mustern spielt,

ist die selbe Rolle, die ein Ereignis beim Suchen nach häufigen Episoden spielt. Allerdings sind die zeitlichen Relationen der Intervalle untereinander komplexer als die zeitlichen Relationen der Ereignisse. Während zwei Ereignisse (e_1, t_1) und (e_2, t_2) entweder zeitgleich ($t_1 = t_2$) oder nacheinander ($t_1 < t_2$ oder $t_2 < t_1$) stattfinden können, gibt es nach ([12]) dreizehn Relationen, die zwei Intervalle aufweisen können, von denen aber sechs die Inversen anderer Relationen sind. Diese Relationen sind in der Tabelle (3.2) dargestellt.

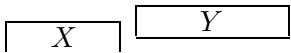
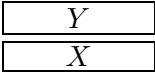

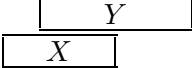
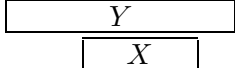
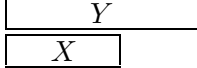
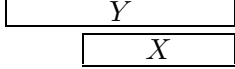
Relation	inverse Relation	graphische Darstellung	Bedingungen
X before Y	Y after X		$X \cdot t_e < Y \cdot t_s$
X equal Y	Y equal X		$X \cdot t_s = Y \cdot t_s$ $X \cdot t_e = Y \cdot t_e$
X meets Y	Y met-by X		$X \cdot t_e = Y \cdot t_s$
X overlaps Y	Y overlapped-by X		$X \cdot t_s < Y \cdot t_s$ $X \cdot t_e > Y \cdot t_s$ $X \cdot t_e < Y \cdot t_e$
X during Y	Y contains X		$X \cdot t_s > Y \cdot t_s$ $X \cdot t_e < Y \cdot t_e$
X starts Y	Y started-by X		$X \cdot t_s = Y \cdot t_s$ $X \cdot t_e < Y \cdot t_e$
X finishes Y	Y finished-by X		$X \cdot t_s > Y \cdot t_s$ $X \cdot t_e = Y \cdot t_e$

Tabelle 3.2: Intervall-Relationen

Mithilfe der zeitlichen Relationen der Intervalle können die Konzepte „Intervall-Muster“ und „Intervall-Sequenz“ beschrieben werden. In ([13]) und ([14]) sind aber diese beiden Konzepte unterschiedlich aufgefasst. Dementsprechend ist das Problem der Entdeckung häufiger Intervall-Muster in Intervall-Sequenzen jeweils anders formuliert. Im folgenden Unterabschnitt werden die beiden Formulierungen des Problems vorgestellt.

3.3.1 Häufige Intervall-Muster nach [13]

Eine Intervall-Sequenz S in ([13]) besteht aus einer Folge von Intervallen, die nach ihrer zeitlichen Enden aufsteigend geordnet sind:

$$S = [(m_1, t_{s1}, t_{e1}), \dots, (m_n, t_{sn}, t_{en})] \text{ mit } t_{ei} \leq t_{ei+1} \text{ für alle } 1 \leq i \leq n - 1.$$

Ein Intervall-Muster ist rekursiv wie folgt definiert: Ist $m \in M$ eine Intervall-Markierung, dann ist $P = m$ ein Intervall-Muster. Sind P_1 und P_2 Intervall-Muster und ist $r \in \{before, equal, meets, overlaps, during, starts, finishes\}$, dann ist $(P_1 r P_2)$ ein Intervall-

Muster.

Die Größe eines Intervall-Musters P ist die Anzahl der in ihm vorkommenden Intervall-Markierungen. Die Suche nach häufigen Intervall-Mustern ist in ([13]) auf Intervall-Muster der folgenden Form eingeschränkt:

$$P = (((m_1 r_1 m_2) r_2 m_3) \dots) r_{l-1} m_l \quad (*)$$

wobei $m_i \in M$ und $r_i \in \{\textit{before, equal, meets, overlaps, during, starts, finishes}\}$ gilt.

Die Menge aller Intervall-Muster der obigen Form wird durch \mathcal{P}^* gekennzeichnet. Ein Grund für die Einschränkung auf die Intervall-Muster der Form (*) ist, dass in ([13]) vermutet wird, dass Intervall-Muster dieser Form die kausalen Beziehungen, die in den analysierten Daten stecken könnten, gut darstellen können. Weitere Gründe für diese Einschränkung sind Rechenzeit und Speicherbedarf, die bei allgemeineren Formen sehr rasch ansteigen würden.

Mithilfe der zu definierenden Abbildung $map : \mathcal{P}^* \times \mathcal{IS} \rightarrow 2^{\mathcal{I}}$ kann festgestellt werden, ob ein Intervall-Muster $P \in \mathcal{P}^*$ in einer Intervall-Sequenz $S \in \mathcal{IS}$ vorkommt. Dabei ist \mathcal{IS} die Menge aller Intervall-Sequenzen. Zur Definition der Abbildung map ist zwischen zwei Fällen zu unterscheiden.

Im ersten Fall hat das Intervall-Muster die Form $P = m$, wobei $m \in M$ ist: Enthält die Sequenz S kein Intervall mit der Markierung m , dann ist $map(P, S) = \emptyset$. Enthält die Sequenz S ein Intervall X mit der Markierung m , dann ist $map(P, S) = \{X\}$. Hierbei wird $map(P, S)$ zwei Werte $map(P, S) \cdot t_s = X \cdot t_s$ und $map(P, S) \cdot t_e = X \cdot t_e$ zugeordnet. Im zweiten Fall hat das Intervall-Muster die Form $P = (P' r m)$, wobei $P' \in \mathcal{P}^*$ und $m \in M$ ist: Ist $map(P', S) \neq \emptyset$ und enthält S ein Intervall $X \notin map(P', S)$ mit der Markierung m , für das $Y r X$ gilt, wobei Y ein imaginäres Intervall mit dem Anfangszeitpunkt $map(P', S) \cdot t_s$ und dem Endzeitpunkt $map(P', S) \cdot t_e$ ist, dann ist $map(P, S) = map(P', S) \cup \{X\}$. Hierbei sind $map(P, S) \cdot t_s = \min\{map(P', S) \cdot t_s, X \cdot t_s\}$ und $map(P, S) \cdot t_e = X \cdot t_e$ zu definieren. In allen anderen Unterfällen ist $map(P, S) = \emptyset$. Für ein gegebenes $win \in \mathbb{N} \setminus \{0\}$ kommt ein Intervall-Muster $P \in \mathcal{P}^*$ in einer Intervall-Sequenz $S \in \mathcal{IS}$ genau dann vor, wenn $map(P, S) \neq \emptyset$ und $(map(P, S) \cdot t_e - map(P, S) \cdot t_s) \leq win$ gilt. Durch den Parameter win ist explizit verlangt, dass das Vorkommen eines Intervall-Musters in einer Intervall-Sequenz in einer bestimmten Zeitspanne stattfinden muss.

In Bezug auf ein gegebenes $s \in \mathbb{N}$ und ein gegebenes $win \in \mathbb{N} \setminus \{0\}$ ist ein Intervall-Sequenz $P \in \mathcal{P}^*$ häufig in \mathcal{IS} genau dann, wenn die Anzahl der Intervall-Sequenzen, in denen P in der Zeitspanne win vorkommt, größer als der Wert s ist.

3.3.2 Häufige Intervall-Muster nach [14]

Eine Intervall-Sequenz S ist hier auch eine Folge von zeitlichen Intervallen, die aber nach ihrer Anfangszeitpunkten aufsteigend geordnet sind:

$$S = [(m_1, t_{s1}, t_{e1}), \dots, (m_n, t_{sn}, t_{en})] \text{ mit } t_{si} \leq t_{si+1} \text{ für alle } 1 \leq i \leq n - 1.$$

Falls $t_{si} = t_{si+1}$ für zwei direkt aufeinanderfolgende Intervalle in S gilt, dann liegt m_i lexikographisch vor m_{i+1} . Also unterliegen die Intervall-Markierungen einer lexikographischen Ordnung. Eine weitere Anforderung an die Intervall-Sequenz S ist die Maximalitätsbedingung, die besagt, dass sich keine zwei Intervalle mit derselben Markierung

überlappen dürfen:

$$\forall (m_i, t_{si}, t_{ei}), (m_j, t_{sj}, t_{ej}) \in S : i < j \wedge t_{ei} \leq t_{sj} \Rightarrow m_i \neq m_j$$

Wird diese Bedingung durch zwei Intervalle verletzt, dann werden diese einfach zu einem Intervall $(m, \min\{t_{si}, t_{sj}\}, \max\{t_{ei}, t_{ej}\})$ zusammengefasst. Der Grund für die Maximalitätsbedingung ist, dass man in ([14]) davon ausgegangen ist, dass die Intervall-Sequenz eine qualitative Beschreibung einer Zeitreihe darstellt. Die Intervalle stellen also Trends wie „fallend“ oder „steigend“ oder „konstant“ oder andere relevante Merkmale einer Zeitreihe dar. Bei einer solchen qualitativen Beschreibung einer Zeitreihe macht es keinen Sinn, dass sich zwei Intervalle überlappen, die dieselbe Eigenschaft haben. Zur Definition des Begriffes „Intervall-Muster“ nach ([14]) bezeichnen wir durch \mathcal{R} die Menge aller Intervall-Relationen (s. Tabelle (3.2)):

$$\mathcal{R} = \{before, equal, meets, overlaps, during, starts, finishes, after, met - by, overlapped - by, contains, started - by, finished - by\}$$

Ein Paar $P = (map, R)$ heißt zeitliches Intervall-Muster der Größe n genau dann, wenn Folgendes gilt:

1. $map : \{1, \dots, n\} \rightarrow M$
2. R ist eine $n \times n$ -Matrix, wobei $\forall i, j \in \{1, \dots, n\} : R[i, j] \in \mathcal{R}$
3. Es existiert $I_P = \{[m, t_s, t_e] \mid \exists i \in \{1, \dots, n\} : map(i) = m\} \subseteq \mathcal{I}$ mit $|I_P| = n$, wobei die Relationen zwischen allen Intervallen in I_P durch die Matrix R beschrieben sind. D.h:
 $\forall X = [map(i), t_{si}, t_{ei}], Y = [map(j), t_{sj}, t_{ej}] \in I_P : X R[i, j] Y$ ist wahr.

Die Abbildung (3.4) stellt ein Intervall-Muster der Größe $n = 3$ im Sinne der obigen Definition dar.

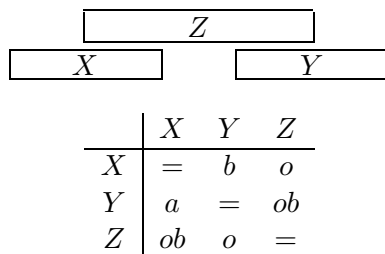


Abbildung 3.4: Ein Intervall-Muster nach ([14]). (b = before, a = after, o = overlaps, ob = overlapped-by)

Die Größe eines Intervall-Musters P wird durch $dim(P)$ gekennzeichnet. Ein Intervall-Muster $P = (map, R)$ kommt in einer Intervall-Sequenz S genau dann vor, wenn $I_P \subseteq S$ gilt. Da es mehr als eine Menge I_P in \mathcal{I} geben kann, die im Teil (3) der obigen Definition gestellten Anforderungen erfüllt, bezeichnen wir die Menge aller solchen Mengen durch

\mathcal{I}_P . Zur Definition des Supports eines Intervall-Muster P sind jeder $I_P \in \mathcal{I}_P$ zwei Werte $I_P \cdot t_s$ und $I_P \cdot t_e$ zuzuordnen, die wie folgt definiert sind:

$$I_P \cdot t_s = \max\{t_s \mid [m, t_s, t_e] \in I_P\}, \quad I_P \cdot t_e = \min\{t_e \mid [m, t_s, t_e] \in I_P\}.$$

Dementsprechend ist ein Intervall-Muster P innerhalb eines Zeitfensters $[t, t + win]$ mit $win \in \mathbb{N} \setminus \{0\}$ beobachtbar genau dann, wenn es eine $I_P \in \mathcal{I}_P$ gibt, die Folgendes erfüllt:

$$I_P \cdot t_s \leq t + win \wedge I_P \cdot t_e \geq t \quad (1)$$

Um die Bedingung (1) zu verstehen bzw. zu interpretieren, betrachten wir ein beliebiges Intervall $X \in I_P$. Es gilt:

$$X \cdot t_s \leq I_P \cdot t_s \leq t + win \text{ und } X \cdot t_e \geq I_P \cdot t_e \geq t.$$

Bildlich bedeutet dies, dass alle in der Menge I_P enthaltenden Intervalle gleichzeitig innerhalb des Fensters gesehen werden können, damit das Intervall-Muster P als beobachtbar innerhalb des Fensters gilt. Dabei wird ein Intervall X als gesehen innerhalb eines Fensters $[t, t + win]$ betrachtet, wenn $t \leq X \cdot t_s \leq t + win$ oder $t \leq X \cdot t_e \leq t + win$ gilt. Also muss es nicht komplett innerhalb des Fensters sein.

Der Support eines Intervall-Musters P in einer Intervall-Sequenz S ist die Summe der gesamten Zeiträume, in denen es innerhalb eines jeweils um eine Zeiteinheit komplett durch die Intervall-Sequenz S zu schiebenden Fensters der Breite $win \in \mathbb{N} \setminus \{0\}$ beobachtet werden kann. Dabei spielt es keine Rolle, wie viele Instanzen vom Intervall-Muster im zu schiebenden Fenster beobachtet werden können. Die Abbildung (3.5) illustriert die Berechnung des Supports eines Intervall-Musters. Ist der Support eines Intervall-Musters größer als ein gegebenes $s \in \mathbb{N}$, dann wird dieses Intervall-Muster als häufig in S bezeichnet.

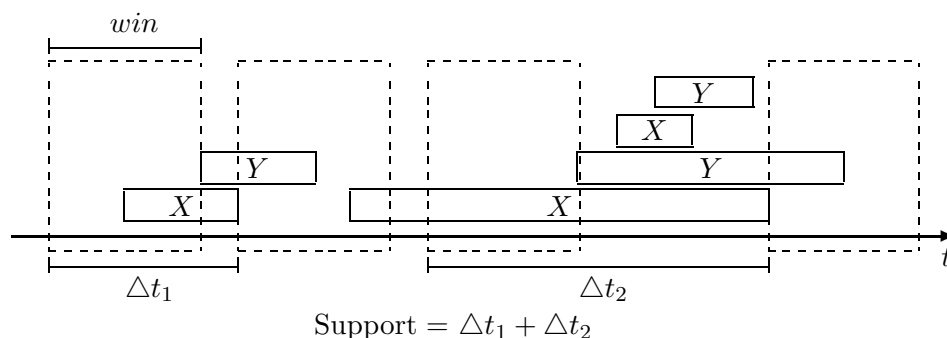


Abbildung 3.5: Illustration der Support-Berechnung eines Intervall-Musters nach ([14])

Um die Suche nach den häufigen Intervall-Mustern einzuschränken, wird die Menge aller Intervall-Muster \mathcal{P} in Äquivalenzklassen aufgeteilt. Danach werden nur die häufigen Intervall-Muster entdeckt, die nicht äquivalent sind. Dies ist zu erreichen, indem man eine Äquivalenzrelation in \mathcal{P} findet bzw. definiert, was im Folgenden bewerkstelligt wird. Es wird zuerst die binäre Relation \sqsubseteq in \mathcal{P} wie folgt definiert: $\forall P = (map, R), P' = (map', R') \in \mathcal{P} : P \sqsubseteq P'$ genau dann, wenn $dim(P) \leq dim(P')$ gilt und eine injektive Funktion $\pi : \{1, \dots, dim(P)\} \rightarrow \{1, \dots, dim(P')\}$ existiert, die Folgendes erfüllt:

$$\forall i, j \in \{1, \dots, \dim(P)\} : R[i, j] = R'[\pi(i), \pi(j)].$$

Die Relation \sqsubseteq ist reflexiv und transitiv in \mathcal{P} aber nicht antisymmetrisch. Also ist sie keine Ordnungsrelation. Basierend auf \sqsubseteq wird die Äquivalenzrelation \equiv in \mathcal{P} wie folgt definiert: $\forall P, P' \in \mathcal{P} : P \equiv P'$ genau dann, wenn $P \sqsubseteq P' \wedge P' \sqsubseteq P$ gilt. Die Relation \sqsubseteq ist jetzt eine Ordnungsrelation in $\mathcal{NP} \subseteq \mathcal{P}$, wobei die Menge \mathcal{NP} Folgendes erfüllt: $\forall P_1, P_2 \in \mathcal{NP} : P_1 \not\equiv P_2$. In ([14]) werden also alle häufigen Intervall-Muster aus \mathcal{NP} entdeckt.

Um den wesentlichen Unterschied zwischen den beiden Auffassungen des Konzeptes „Intervall-Muster“ in ([13]) und ([14]) hervorzuheben, betrachten wir die Abbildung (3.6). Während die in dieser Abbildung gezeigten Intervall-Muster nach ([14]) drei verschiedene Instanzen von drei verschiedenen Mustern repräsentieren, stellen sie nach ([13]) eine Instanz des selben Musters ($((X \text{ overlaps } Y) \text{ before } Z) \text{ overlaps } W$) dar. Der Grund dafür liegt darin, dass ein Intervall-Muster in ([13]) so aufgebaut ist, dass immer ein weiteres Intervall hinten an ein bestehendes Muster angefügt wird, wobei das vorherige Muster als ein großes Intervall angesehen wird (s. Formel (*) auf der Seite 38). Das neue Intervall wird somit zu diesem imaginären Intervall in Relation gesetzt, was die Mehrdeutigkeit des Musters verursacht.

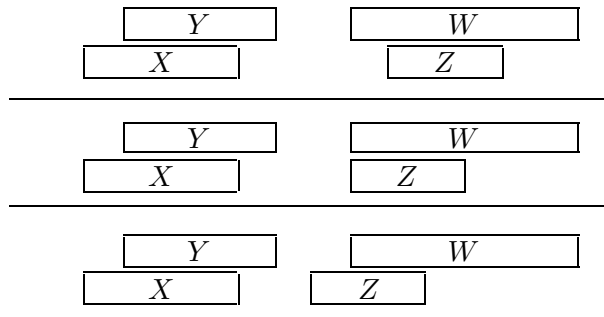


Abbildung 3.6: Diese drei Muster stellen nach([13]) eine Instanz des selben Intervall-Musters ($((X \text{ overlaps } Y) \text{ before } Z) \text{ overlaps } W$), während sie nach ([14]) drei verschiedene Instanzen von drei verschiedenen Intervall-Mustern sind.

4 Neuer Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen

Im Abschnitt (3.1.1) wurden die Konzepte „Ereignis-Sequenz“, „Episode“ und „häufige Episode“ nach ([9]) formuliert. Entsprechend wurde das Problem der Entdeckung häufiger Episoden in einer Ereignis-Sequenz vorgestellt und der WINEPI-Algorithmus zur Lösung dieses Problems ausführlich im Abschnitt (3.1.2) beschrieben. Dort haben wir erfahren, dass dieser Algorithmus ein iterativer Prozess ist. In jedem Iterationschritt dieses Prozesses wird eine Kollektion von Episoden generiert und danach werden die häufigen Episoden in der Kollektion durch den Durchlauf der Ereignis-Sequenz festgestellt und ausgewählt. Die Episoden jeder in einem Iterationschritt generierten Kollektion heißen Kandidaten und haben zwei Eigenschaften. Die eine ist, dass sie die selbe Länge haben und die andere ist, dass alle ihren Unterepisoden häufig sind. Also folgt der WINEPI-Algorithmus dem Prinzip der Kandidaten-Generierung, auf dem auch die Arbeitsweise des Apriori-Algorithmus zur Entdeckung häufiger Mengen in einer Transaktionendatenbank basiert, was im Abschnitt (2.2.1) gezeigt wurde. Die Rechtfertigung für die Anwendung des Prinzips der Kandidaten-Generierung bei WINEPI und Apriori liegt in der Tatsache, dass die Menge der häufigen Episoden bzw. die Menge der häufigen Mengen in Bezug auf Unterepisoden-Beziehung bzw. in Bezug auf Teilmengen-Beziehung abwärts abgeschlossen ist. Dies bedeutet nach dem Lemma (3.3), dass jede Unterepisode einer häufigen Episode häufig sein muss, bzw. nach dem Lemma (2.2), dass jede Teilmenge einer häufigen Menge häufig sein muss. Durch eine kompakte Darstellung der Transaktionendatenbank und eine effiziente Ausnutzung der Eigenschaften dieser Darstellung konnte der FP-growth-Algorithmus das Problem der Entdeckung häufiger Mengen in Transaktionendatenbanken lösen, ohne das Prinzip der Kandidaten-Generierung anzuwenden. Dadurch hat der FP-growth-Algorithmus eine bessere Laufzeit als Apriori-Algorithmus erreicht ([6], [7]). Also konnte das Problem der Entdeckung häufiger Mengen durch das Verzichten auf die Kandidaten-Generierung schneller gelöst werden. Dies ist eine Motivation für die Frage, ob das Problem der Entdeckung häufiger Episoden in Ereignis-Sequenzen ohne die Anwendung des Prinzips der Kandidaten-Generierung gelöst werden kann. In der Tat liegt die Untersuchung dieser Frage im Rahmen dieser Diplomarbeit.

In diesem Kapitel wird ein neuer Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen entwickelt. Basierend auf dem Studieren des Zusammenhanges zwischen häufigen Mengen und häufigen Episoden und auf der Trennung der zeitlichen von den symbolischen Informationen, die jeweils in den Episoden und in den Zeitfenstern enthalten sind, werden im Abschnitt (4.2) die Datenstrukturen des Algorithmus entworfen. Auf der Basis der Eigenschaften dieser Datenstrukturen wird im Abschnitt(4.3) der

Algorithmus formuliert. Zum allgemeinen Verständnis ist im Abschnitt (4.1) ein Überblick über die Arbeitsweise des Algorithmus gegeben. Für die Illustrationen im Laufe dieses Kapitels wird oft ein Bezug auf die folgende Ereignis-Sequenz genommen:

$$\mathcal{S} = ([(A, 1), (C, 2), (B, 3), (D, 3), (A, 7), (A, 8), (B, 9)], 1, 10)$$

Diese Ereignis-Sequenz ist in der Abbildung (4.1) graphisch dargestellt.

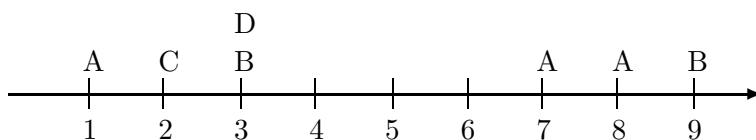


Abbildung 4.1: Eine Ereignis-Sequenz \mathcal{S} mit $T_s = 1$ und $T_e = 10$ über $E = \{A, B, C, D\}$

4.1 Überblick

Im Gegensatz zum WINEPI-Algorithmus wird die Menge $WIN(\mathcal{S}, win)$, also die Menge aller Zeitfenster der Breite win , explizit berechnet. In einer speziellen Tabelle namens WT werden für jedes Zeitfenster $\mathcal{W} \in WIN(\mathcal{S}, win)$ alle in ihm vorkommenden Ereignis-Typen als Menge gespeichert. Jedem Ereignis-Typ $e \in E$ wird eine Tabelle namens $e \cdot T$ zugeordnet, wobei jede Zeile dieser Tabelle einem Fenster entspricht. In jeder Zeile der Tabelle $e \cdot T$ werden alle Zeitpunkte gespeichert, an denen der Ereignis-Typ e in dem dieser Zeile entsprechenden Fenster vorkommt. Die Tabellen WT und $e \cdot T$ sind zusammen eine Repräsentation der Menge $WIN(\mathcal{S}, win)$ und dienen dazu, die zeitlichen Informationen in $WIN(\mathcal{S}, win)$ von der symbolischen Informationen ohne Verlust zu trennen. Die Tabelle WT wird als Transaktionsdatenbank betrachtet und die Menge $\mathcal{F}(WT, s)$, also die Menge aller häufigen Mengen in WT mit einem Support größer als s , wird mit Hilfe von **FP-growth** generiert. Handelt es sich um die Entdeckung der häufigen parallelen Episoden in \mathcal{S} , dann entspricht $\mathcal{F}(WT, s)$ genau der Menge aller solchen Episoden. Es ist dabei zu erkennen, dass das Problem der Entdeckung häufiger paralleler Episoden dadurch auf das Problem der Entdeckung häufiger Mengen zurückgeführt wurde. Da **FP-growth** die Menge $\mathcal{F}(WT, s)$ ohne Kandidaten-Generierung erzeugt, ist also das Problem der Entdeckung der häufigen parallelen Episoden ohne Kandidaten-Generierung gelöst. Der Kern der Arbeit liegt also in der Berechnung der Menge aller häufigen seriellen Episoden, weil sich die Entdeckung der häufigen generellen Episoden auf die Entdeckung der parallelen und seriellen Episoden zurückführen lässt ([9]).

Der Rest dieses Abschnittes stellt die zentralen Ideen für die Entwicklung einer Lösung für die Entdeckung aller häufigen seriellen Episoden ohne Kandidaten-Generierung vor. Eine der Ideen ist die Ausnutzung der folgenden Tatsache: Ist $\alpha = (V, R, f)$ eine häufige Episode, dann muss die Knotenmenge V eine häufige Menge in WT sein. Also ist die

Häufigkeit der Knotenmenge einer Episode in WT eine notwendige aber keine hinreichende Bedingung, damit die Episode häufig in der Ereignis-Sequenz vorkommen kann. Mit anderen Worten ist diese Idee wie folgt zu formulieren: Ist $\alpha = (V, R, f)$ häufig in \mathcal{S} , dann ist $V \in \mathcal{F}(WT, s)$. Ein anderer Schlüssel der Lösung ist folgendes: Kommen die Ereignis-Typen einer häufigen Menge V aus $\mathcal{F}(WT, s)$ in einer bestimmten Reihenfolge, die in der Ordnungsrelation R auszudrücken ist, mehr als s -Mal in der Ereignis-Sequenz \mathcal{S} vor, dann ist (V, R, f) eine häufige Episode in \mathcal{S} . Dies bedeutet, dass aus einer häufigen Menge $V \in \mathcal{F}(WT, s)$ mehrere häufige Episoden konstruiert werden können. Nun stellt sich die Frage, wie man aus einer häufigen Menge $V \in \mathcal{F}(WT, s)$ alle möglichen häufigen seriellen Episoden gewinnen kann. Für die Antwort auf diese Frage kommen die Datenstrukturen bzw. die Tabellen $e \cdot T$ ($e \in E$) und das folgende Verfahren ins Spiel. Bezeichnen wir für eine häufige Menge $V \in \mathcal{F}(WT, s)$ die Menge aller häufigen Episoden (V, \cdot, \cdot) , deren Knotenmenge die Menge V ist, mit $V \cdot EP$, dann werden wir für jede häufige Menge $V \cup \{e\} \in \mathcal{F}(WT, s)$ mit $e \notin V$ alle möglichen häufigen Episoden $(V \cup \{e\}, \cdot, \cdot)$ aus der Menge $V \cdot EP$ und dem Ereignis-Typ e generieren. Danach werden wir für jede häufige Menge $(V \cup \{e, e'\}) \in \mathcal{F}(WT, s)$ mit $e' \notin V \cup \{e\}$ alle möglichen häufigen Episoden $(V \cup \{e, e'\}, \cdot, \cdot)$ aus der Menge $(V \cup \{e\}) \cdot EP$ und dem Ereignis-Typ e' generieren. Dies wird für alle $e'' \in E \setminus (V \cup \{e, e'\})$ mit $(V \cup \{e, e'\}) \cup \{e''\} \in \mathcal{F}(WT, s)$ fortgesetzt. Dieses Verfahren beinhaltet die Erzeugung aller möglichen in der Ereignis-Sequenz vorkommenden Episoden aus einer in der Ereignis-Sequenz vorkommenden Episode und einem Ereignis-Typ. Um aus einer Episode $\beta = (V_\beta, R_\beta, \cdot)$ und einem Ereignis-Typ $e \notin V_\beta$ eine neue Episode $\alpha = (V_\beta \cup \{e\}, R_\alpha, \cdot)$ erzeugen zu können, wird eine neue Datenstruktur für die Repräsentation der Episoden entworfen. Jeder Episode $\beta = (V_\beta, R_\beta, \cdot)$ werden zwei Komponente $\beta \cdot L$ und $\beta \cdot T$ zugeordnet. Die erste Komponente $\beta \cdot L$ ist eine Zeichenkette aus allen Symbolen aus V_β , die die Ordnungsrelation R_β widerspiegelt. Die zweite Komponente $\beta \cdot T$ ist eine Tabelle, wobei jede Zeile einem Fenster entspricht, in dem die Episode β vorkommt. Jede Zeile in $\beta \cdot T$ enthält alle Zeitpunkte, in denen die Ereignis-Typen von β in dem entsprechenden Fenster vorkommen. Die Zeitpunkte innerhalb jeder Zeile sind so gruppiert, dass jede Gruppe von Zeitpunkten das Vorkommen der Episode in dem entsprechenden Fenster widerspiegelt. Zusammenfassend werden also aus einer Episode $\beta = (V_\beta, R_\beta, \cdot)$ und einem Ereignis-Typ $e \notin V_\beta$ durch die Definition von bestimmten Operationen auf $\beta \cdot T$ und $e \cdot T$ alle möglichen häufigen Episoden $\alpha = (V_\beta \cup \{e\}, R_\alpha, \cdot)$ erzeugt.

4.2 Datenstrukturen des Algorithmus

In diesem Abschnitt werden die dem neuen Algorithmus zugrundeliegenden Datenstrukturen entwickelt. Der Entwurf dieser Datenstrukturen stützt sich auf zwei Säulen. Die erste ist die Trennung der zeitlichen von den symbolischen Informationen, die jeweils in den Zeitfenstern und in den Episoden gespeichert sind. Die zweite ist das Studieren der Beziehungen zwischen häufigen Episoden und häufigen Mengen.

4.2.1 Datenstrukturen zur Repräsentation der Menge $WIN(\mathcal{S}, win)$

Zuerst wird jedes Zeitfenster \mathcal{W} in der Menge aller Zeitfenster der Breite win $WIN(\mathcal{S}, win)$ mit einer eindeutigen Nummer $id \in \mathbb{N} \cap [1, T_e - T_s + win - 1]$ identifiziert. Die Identifikationsnummern sind die ganzen Zahlen aus dem Intervall $[1, T_e - T_s + win - 1]$, weil nach dem Lemma (3.1) die Anzahl aller Zeitfenster der Breite win gleich $T_e - T_s + win - 1$ ist. Die Identifikationsnummer eines Fenster \mathcal{W} wird durch $Id(\mathcal{W})$ gekennzeichnet. Zur Illustration sind in der Tabelle (4.1) alle Fenster der Breite $win = 4$ über der im Abbildung (4.1) dargestellten Ereignis-Sequenz mit deren Identifikationsnummern aufgelistet.

id	Fenster
1	$([(A, 1)], -2, 2)$
2	$([(A, 1), (C, 2)], -1, 3)$
3	$([(A, 1), (C, 2), (B, 3), (D, 3)], 0, 4)$
4	$([(A, 1), (C, 2), (B, 3), (D, 3)], 1, 5)$
5	$([(C, 2), (B, 3), (D, 3)], 2, 6)$
6	$([(B, 3), (D, 3)], 3, 7)$
7	$([(A, 7)], 4, 8)$
8	$([(A, 7), (A, 8)], 5, 9)$
9	$([(A, 7), (A, 8), (B, 9)], 6, 10)$
10	$([(A, 7), (A, 8), (B, 9)], 7, 11)$
11	$([(A, 8), (B, 9)], 8, 12)$
12	$([(B, 9)], 9, 13)$

Tabelle 4.1: Die Menge aller Fenster der Breite $win = 4$ über der im Abbildung (4.1) dargestellten Ereignis-Sequenz mit deren Identifikationsnummern

Danach wird die Menge $WIN(\mathcal{S}, win)$ durch die Tabellen $WT, e \cdot T$ ($\forall e \in E$) dargestellt, die wie folgt zu definieren sind.

Die Tabelle WT heißt Fenster-Tabelle und hat die Größe $|WIN(\mathcal{S}, win)|$. Der Inhalt der Zeile mit der Nummer $n \in \mathbb{N} \cap [1, T_e - T_s + win - 1]$ wird durch $WT[n]$ gekennzeichnet und für das Fenster $\mathcal{W} = (W, T_{ws}, T_{we})$ mit $Id(\mathcal{W}) = n$ wie folgt definiert:

$$WT[n] = \{e, e_{-1}, \dots, e_k \mid e \in E \text{ und } e \text{ kommt genau } (k+1)\text{-Mal in } W \text{ vor}\}.$$

Handelt es sich im besonderen Fall um die Entdeckung häufiger injektiver Episoden, dann wird die Definition von $WT[n]$ auf die folgende Menge eingeschränkt:

$$WT[n] = \{e \in E \mid \exists t \in \mathbb{Z} : (e, t) \in W\}$$

Betrachtet man zum Beispiel das Fenster $\mathcal{W} = ((F, 2), (G, 5), (F, 5), (F, 7)), \cdot, \cdot)$ mit $Id(\mathcal{W}) = n$, dann ist $WT[n]$ entweder die Menge $\{F, G\}$, falls es sich um injektive Episoden handelt, oder die Menge $\{F, F_{-1}, F_{-2}, G\}$ im allgemeinen Fall. Die Tabelle (4.2) illustriert die Definition der Tabelle WT .

Um die Repräsentation der Menge $WIN(\mathcal{S}, win)$ zu vervollständigen, wird jedem Ereignis-

id	$WT[id]$	id	$WT[id]$
1	{A}	1	{A}
2	{A, C}	2	{A, C}
3	{A, C, B, D}	3	{A, C, B, D}
4	{A, C, B, D}	4	{A, C, B, D}
5	{C, B, D}	5	{C, B, D}
6	{B, D}	6	{B, D}
7	{A}	7	{A}
8	{A, A_1}	8	{A}
9	{A, A_1, B}	9	{A, B}
10	{A, A_1, B}	10	{A, B}
11	{A, B}	11	{A, B}
12	{B}	12	{B}

Tabelle 4.2: Die linke Tabelle repräsentiert die Tabelle WT der im Tabelle (4.1) dargestellten Menge $WIN(\mathcal{S}, 4)$. Die rechte Tabelle stellt auch die Tabelle WT der im Tabelle (4.1) dargestellten Menge $WIN(\mathcal{S}, 4)$ dar, nur geht es hier um Entdeckung injektiver Episoden.

Typ $e \in E$ eine Tabelle $e \cdot T$ zugeordnet. Jede Zeile wird hier mit einem Fenster assoziiert, in dem der Ereignis-Typ e vorkommt, und erhält als Nummer die Identifikationsnummer dieses Fensters. Also ist die Anzahl der Zeilen von $e \cdot T$ gleich der Anzahl der Zeitfenster, in denen der Ereignis-Typ e vorkommt. Der Inhalt der Zeile mit der Nummer n ($1 \leq n \leq |WIN(\mathcal{S}, win)|$) wird durch $e \cdot T[n]$ gekennzeichnet und als die Liste $e \cdot T[n] = [t_1, t_2, \dots, t_k]$ mit $t_i \leq t_{i+1}$ für alle $1 \leq i < k$ definiert, wobei t_i 's ($1 \leq i \leq k$) alle Zeitpunkte des Vorkommens von e im mit der Nummer n identifizierten Fenster $\mathcal{W} = (W, \cdot, \cdot)$ und in derjenigen Reihenfolge geordnet sind, in der die Ereignisse (e, t_i) in W vorkommen. Zur Illustration der Definition der Datenstruktur $e \cdot T$ ist $A \cdot T$ in der Tabelle (4.3) dargestellt.

In **Algorithm (6)** ist eine Prozedur namens „createTables“ zur Erzeugung der Tabellen WT und $e \cdot T$ ($\forall e \in E$) formuliert. Zur fortlaufenden Zählung des Vorkommens eines Ereignis-Typ e in einem Zeitfenster \mathcal{W} wird extra für diese Prozedur eine $|E| \times |WT|$ -Matrix namens $Count$ angelegt. Vor dem Aufruf der Prozedur „createTables“ ist $Count[e, i]$ mit dem Wert „0“ für jedes $e \in E$ und jedes $\mathcal{W} \in WIN(\mathcal{S}, win)$ mit $Id(\mathcal{W}) = i$ zu initialisieren. Ebenso sind die Tabellen WT und $e \cdot T$ vor dem Aufruf der Prozedur zu deklarieren. Kommt der Ereignis-Typ e im mit der Nummer i identifizierten Zeitfenster \mathcal{W} zum ersten Mal vor, dann wird der Wert $Count[e, i]$ nicht geändert, also bleibt er gleich „0“. Für jedes weitere Vorkommen von e in \mathcal{W} wird der Wert $Count[e, i]$ um eines erhöht (Zeile 9). Das heißt, wird e bis jetzt k Mal in \mathcal{W} betrachtet, dann ist $Count[e, i] = k - 1$. Die Einträge $Count[e, Id(\mathcal{W})]$ der Matrix $Count$ dienen zur Produktion der Suffixes „- k “ mit $1 \leq k$, die bei der Konstruktion von $WT[Id(\mathcal{W})]$ den

id	$A \cdot T[id]$
1	[1]
2	[1]
3	[1]
4	[1]
7	[7]
8	[7, 8]
9	[7, 8]
10	[7, 8]
11	[8]

Tabelle 4.3: Diese Tabelle stellt die Tabelle $A \cdot T$ des Ereignis-Typs A dar. Die aufgelisteten Identifikationsnummern bestimmen die Fenster der in der Tabelle (4.1) dargestellten Menge $WIN(\mathcal{S}, 4)$, in denen der Ereignis-Typ A vorkommt.

jeweiligen Ereignis-Typ e angehängt werden (Zeilen 10 und 11).

Handelt sich um die Entdeckung der häufigen injektiven Episoden in \mathcal{S} , dann braucht man die Matrix *Count* nicht mehr und deshalb ist der „**else**“-Block (Zeilen 8 - 11) aus der Formulierung der Prozedur „**createTables**“ zu entfernen.

Die Anweisungen innerhalb des Rumpfes der „**for all**“-Schleife (Zeile 4) sind in konstanter Zeit durchführbar. Deshalb kostet die Ausführung dieser Schleife $\mathcal{O}(|E|)$ Laufzeit. Dabei wird hier angenommen, dass ein Ereignis-Typ $e \in E$ maximal ein Mal an einem Zeitpunkt vorkommen kann. Dementsprechend ist die Laufzeit der Prozedur „**createTables**“ $\mathcal{O}(win \times |E| \times (T_e - T_s + win))$. Hier haben wir die Laufzeit dieser Prozedur berechnet, um das Lemma (4.4) im Abschnitt (4.3) formulieren zu können.

Es stellt sich nun die Frage, ob die Datenstrukturen WT und $e \cdot T$ ($\forall e \in E$) verlustfrei die Menge $WIN(\mathcal{S}, win)$ repräsentieren. Das folgende Lemma gibt eine positive Antwort auf diese Frage, wenn man den Anfang T_{ws} und das Ende T_{we} jedes Fensters außer Acht liebt. Dies ist akzeptabel, weil die Werte T_{ws} und T_{we} jedes Fensters bei Feststellung des Vorkommens einer Episode im entsprechenden Fenster überhaupt keine Rolle spielen.

Lemma 4.1 *Die Datenstrukturen WT und $e \cdot T$ ($\forall e \in E$) sind zusammen eine verlustfreie Repräsentation der Menge $WIN(\mathcal{S}, win)$.*

Beweis: Sei $\mathcal{W} = (W, \cdot, \cdot)$ mit $W = [(e_1, t_1), \dots, (e_k, t_k)]$ ein beliebiges Zeitfenster aus $WIN(\mathcal{S}, win)$. Es wird gezeigt, dass sich W aus den Tabellen $WT, e_1 \cdot T, e_2 \cdot T, \dots, e_k \cdot T$ rekonstruieren lässt.

Erstens sind alle Ereignis-Typen e_1, e_2, \dots, e_k nach der Definition von WT in $WT[Id(\mathcal{W})]$ enthalten. Zweitens enthält $e_i \cdot T[Id(\mathcal{W})]$ ($1 \leq i \leq k$) nach der Definition von $e_i \cdot T$ genau nur die Zeitpunkte t_{i_j} ($1 \leq j \leq k$), für die $(e_i, t_{i_j}) \in W$ gilt. Deshalb lässt sich

jedes Fenster mit der Identifikationsnummer id durch die in **Algorithm (7)** formulierte Prozedur rekonstruieren. \square

Algorithm 6 Eine Prozedur zur Erzeugung der Tabellen WT und $e \cdot T$.

```

1: procedure CREATETABLES( $\mathcal{S}, WT, Count, win$ )
2:   for  $i \leftarrow 1, T_e - T_s + win - 1$  do
3:     for  $j \leftarrow i, win + i - 1$  do
4:       for all  $(e, j) \in \mathcal{S}$  do
5:          $e \cdot T[i] \leftarrow e \cdot T[i] + [j]$ 
6:         if  $e \notin WT[i]$  then
7:            $WT[i] \leftarrow WT[i] + [e]$ 
8:         else
9:            $Count[e, i] \leftarrow Count[e, i] + 1$ 
10:           $suffix \leftarrow \text{„-“} + Count[e, i]$ 
11:           $WT[i] \leftarrow WT[i] + [e + suffix]$ 
12:        end if
13:      end for
14:    end for
15:  end for
16: end procedure

```

Algorithm 7 Eine Prozedur zur Rekonstruktion eines Fensters mit der Nummer id aus den Datenstrukturen WT und $e \cdot T$

```

1: function REKONSTRUIERE( $id, E, WT$ )
2:    $W \leftarrow []$ 
3:   for all  $e \in E \cap WT[id]$  do
4:      $[t_1, \dots, t_k] \leftarrow e \cdot T[id]$ 
5:      $W_e \leftarrow []$ 
6:     for  $i \leftarrow 1, k$  do
7:        $W_e \leftarrow W_e + [(e, t_i)]$ 
8:     end for
9:      $W \leftarrow W + W_e$ 
10:  end for
11:  sort( $W$ ) ▷ Aufsteigende Sortierung nach  $t$ 's Werten
12:  return  $W$ 
13: end function

```

4.2.2 Datenstrukturen zur Repräsentation einer Episode

Bevor wir Datenstrukturen zur Repräsentation einer Episode konzipieren, betrachten wir die Episode $\alpha = (\{A, B\}, \{(A, A), (B, B), (A, B)\}, fid)$ und die Zeitfenster $\mathcal{W} = ((A, 7), (A, 8), (B, 9), 7, 11)$. Da A vor B bezüglich der Ordnungsrelation von α liegt und es Zeitpunkte $t_1 = 7$ und $t_3 = 9$ gibt, für die $t_1 < t_3$ und $(A, t_1), (B, t_3) \in \mathcal{W}$

gelten, erfüllt die Funktion $h_1 : \{A, B\} \rightarrow \{1, 2, 3\}$ mit $h_1(A) = 1$ und $h_1(B) = 3$ die Bedingungen der Definition (3.4). Also kommt α in \mathcal{W} vor. Auch bezeugt die Funktion $h_2 : \{A, B\} \rightarrow \{1, 2, 3\}$ mit $h_2(A) = 2$ und $h_2(B) = 3$ das Vorkommen von α in \mathcal{W} , weil es einen anderen Zeitpunkt $t_2 = 8$ mit $t_2 < t_3$ und $(A, t_2) \in \mathcal{W}$ gibt. Dies bedeutet, dass für eine Episode α mehr als eine Funktion h geben kann, die ihr Vorkommen in einer Ereignis-Sequenz bzw. in einem Fenster beweisen kann. Diese Tatsache führt uns zu der folgenden Definition.

Definition 4.1 (Instanz des Vorkommens) Sei $\alpha = (V, R, f)$ eine serielle Episode, die in einer Ereignis-Sequenz $\mathcal{S} = (S, T_s, T_e)$ vorkommt. Dann existiert eine injektive Funktion $h : V \rightarrow \{1, \dots, n\}$ mit den in der Definition (3.4) angegebenen Eigenschaften. Sobald man die Knoten $x \in V$ so indiziert hat, dass $\forall x_i, x_j \in V : i < j \Leftrightarrow (x_i, x_j) \in R$ gilt, dann heißt die Liste $[t_{h(x_1)}, t_{h(x_2)}, \dots, t_{h(x_{|V|})}]$ Instanz des Vorkommen von α in \mathcal{S} .

Die Menge aller Instanzen des Vorkommen von α in \mathcal{S} wird durch $Ins(\alpha, \mathcal{S})$ gekennzeichnet. Betrachten wir zum Beispiel die Episode α und die Zeitfenster \mathcal{W} , die als einleitendes Beispiel für diesen Unterabschnitt angegeben sind, dann ist $Ins(\alpha, \mathcal{W}) = \{[7, 9], [8, 9]\}$. Die Instanz $[t_1, t_2, \dots, t_{|V|}]$ des Vorkommen einer Episode $\alpha = (V, R, f)$ in einer Ereignis-Sequenz $\mathcal{S} = (S, \cdot, \cdot)$ hat folgende Eigenschaften, die direkt aus der obigen Definition zu schließen sind:

1. $\forall t_i \in [t_1, t_2, \dots, t_{|V|}] : (f(x_i), t_{h(x_i)}) \in S$
2. $\forall t_i, t_j \in [t_1, t_2, \dots, t_{|V|}] : i < j \Leftrightarrow t_i < t_j$

Basierend auf dem Konzept „Instanz des Vorkommens“ werden die Datenstrukturen zur Darstellung serieller Episoden konzipiert. Einer seriellen Episode $\alpha = (V, R, f)$ wird erstens eine Liste namens $\alpha \cdot L$ zugeordnet. Diese Liste hat die Gestalt $\alpha \cdot L = [x_1, x_2, \dots, x_{|V|}]$ mit der folgenden Eigenschaft:

$$\forall x_i, x_{i+1} \in \alpha \cdot L : (x_i, x_{i+1}) \in R \quad (1 \leq i < |V|).$$

Zweitens wird α eine Tabelle namens $\alpha \cdot T$ zugeordnet. Jede Zeile dieser Tabelle wird mit einem Fenster assoziiert, in dem α vorkommt, und erhält als Nummer die Identifikationsnummer dieses Fensters. Also ist die Anzahl von $\alpha \cdot T$ gleich der Anzahl der Fenster, in denen α vorkommt. Der Inhalt der Zeile mit der Nummer n ist durch $\alpha \cdot T[n]$ bezeichnet und durch $\alpha \cdot T[n] = Ins(\alpha, \mathcal{W})$ definiert, wobei $Id(\mathcal{W}) = n$ ist. Also enthält die Zeile mit der Nummer n alle Instanzen des Vorkommens von α in dem mit n identifizierten Fenster.

Betrachten wir als Beispiel die Episode $\alpha = (\{A, B\}, \{(A, A), (B, B), (A, B)\}, f_{id})$, dann ist $\alpha \cdot L = [A, B]$ und $\alpha \cdot T$ ist in der Tabelle (4.4) dargestellt.

4.2.3 Datenstrukturen zur Repräsentation von $\mathcal{F}(WT, s)$ und $\mathcal{F}(\mathcal{S}, win, s)$

Um diesen Abschnitt formulieren zu können, erinnern wir uns an die Vereinbarung zur Vereinfachung der Notation einer Episode, die wir direkt nach der Definition (3.3) auf der Seite (26) im Kapitel (3) getroffen haben.

id	$\alpha \cdot T[id]$
3	{[1, 3]}
4	{[1, 3]}
9	{[7, 9], [8, 9]}
10	{[7, 9], [8, 9]}
11	{[8, 9]}

Tabelle 4.4: Diese Tabelle repräsentiert $\alpha \cdot T$ der seriellen Episode $\alpha = [A, B]$. Die aufgelisteten Identifikationsnummern bestimmen die Fenster aus der in der Tabelle (4.1) dargestellten Menge $WIN(\mathcal{S}, 4)$, in denen α vorkommt.

Dabei wird für eine Episode $\alpha = (V, R, f)$ jeder Knoten $x \in V$ durch $f(x)$ unter der Bedingung ersetzt, dass für alle $x_1, \dots, x_k \in V$ mit $x_1 \neq \dots \neq x_k \wedge f(x_1) = \dots = f(x_k) = e$ der Ereignis-Typ e k -Mal in V vorkommen muss. Dies bedeutet, dass die Funktion f in dieser Notation implizit gegeben und als die Identitätsfunktion f_{id} zu betrachten ist. Zum Beispiel sieht die Episode $\alpha = (\{x, y, z\}, \{(x, x), (y, y), (z, z), (x, z)\}, f)$, wobei $f : \{x, y, z\} \rightarrow \{F, G\}$ mit $f(x) = f(y) = G$ und $f(z) = F$ ist, in dieser Notation so aus: $\alpha = (\{G, G, F\}, \{(G, G), (F, F), (G, F)\}, f_{id})$.

Der Rest dieses Abschnitt geht davon aus, dass die betrachteten Episoden in der oben vorgestellten Notation ausgedrückt sind.

Lemma 4.2 *Betrachtet man die Tabelle WT als Transaktionsdatenbank, dann gilt für jede Episode $\alpha = (V, R, \cdot)$ folgendes:*

Ist $\alpha \in \mathcal{F}(\mathcal{S}, win, s)$, dann ist $V \in \mathcal{F}(WT, s)$.

Außerdem gilt: $sup(\alpha, \mathcal{S}, win) \leq sup(V, WT)$.

Beweis: Sei $\alpha = (V, R, \cdot)$ eine beliebige häufige Episode in \mathcal{S} . Also ist $sup(\alpha, \mathcal{S}, win) > s$. Dies bedeutet nach der Definition des Supports, dass die Symbole aus V alle zusammen in einer bestimmten durch die Relation R festgelegten Reihenfolge in mehr als s Zeitfenstern vorkommen. Dies bedeutet nach der Definition von WT , dass die Menge V in mehr als s Zeilen der Tabelle WT vorkommt. Dies besagt, dass $V \in \mathcal{F}(WT, s)$. Da die Symbolen aus V alle zusammen in mehr als einer bestimmte Reihenfolge, wie z.B. die durch die Relation R definierte Reihenfolge, in WT vorkommen können, ist $sup(\alpha, \mathcal{S}, win) \leq sup(V, WT)$. \square

Nach dem obigen Lemma ist also die Häufigkeit von V in WT eine notwendige Bedingung für die Häufigkeit von α in \mathcal{S} . Dies ist der Grund für den Entwurf der Datenstruktur WT . Also ist die Menge $\mathcal{F}(WT, s)$ mithilfe eines Algorithmus wie Apriori, Eclat oder FP-growth zu berechnen. Der Grund dafür, dass wir für jeden Ereignis-Typ e , der in einem Fenster \mathcal{W} $(k + 1)$ -Mal mit $k \geq 1$ vorkommt, das zweite Vorkommen als e_1 , das dritte Vorkommen als e_2 , ... und das $(k + 1)$ -te Vorkommen als e_k in $WT[Id(\mathcal{W})]$ gespeichert haben, ist, dem die Menge $\mathcal{F}(WT, s)$ erzeugenden Algorithmus die Betrachtung

von mehreren e 's in einer Menge als verschiedene Symbole zu ermöglichen. Dies führt zur korrekten Berechnung des Supports derjenigen Mengen, die einen Ereignis-Typ mehr als ein Mal enthalten, wie z.B. die Menge $V = \{G, G, F\}$. Natürlich werden nach der Berechnung der Menge $\mathcal{F}(WT, s)$ alle Suffixe der Form „ k “ mit $k \geq 1$ aus denjenigen Ereignis-Typen entfernt, die solche Suffixe haben. Wird z.B. die Menge $\{G, G_1, G_2, F\}$ als häufige Menge in WT entdeckt, dann wird sie als $\{G, G, G, F\}$ in $\mathcal{F}(WT, s)$ eingefügt. Aber es gibt bei diesem Verfahren ein kleines Problem, das im folgenden Beispiel geschildert ist.

Seien $X_1 = \{F_1, F_2, G\}$ und $X_2 = \{F, F_1, G\}$ häufige Mengen in WT mit $\text{sup}(X_1, WT) \neq \text{sup}(X_2, WT)$ gegeben. Beim Einfügen von X_1 und X_2 in $\mathcal{F}(WT, s)$ werden sie identisch mit der Menge $X = \{F, F, G\}$. Deshalb stellt sich die Frage, mit welchem Support die neue entstehende Menge X in $\mathcal{F}(WT, s)$ aufgenommen werden muss. Um die Antwort auf diese Frage zu finden, betrachten wir die den Mengen X_1 und X_2 entsprechende Struktur von WT . Bei der Berechnung von $\text{sup}(X_1, WT)$ müssen alle Zeitfenster gezählt werden, in denen der Ereignis-Typ F mindestens 3 Mal vorkommt, weil nach der Definition von WT jedes Vorkommen von F mit einem Suffix „ k “ das Vorkommen von F ohne Suffix und das Vorkommen von F mit den Suffixen „ k' “ für alle $k' < k$ impliziert. Aber bei der Berechnung von $\text{sup}(X_2, WT)$ müssen alle Zeitfenster gezählt werden, in denen der Ereignis-Typ F mindestens 2 Mal vorkommt, was dem Support von X entspricht. Also muss der Support der Menge X gleich dem Support der Menge X_2 sein. Außerdem gilt nach dieser Argumentation, dass $\text{sup}(X_2, WT) > \text{sup}(X_1, WT)$ ist. Dies lässt sich wie folgt verallgemeinern:

Werden die Mengen X_1, X_2, \dots, X_n nach der Entfernung aller Suffixe aus ihren Elementen identisch zu einer Menge X , dann wird X in $\mathcal{F}(WT, s)$ mit dem Support $\text{sup}(X, WT) = \max_{1 \leq i \leq n} \text{sup}(X_i, WT)$ eingefügt.

Nach dem Lemma (4.2) lässt sich die Häufigkeit der Menge V auf die Häufigkeit der Episode (V, R, \cdot) zurückführen. Analog stellt man sich die Frage, welche Schlussfolgerung aus der Häufigkeit einer Menge V in WT in Bezug auf die Häufigkeit der Episode (V, R, \cdot) gezogen werden kann. Das folgende Beispiel zeigt, dass die Häufigkeit von V in WT keine hinreichende Bedingung für die Häufigkeit von $\alpha = (V, R, \cdot)$ in \mathcal{S} ist.

Beispiel 4.1 Für $\mathcal{S} = (([F, 1), (G, 2), (G, 3), (F, 4)], 1, 5)$ sieht $\text{WIN}(\mathcal{S}, 2)$ wie folgt aus:

$$\text{WIN}(\mathcal{S}, 2) = \{([F, 1), (G, 2)], 1, 3), ([G, 3), (F, 4)], 3, 5), \dots\}$$

Dementsprechend ist $WT = \{(2, \{F, G\}), (4, \{F, G\}), \dots\}$. Für den minimalen Support $s = 1$ sind die Episoden $\alpha_1 = [F, G]$ und $\alpha_2 = [G, F]$ keine häufigen Episoden in \mathcal{S} , obwohl die Menge $V_1 = V_2 = \{F, G\}$ eine häufige Menge in WT ist.

Auf der anderen Seite können mehrere häufige Episoden dieselbe Knotenmenge als häufige Menge in $\mathcal{F}(WT, s)$ haben, was das folgende Beispiel demonstriert.

Beispiel 4.2 Für $\mathcal{S} = (([L, 7), (M, 8), (L, 9), (M, 10), (L, 11)], 7, 12)$ haben wir:

$$\text{WIN}(\mathcal{S}, 2) = \{([L, 7), (M, 8)], \cdot, \cdot), ([L, 9), (M, 10)], \cdot, \cdot), \\ ([M, 8), (L, 9)], \cdot, \cdot), ([M, 10), (L, 11)], \cdot, \cdot), \dots\}$$

$$WT = \{(2, \{L, M\}), (3, \{L, M\}), (4, \{L, M\}), (5, \{L, M\}), \dots\}.$$

Für den minimalen Support $s = 1$ ist $V = \{L, M\} \in \mathcal{F}(WT, 1)$ mit $\text{sup}(V, WT) = 4$. Die Episoden $\alpha_1 = [L, M]$ und $\alpha_2 = [M, L]$ haben V als Knotenmenge und sind häufig in \mathcal{S} mit $\text{sup}(\alpha_1, \mathcal{S}, 2) = 2$ und $\text{sup}(\alpha_2, \mathcal{S}, 2) = 2$. Dies bedeutet, dass mehrere häufige Episoden dieselbe Knotenmenge in $\mathcal{F}(WT, s)$ haben können.

Die Tatsache, die das Beispiel (4.2) widerspiegelt, führt uns zum Konzept „Episoden-Vertreter“.

Definition 4.2 (Episoden-Vertreter) Sei $\{(V, R_i, f)\}_{1 \leq i \leq n}$ eine endliche Menge von häufigen Episoden in einer Ereignis-Sequenz gegeben. Der Vertreter dieser Episoden ist die Menge V . Durch $V \cdot EP$ wird die Menge $\{(V, R_i, f)\}_{1 \leq i \leq n}$ gekennzeichnet.

In Bezug auf das Beispiel (4.1) ist $(\{F, G\}) \cdot EP = \emptyset$, während im Beispiel (4.2) $(\{L, M\}) \cdot EP = \{[L, M], [M, L]\}$ gilt. Also dient der Begriff „Episoden-Vertreter“ zu der konzeptionellen Darstellung der Beziehung zwischen den häufigen Mengen in $\mathcal{F}(WT, s)$ und den häufigen Episoden in $\mathcal{F}(\mathcal{S}, \text{win}, s)$. Diese Beziehung ist im folgenden Lemma formuliert, dessen Beweis direkt aus dem Lemma (4.2) und aus der Definition (4.2) zu schließen ist.

Lemma 4.3 Es gilt $\mathcal{F}(\mathcal{S}, \text{win}, s) = \bigcup_{V \in \mathcal{F}(WT, s)} V \cdot EP$

Nach dem Lemma (4.3) hat man also die folgende Frage zu beantworten: Wie lassen sich alle häufigen Episoden eines gegebenen Vertreters generieren? Der nächste Abschnitt (4.3) widmet sich einer Antwort auf diese Frage, während sich der Rest dieses Abschnittes um den Entwurf einer Datenstruktur zur Darstellung der beiden Mengen $\mathcal{F}(WT, s)$ und $\mathcal{F}(\mathcal{S}, \text{win}, s)$ kümmert. Der Entwurf dieser Datenstruktur hat die im Lemma (4.3) formulierten Beziehung zwischen $\mathcal{F}(WT, s)$ und $\mathcal{F}(\mathcal{S}, \text{win}, s)$ in Betracht zu ziehen.

Die Datenstruktur zur gemeinsamen Verwaltung von $\mathcal{F}(WT, s)$ und $\mathcal{F}(\mathcal{S}, \text{win}, s)$ heißt „Episoden-Manager“ und ist eine Baum-Struktur. Zur Illustration stellt die Abbildung (4.2) den Episoden-Manager von $\mathcal{F}(WT, 1)$ und $\mathcal{F}(\mathcal{S}, 4, 1)$ dar, wobei \mathcal{S} die in der Abbildung (4.1) repräsentierte Ereignis-Sequenz und WT die entsprechende, im linken Teil der Tabelle (4.2) repräsentierte Menge ist. Bevor wir die Datenstruktur „Episoden-Manager“ definieren können, nehmen wir an, dass die Ereignis-Typen einer lexikographischen Ordnung \leq_L unterliegen und die Elemente jeder häufigen Menge in $\mathcal{F}(WT, s)$ bezüglich \leq_L geordnet sind.

Die Wurzel des Baumes bzw. des Episoden-Managers ist mit der Zeichenkette „root“ markiert und dient einfach als Startknoten zur Konstruktion bzw. zum Durchlaufen des Baumes. Jeder inneren Knoten enthält eine einelementige häufige Menge aus $\mathcal{F}(WT, s)$ und die Menge der häufigen Episoden eines bestimmten Episoden-Vertreters, so dass folgendes gilt:

- Jeder Pfad
 $P = [\text{root}, (\{e_1\} \cdot EP, e_1), (\{e_1, e_2\} \cdot EP, e_2), \dots, (\{e_1, e_2, \dots, e_n\} \cdot EP, e_n)]$
 hat folgende Eigenschaften:

1. $e_1 \leq_L e_2 \leq_L \dots \leq_L e_n$

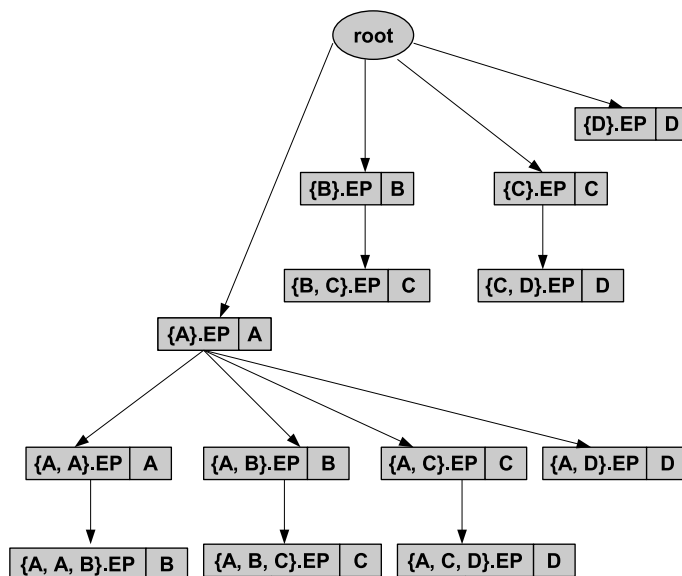


Abbildung 4.2: Der Episoden-Manager von $\mathcal{F}(\mathcal{S}, 4, 1)$ und $\mathcal{F}(WT, 1)$, wobei \mathcal{S} die in der Abbildung (4.1) dargestellten Ereignis-Sequenz und WT die im linken Teil der Tabelle (4.2) dargestellten Menge ist.

2. Für alle k mit $1 \leq k \leq n$ gilt: $\{e_1, e_2, \dots, e_k\} \in \mathcal{F}(WT, s)$
 3. Für alle k mit $1 \leq k \leq n$ gilt: $\{e_1, e_2, \dots, e_k\} \cdot EP$ ist die Menge aller häufigen Episoden in $\mathcal{F}(\mathcal{S}, win, s)$, deren Vertreter die Menge $\{e_1, e_2, \dots, e_k\}$ ist.
- Für jede Menge $V = \{e_1, e_2, \dots, e_n\} \in \mathcal{F}(WT, s)$ existiert ein Pfad der Form:
 $P = [root, (\{e_1\} \cdot EP, e_1), (\{e_1, e_2\} \cdot EP, e_2), \dots, (V \cdot EP, e_n)]$
 - Für jeden Pfad P existiert kein anderer Pfad P' mit $P = P'$.

Also repräsentiert jeder Pfad $P = [(\{e_1\} \cdot EP, e_1), \dots, (\{e_1, \dots, e_n\} \cdot EP, e_n)]$ im Episoden-Manager genau und nur genau eine häufige Menge $V = \{e_1, \dots, e_n\} \in \mathcal{F}(WT, s)$ und gleichzeitig alle häufigen Episoden, deren Knotenmenge $\{e_1, \dots, e_n\}$ ist.

Beispiel 4.3 Betrachten wir im Beispiel (4.2) die Ereignis-Sequenz \mathcal{S} und die entsprechenden Mengen $\mathcal{F}(WT, 1)$ und $\mathcal{F}(\mathcal{S}, 2, 1)$, dann repräsentiert der Pfad $P = [(\{L\} \cdot EP, L), (\{L, M\} \cdot EP, M)]$ im Episoden-Manager, der die beiden Mengen $\mathcal{F}(WT, 1)$ und $\mathcal{F}(\mathcal{S}, 2, 1)$ gemeinsam verwaltet, die häufige Menge $\{L, M\}$ und die häufigen Episoden $\{L, M\} \cdot EP = \{[L, M], [M, L]\}$.

Um die Hintergründe des Entwurfes dieser Datenstruktur zu erklären, zeigen wir hier die grobe Idee der Generierung aller häufigen Episoden, deren Vertreter die Menge

$\{e_1, \dots, e_n\} \in \mathcal{F}(WT, s)$. Dazu betrachten wir den Pfad

$P = [root, (\{e_1\} \cdot EP, e_1), (\{e_1, e_2\} \cdot EP, e_2), \dots, (\{e_1, \dots, e_n\} \cdot EP, e_n)]$ im Episoden-Manager. Es ist klar, dass $\{e_1\} \cdot EP = \{[e_1]\}$, $([e_1]) \cdot L = [e_1]$ und $([e_1]) \cdot T = e_1 \cdot T$ ist. Diese Tatsache gilt als Start-Information zur Generierung der Menge $\{e_1, e_2\} \cdot EP$, indem man bestimmte, im nächsten Abschnitt zu definierende Operationen auf den Tabellen $e_1 \cdot T$ und $e_2 \cdot T$ ausführt. Aus der Menge $\{e_1, e_2\} \cdot EP$ und der Tabelle $e_3 \cdot T$ wird die Menge $\{e_1, e_2, e_3\} \cdot EP$ generiert, indem man bestimmte Operationen auf jeder Episode in $\{e_1, e_2\} \cdot EP$ und der Tabelle $e_3 \cdot T$ ausführt. Dieser Vorgang wird bis zum Zeitpunkt fortgesetzt, in dem die Menge $\{e_1, e_2, \dots, e_n\} \cdot EP$ aus der Menge $\{e_1, e_2, \dots, e_{n-1}\} \cdot EP$ und der Tabelle $e_n \cdot T$ zu generieren ist. Danach erhalten wir die häufigen Episoden $\bigcup_{k=1}^n (\{e_1, \dots, e_k\} \cdot EP)$.

Dieses grob erklärte Verfahren setzt also voraus, dass der Episoden-Manager vor dem Beginn der Generierung aller häufigen Episoden mit allen häufigen Mengen aus $\mathcal{F}(WT, s)$ initialisiert werden muss. Bevor die Prozeduren zur Initialisierung des Episoden-Managers mit den häufigen Mengen formuliert werden können, muss zuerst die Frage geklärt werden, wie man die Kinderknoten eines Knotens verwalten bzw. speichern kann. Zur Lösung dieses Problems stellen wir im folgenden zwei Techniken vor.

Nach der ersten Methode wird jedem Ereignis-Typ $e \in E$ einen Index aus $\{1, 2, \dots, |E|\}$ zugeordnet, der durch $e \cdot ind$ gekennzeichnet wird. Dementsprechend wird die lexikographische Ordnung \leq_L wie folgt definiert:

$$\forall e', e'' \in E : e' \leq_L e'' \Leftrightarrow (e' \cdot ind) \leq (e'' \cdot ind)$$

Danach wird jedem Knoten $node$ im Episoden-Manager ein Array der Größe $|E|$ zugeordnet. Dieses Array ist durch $node \cdot children$ gekennzeichnet und wie folgt definiert: Falls der Knoten $node$ einen den Ereignis-Typ e enthaltenden Knoten als Kinderknoten hat, dann ist in $node \cdot children[e \cdot ind]$ ein Zeiger auf diesen Kinderknoten gespeichert. Falls der Knoten $node$ keinen den Ereignis-Typ e enthaltenden Knoten als Kinderknoten hat, dann ist in $node \cdot children[e \cdot ind]$ der null-Zeiger gespeichert.

Nach der zweiten Methode wird jedem Knoten $node$ im Episoden-Manager eine Hash-Tabelle zugeordnet, die durch $node \cdot HT$ gekennzeichnet ist. Die Schlüssel dieser Hash-Tabelle sind die Ereignis-Typen e ($e \in E$) selbst und der Eintrag in $node \cdot HT$ mit dem Schlüssel e ist ein Zeiger auf einen den Ereignis-Typ e enthaltenden Knoten, der einer der Kinderknoten von $node$ ist.

Eine Prozedur zur Initialisierung des Episoden-Manager ist in **Algorithm (8)** gegeben. Der Kern dieser Prozedur ist der Aufruf der in **Algorithm (9)** formulierten Prozedur. Bei der Formulierung beider Prozeduren wurde die erste der gerade oben erklärten Methoden zur Verwaltung der Kinderknoten eines Knoten angewendet.

4.3 Formulierung des Algorithmus

Basierend auf den Datenstrukturen, die in den Abschnitten (4.2.1), (4.2.2) und (4.2.3) entworfen wurden, entwickeln wir in diesem Abschnitt einen neuen Algorithmus zur Berechnung der Menge aller häufigen seriellen Episoden $\mathcal{F}^s(\mathcal{S}, win, s)$.

Algorithm 8 Eine Prozedur zur Initialisierung des Episoden-Managers

```

1: function INITEPISODENMANAGER( $\mathcal{F}(WT, s)$ ,  $|E|$ )
2:    $root \leftarrow$  new Node()
3:    $root \cdot label \leftarrow$  „root“
4:    $root \cdot children \leftarrow$  new Node[ $|E|$ ]
5:   for all  $V \in \mathcal{F}(WT, s)$  do
6:     sort( $V$ ) ▷ Lexikographische Sortierung
7:     insert( $V, root, |E|$ )
8:   end for
9:   return  $root$ 
10: end function

```

Algorithm 9 Eine Prozedur zum Einfügen einer Menge aus $\mathcal{F}(WT, s)$ in den Episoden-Manager.

```

1: procedure INSERT( $V, node, |E|$ )
2:   if  $V \neq \emptyset$  then
3:      $e \leftarrow V[1]$ 
4:      $V \leftarrow V \setminus [e]$ 
5:     if  $node \cdot children[e \cdot ind] = null$  then
6:        $newNode \leftarrow$  new Node()
7:        $newNode \cdot e \leftarrow e$ 
8:        $newNode \cdot EP \leftarrow \emptyset$ 
9:        $newNode \cdot children \leftarrow$  new Node[ $|E|$ ]
10:       $node \cdot children[e \cdot ind] \leftarrow newNode$ 
11:    else
12:       $newNode \leftarrow node \cdot children[e \cdot ind]$ 
13:    end if
14:    insert( $V, newNode, |E|$ )
15:  end if
16: end procedure

```

Um die Generierung der Menge aller häufigen parallelen Episoden $\mathcal{F}^p(\mathcal{S}, win, s)$ kümmern wir uns nicht mehr, weil sie gleich der Menge $\mathcal{F}(WT, s)$ ist. Dementsprechend und basierend auf der in **Algorithm (6)** auf der Seite (49) formulierten Prozedur „createTables“ und ihrer Laufzeit können wir das folgende Lemma angeben:

Lemma 4.4 *Das Problem der Entdeckung häufiger paralleler Episoden in einer Ereignis-Sequenz $\mathcal{S} = (S, T_s, T_e)$ über E lässt sich in Zeit $\mathcal{O}(win \times |E| \times (T_e - T_s + win))$ auf das Problem der Entdeckung der häufigen Mengen in Datenbank zurückführen. Dabei ist win die Breite der Zeitfenster.*

Die Berechnung der Menge $\mathcal{F}^s(\mathcal{S}, \text{win}, s)$ lässt sich nach dem Lemma (4.3) anfänglich wie folgt formulieren:

- Für jede Menge $V \in \mathcal{F}(WT, s)$:
 1. Berechne die Menge $V \cdot EP^s$, also die Menge aller häufigen seriellen Episoden, deren Vertreter V ist.
 2. Füge $V \cdot EP^s$ in $\mathcal{F}^s(\mathcal{S}, \text{win}, s)$ ein.

Dementsprechend hat man sich die Frage zu stellen, wie sich die Generierung der Menge $V \cdot EP^s$ effizient ausführen lässt. Um eine Antwort auf diese Frage geben zu können, studieren wir zuerst das folgende Beispiel. Wir betrachten eine Episode β mit $\beta = [Y, X]$, die häufig in irgendeiner Ereignis-Sequenz \mathcal{S} ist, und einen Ereignis-Typ Z , der in der selben Ereignis-Sequenz \mathcal{S} vorkommt. Die Datenstrukturen $\beta \cdot T$ und $Z \cdot T$ sind in der Tabelle (4.5) dargestellt.

id	$\beta \cdot T[id]$
3	{[3, 6], [4, 6]}
5	{[8, 10]}
9	{[15, 18]}

id	$Z \cdot T[id]$
3	{[2]}
5	{[7], [11]}
9	{[17]}

Tabelle 4.5: Die linke Tabelle ist $\beta \cdot T$ der Episode $\beta = [Y, X]$. Die rechte Tabelle ist $Z \cdot T$ des Ereignis-Typs Z .

Betrachten wir das Zeitfenster mit der Identifikationsnummer $id = 3$ und vergleichen wir die Instanz $[3, 6]$ des Vorkommens von β mit dem Zeitpunkt des Vorkommens von Z , dann schließen wir das Vorkommen der Episode $\alpha_1 = [Z, Y, X]$ in diesem Fenster, weil $[2, 3, 6]$ eine Instanz dieses Vorkommens ist. Auch hat die Episode α_1 eine andere Instanz des Vorkommens im betrachteten Fenster, nämlich die Instanz $[2, 4, 6]$, die durch das Vergleichen vom Zeitpunkt des Vorkommens von Z mit der anderen Instanz des Vorkommens von β , also mit der Instanz $[4, 6]$ zu erkennen ist. Mithilfe der selben Methode, also mit der Methode, die in einem bestimmten Zeitfenster die Zeitpunkte des Vorkommens von Z mit den Instanzen des Vorkommens von β vergleicht, lassen sich das Vorkommen der Episode $\alpha_2 = [Y, Z, X]$ im Fenster mit der Nummer $id = 9$ durch die Instanz $[15, 17, 18]$ und das Vorkommen der Episode $\alpha_3 = [Y, X, Z]$ im Fenster mit der Nummer $id = 5$ durch die Instanz $[8, 10, 11]$ schließen. Zum Überblick sind die Datenstrukturen $\alpha_1 \cdot T$, $\alpha_2 \cdot T$ und $\alpha_3 \cdot T$ in der Tabelle (4.6) dargestellt, wobei die Zeitpunkte des Ereignis-Typs Z zur Hervorhebung ihrer Rolle unterstrichen sind.

Ist der minimale Support $s = 1$, dann ist α_1 eine häufige Episode, während α_2 und α_3 nicht häufig sind. Dies kann man einfach erkennen, indem man die Anzahl der Zeilen in

<i>id</i>	$\alpha_1 \cdot T[id]$
3	{[<u>2</u> , 3, 6], [<u>2</u> , 4, 6]}
5	{[<u>7</u> , 8, 10]}

<i>id</i>	$\alpha_2 \cdot T[id]$
9	{[15, <u>17</u> , 18]}

<i>id</i>	$\alpha_3 \cdot T[id]$
5	{[8, 10, <u>11</u>]}

Tabelle 4.6: Die linke Tabelle ist $\alpha_1 \cdot T$ der Episode $\alpha_1 = [\underline{Z}, Y, X]$. Die mittlere Tabelle ist $\alpha_2 \cdot T$ der Episode $\alpha_2 = [Y, \underline{Z}, X]$. Die rechte Tabelle ist $\alpha_3 \cdot T$ der Episode $\alpha_3 = [Y, X, \underline{Z}]$. Die Episoden α_1 , α_2 und α_3 lassen sich aus $\beta \cdot T$ der Episode $\beta = [Y, X]$ und $Z \cdot T$ des Ereignis-Typs Z generieren, wobei $\beta \cdot T$ und $Z \cdot T$ die in der Tabelle (4.5) repräsentierten Datenstrukturen sind.

der jeweiligen Tabelle $\alpha_i \cdot T$ ($i = 1, 2, 3$) ermittelt.

Durch das Studieren dieses Beispiels haben wir also folgende Kenntnisse gewonnen: Aus einer seriellen Episode $\beta = (V_\beta, R_\beta, \cdot)$ und einem Ereignis-Typ e kann man alle häufigen seriellen Episoden erzeugen, deren Knotenmenge $V_\beta \cup \{e\}$ ist. Dies kann man bewerkstelligen, indem man für jedes Fenster \mathcal{W} , in dem β und e zusammen vorkommen, die zeitlichen in $\beta \cdot T[Id(\mathcal{W})]$ und in $e \cdot T[Id(\mathcal{W})]$ gespeicherten Informationen miteinander vergleicht. Wie diese zeitlichen Informationen miteinander zu vergleichen sind, werden wir formal erklären, sobald wir die Hauptprozedur unseres Algorithmus formuliert haben.

Zur Formulierung der Hauptprozedur ist die Frage zu beantworten, welche Voraussetzungen zu erfüllen sind, damit aus einer häufigen seriellen Episode β und einem Ereignis-Typ e alle häufigen seriellen Episoden mit der Knotenmenge $V_\beta \cup \{e\}$ generiert werden können. Erstens muss nach dem Lemma (4.2) die Menge $V_\beta \cup \{e\}$ in $\mathcal{F}(WT, s)$ sein, damit eine Episode $(V_\beta \cup \{e\}, \cdot, \cdot)$ überhaupt als häufige Episode in Frage kommen darf. Zweitens muss die Episode β schon erzeugt und in der Repräsentation $(\beta \cdot L, \beta \cdot T)$ dargestellt worden sein. Mit anderen Worten bedeutet dies, dass alle häufigen Episoden mit der Knotenmenge $V \subseteq V_\beta$ vorhanden sein müssen. Diese Voraussetzungen können durch die Ausnutzung der Eigenschaften und einen geeigneten Durchlauf der im Abschnitt (4.2.3) definierten Datenstruktur „Episoden-Manager“ sicher gestellt werden. Dies wird in der in **Algorithm (10)** formulierten Prozedur realisiert, die „computeFreqSerEpisodes“ heißt und als Hauptprozedur des neuen Algorithmus zur Berechnung der Menge $\mathcal{F}^s(\mathcal{S}, win, s)$ gilt.

Diese Prozedur führt eine Tiefensuche in der Datenstruktur „Episoden-Manager“ aus. Dabei werden für jeden Kinderknoten $(\{e_1, \dots, e_n\} \cdot EP, e_n)$ jedes besuchten Knotens $node = (\{e_1, \dots, e_{n-1}\} \cdot EP, e_{n-1})$ alle häufigen seriellen Episoden generiert, deren Knotenmenge $\{e_1, \dots, e_{n-1}, e_n\}$ ist. Dies wird erreicht, indem man für jede Episode $\beta \in \{e_1, \dots, e_{n-1}\} \cdot EP$ die Prozedur „createEpisodes“ (Zeile 5) aufruft, die alle häufigen seriellen Episoden aus β und dem Ereignis-Typ e_n erzeugt.

Um die Prozedur „createEpisodes“, also um die Generierung aller häufigen seriellen Epi-

Algorithm 10 Die Hauptprozedur des Algorithmus zur Berechnung der Menge $\mathcal{F}^s(\mathcal{S}, win, s)$. Sie ist mit dem aus der Prozedur „initEpisodenManager“ gelieferten Wurzelknoten des Episoden-Managers und mit dem minimalen Support s aufzurufen

```

1: function COMPUTEFREQSEREPISODES(node, s)
2:    $\mathcal{F}^s(\mathcal{S}, win, s) \leftarrow \emptyset$ 
3:   for all child  $\in$  node · children do
4:     for all  $\beta \in$  node · EP do
5:        $EP' \leftarrow createEpisodes(child \cdot e, \beta, s)$ 
6:        $child \cdot EP \leftarrow child \cdot EP \cup EP'$ 
7:     end for
8:      $\mathcal{F}_1^s(\mathcal{S}, win, s) \leftarrow computeFreqSerEpisodes(child, s)$ 
9:      $\mathcal{F}^s(\mathcal{S}, win, s) \leftarrow \mathcal{F}^s(\mathcal{S}, win, s) \cup \mathcal{F}_1^s(\mathcal{S}, win, s) \cup child \cdot EP$ 
10:  end for
11:  return  $\mathcal{F}^s(\mathcal{S}, win, s)$ 
12: end function

```

soden aus einer seriellen Episode und einem Ereignis-Typ zu formulieren, stellen wir im folgenden Lemmata vor, deren Aussagen und Beweisen die Ideen dieser Generierung repräsentieren. Das Lemma (4.5) begründet, warum wir eine serielle Episode um mindestens einen Ereignis-Typ bzw. einen Knoten erweitern müssen, um eine neue serielle Episode zu bekommen. Das Lemma (4.6) zeigt uns, wann und wie wir dies bewerkstelligen können, während die Lemmata (4.8) und (4.9) sagen, wann man aus einer seriellen Episode und einem Ereignis-Typ keine neue serielle Episode gewinnen kann. Wie viele Episoden mithilfe dieses Verfahrens als neue serielle Episoden in Frage kommen können, gibt das Lemma (4.10) an.

Es ist zu betonen, dass die Beweise der Lemmata (4.6) und (4.10) konstruktive Beweise sind und somit stellen sie die Verfahren dar, nach denen die Prozedur „createEpisodes“ formuliert wird.

Lemma 4.5 *Für jede echte serielle Oberepisode α von einer beliebigen seriellen Episode β gilt $|V_\alpha| > |V_\beta|$.*

Beweis: Wir nehmen an, dass $|V_\alpha| = |V_\beta|$ ist.

Da β eine Unterepisode von α ist, gibt es nach der Definition (3.7) eine injektive Funktion $g : V_\beta \rightarrow V_\alpha$ mit den folgenden Eigenschaften:

$$\forall x' \in V_\beta : f_\beta(x') = f_\alpha(g(x')) \quad (1)$$

$$\forall x', y' \in V_\beta : (x', y') \in R_\beta \Rightarrow (g(x'), g(y')) \in R_\alpha \quad (2)$$

Nach unserer Annahme und der Injektivität von g sind V_β und V_α bis auf Knoten-Benennung identisch. Daraus, aus (2) und aus der Tatsache, dass jede der Relationen R_α und R_β eine vollständige Ordnungsrelation in der jeweiligen Knotenmenge V_α bzw. V_β ist, ergibt sich, dass die Relationen R_α und R_β bis auf Knoten-Benennung identisch sind. Daraus und aus (1) ist zu schließen, dass α und β identisch sind. Dies ist ein Widerspruch zur Voraussetzung des Lemmas. Also ist unsere Annahme falsch und somit

muss $|V_\alpha| > |V_\beta|$ sein. □

Lemma 4.6 *Wenn für ein Fenster $\mathcal{W} = (W, \cdot, \cdot)$, eine serielle Episode $\beta = (V_\beta, R_\beta, f_\beta)$ und einen Ereignis-Typ $e \in E$ folgendes:*

1. e und β kommen zusammen in \mathcal{W} vor.
2. Für ein $t \in e \cdot T[Id(\mathcal{W})]$ existiert eine Instanz $I \in \beta \cdot T[Id(\mathcal{W})]$ mit $t \notin I$.

gilt, dann lässt sich eine neue serielle Episode α der Länge $|\beta| + 1$ konstruieren, die in dem selben Fenster \mathcal{W} vorkommt.

Beweis: Wir setzen $V_\alpha = V_\beta \cup \{y\}$ für einen neuen Knoten $y \notin V_\beta$ und definieren $f_\alpha : V_\alpha \rightarrow E$ wie folgt: $f_\alpha(y) = e$ und $\forall x \in V_\alpha \setminus \{y\} : f_\alpha(x) = f_\beta(x)$.

Es ist nun eine totale Ordnungsrelation R_α in V_α zu finden. Für einen Wert $t \in e \cdot T[Id(\mathcal{W})]$ und eine Instanz $I = [t_1, t_2, \dots, t_l] \in \beta \cdot T[Id(\mathcal{W})]$ mit $t \neq t_i$ ($1 \leq i \leq l$) haben wir einen der folgenden drei Fälle:

Fall 1: Es gilt $t < t_1$. In diesem Fall definieren wir R_α wie folgt:

$$R_\alpha = R_\beta \cup \{(y, y), (y, x_1), \dots, (y, x_i), \dots, (y, x_l)\}$$

Fall 2: Es gilt $t > t_l$. Hier ist R_α wie folgt zu definieren:

$$R_\alpha = R_\beta \cup \{(y, y), (x_1, y), \dots, (x_i, y), \dots, (x_l, y)\}$$

Fall 3: Es existiert ein Index i mit $1 < i \leq l$, für den $t_{i-1} < t < t_i$ gilt. In diesem Fall ist R_α wie folgt zu definieren:

$$R_\alpha = R_\beta \cup \{(y, y), (x_1, y), \dots, (x_{i-1}, y), (y, x_i), \dots, (y, x_l)\}$$

In jedem der drei Fälle ist R_α eine vollständige Ordnungsrelation in V_α und somit ist $\alpha = (V_\alpha, R_\alpha, f_\alpha)$ eine serielle Episode. Wir haben letztendlich zu zeigen, dass α in $\mathcal{W} = ((e_1, \cdot), \dots, (e_n, \cdot)), \cdot, \cdot$ vorkommt. Da β in \mathcal{W} vorkommt, existiert eine Funktion $h_\beta : V_\beta \rightarrow \{1, \dots, n\}$, die nach der Definition (3.4) das Vorkommen von β in \mathcal{W} beweist. Wir definieren $h_\alpha : V_\alpha \rightarrow \{1, \dots, n\}$ wie folgt: $\forall x \in V_\alpha \setminus \{y\} : h_\alpha(x) = h_\beta(x)$ und $h_\alpha(y) = p$, wobei p die Position des Ereignisses (e, t) in $((e_1, \cdot), \dots, (e_n, \cdot))$ ist. Nach der Konstruktion von α und den Eigenschaften von h_β erfüllt die Funktion h_α die Bedingungen der Definition (3.4) und somit kommt α in \mathcal{W} vor. □

Die im obigen Beweis konstruierte Episode α hat im ersten Fall offensichtlich die Instanz $[t, t_1, \dots, t_l]$, während sie im zweiten Fall die Instanz $[t_1, \dots, t_l, t]$ und im Dritten die Instanz $[t_1, \dots, t_{i-1}, t, t_i, \dots, t_l]$ hat.

Lemma 4.7 *Seien zwei serielle Episoden α, β gegeben, die zusammen in einem Zeitfenster \mathcal{W} vorkommen. Ist β eine Unterepisode von α , dann enthält jede Instanz des Vorkommens von α in \mathcal{W} eine Instanz des Vorkommens von β in \mathcal{W} .*

Beweis: Da β eine Unterepisode von α ist, gibt es nach der Definition (3.7) eine injektive Funktion $g : V_\beta \rightarrow V_\alpha$, die folgendes erfüllt:

$$\forall x' \in V_\beta : f_\beta(x') = f_\alpha(g(x')) \quad (1)$$

$$\forall x', y' \in V_\beta : (x', y') \in R_\beta \Rightarrow (g(x'), g(y')) \in R_\alpha \quad (2)$$

Da α in \mathcal{W} vorkommt, gibt es nach der Definition (3.4) eine injektive Funktion $h_\alpha : V_\alpha \rightarrow \{1, \dots, n\}$, die folgendes erfüllt:

$$\forall x \in V_\alpha : f_\alpha(x) = e_{h_\alpha(x)} \quad (3)$$

$$\forall x, y \in V_\alpha : x \neq y \wedge (x, y) \in R_\alpha \Rightarrow t_{h_\alpha(x)} < t_{h_\alpha(y)} \quad (4)$$

Diese Funktion definiert die Instanz $I_\alpha = [t_{h_\alpha(x_1)}, \dots, t_{h_\alpha(x_{|V_\alpha|})}]$ des Vorkommens von α .

Wir definieren die Funktion $h_\beta : V_\beta \rightarrow \{1, \dots, n\}$ wie folgt:

$$\forall x' \in V_\beta : h_\beta(x') = h_\alpha(g(x'))$$

Die Funktion h_β ist injektiv, weil die Funktionen g und h_α injektiv sind. Außerdem gilt folgendes:

$$\begin{aligned} \forall x' \in V_\beta : f_\beta(x') &= f_\alpha(g(x')) && \text{(wegen 1)} \\ &= e_{h_\alpha(g(x'))} && \text{(wegen 3)} \\ &= e_{h_\beta(x')} && \text{(Definition von } h_\beta) \end{aligned}$$

$$\forall x', y' \in V_\beta :$$

$$\begin{aligned} x' \neq y' \wedge (x', y') \in R_\beta &\Rightarrow g(x') \neq g(y') \wedge (g(x'), g(y')) \in R_\alpha && (g \text{ injektiv und 2)} \\ &\Rightarrow t_{h_\alpha(g(x'))} < t_{h_\alpha(g(y'))} && \text{(wegen 4)} \\ &\Rightarrow t_{h_\beta(x')} < t_{h_\beta(y')} && \text{(Definition von } h_\beta) \end{aligned}$$

Also erfüllt die Funktion h_β die Definition (3.4) und somit definiert sie eine Instanz

$$I_\beta = [t_{h_\beta(x'_1)}, \dots, t_{h_\beta(x'_{|V_\beta|})}].$$

Weil $t_{h_\beta(x'_i)} = t_{h_\alpha(g(x'_i))} \in I_\alpha$ für alle $x'_i \in V_\beta$ gilt, haben wir $I_\beta \subseteq I_\alpha$ □

Lemma 4.8 Sei $\beta = (V_\beta, R_\beta, f_\beta)$ eine serielle Episode gegeben, die zusammen mit einem Ereignis-Typ $e \in E$ in einem Fenster $\mathcal{W} = (W, \cdot, \cdot)$ vorkommt. Gilt $e \cdot T[Id(\mathcal{W})] \subseteq I$ für jede Instanz $I \in \beta \cdot T[Id(\mathcal{W})]$, dann kommt keine serielle Oberepisode $\alpha = (V_\alpha, R_\alpha, f_\alpha)$ von β in \mathcal{W} vor, für die folgendes gilt: $\exists x \in V_\alpha \setminus V_\beta : f_\alpha(x) = e$

Beweis: Sei $\alpha = (V_\beta \cup \{x_1, \dots, x_l\}, R_\alpha, f_\alpha)$ eine serielle Oberepisode von $\beta = (V_\beta, R_\beta, f_\beta)$ mit der Eigenschaft $\exists x^* \in \{x_1, \dots, x_l\} : f_\alpha(x^*) = e$ gegeben, wobei $x^* \notin V_\beta$ gilt. Wir nehmen an, dass α in $\mathcal{W} = ((e_1, \cdot), \dots, (e_n, \cdot), \cdot, \cdot)$ vorkommt.

Aufgrund unserer Annahme existiert eine Funktion $h_\alpha : V_\alpha \rightarrow \{1, \dots, n\}$, die nach der Definition (3.4) das Vorkommen von α in \mathcal{W} beweist. Diese Funktion definiert also nach der Definition (4.1) eine Instanz $I_\alpha \in \alpha \cdot T[Id(\mathcal{W})]$.

Da β eine Unterepisode von α ist, ist $I_\beta = I_\alpha \setminus \{t_{h_\alpha(x_1)}, \dots, t_{h_\alpha(x^*)}, \dots, t_{h_\alpha(x_l)}\}$ nach dem Lemma (4.7) eine Instanz des Vorkommens von β in \mathcal{W} . Setzen wir $t = t_{h_\alpha(x^*)}$, dann gilt nach der Voraussetzungen des Lemmas $t \in e \cdot T[Id(\mathcal{W})]$ und $t \in I_\beta$. Dies bedeutet, dass der Wert t zwei Mal an zwei verschiedenen Positionen in I_α vorkommt. Dies ist aber ein Widerspruch zur Definition des Vorkommens einer Episode in einem Fenster. Also ist I_α keine Instanz des Vorkommens von α in \mathcal{W} . Das heißt, unsere Annahme ist falsch und

somit kommt α in \mathcal{W} nicht vor. □

Lemma 4.9 *Wenn für ein Fenster $\mathcal{W} = (W, \cdot, \cdot)$, eine serielle Episode $\beta = (V_\beta, R_\beta, f_\beta)$ und einen Ereignis-Typ $e \in E$ folgendes:*

1. e und β kommen zusammen in \mathcal{W} vor.
2. Für ein $t \in e \cdot T[Id(\mathcal{W})]$ existiert eine Instanz $I_\beta \in \beta \cdot T[Id(\mathcal{W})]$ mit $t \in I_\beta$.

gilt, dann gilt für jede Instanz $I_\alpha \in \alpha \cdot T[Id(\mathcal{W})]$ des Vorkommens einer beliebigen seriellen Oberepisode α von β , dass $I_\beta \not\subseteq I_\alpha$ oder $t \notin I_\alpha \setminus I_\beta$ erfüllt.

Beweis: Nehmen wir an, dass eine Instanz I_α folgendes: $I_\beta \subseteq I_\alpha$ und $t \in I_\alpha \setminus I_\beta$ erfüllt, dann kommt der Wert t nach den Voraussetzungen des Lemmas an zwei verschiedenen Positionen in I_α vor. Dies ist aber ein Widerspruch zur Definition der Instanz des Vorkommens. □

Lemma 4.10 *Aus einer seriellen Episode β und einem Ereignis-Typ e , die zusammen in einem Zeitfenster \mathcal{W} vorkommen, kann man maximal $|\beta| + 1$ neue serielle Episoden erzeugen, die alle im selben Zeitfenster \mathcal{W} vorkommen.*

Beweis: Es gibt genau $|\beta| + 1$ Möglichkeiten, um einen neuen Knoten $y \notin V_\beta$ in die Liste $\beta \cdot L = [x_1, \dots, x_l]$ einzuordnen. Dadurch kann für jede der $|\beta| + 1$ Möglichkeiten eine neue serielle Oberepisode $\alpha = (V_\beta \cup \{y\}, R_\alpha, f_\alpha)$ mit $y \notin V_\beta$ und $f_\alpha(y) = e$ entstehen. Mehr als $|\beta| + 1$ Möglichkeiten kann es nicht geben, weil nach dem Lemma (4.5) jede serielle Oberepisode von β um mindestens einen neuen Knoten länger als β sein muss. Wir bezeichnen mit $[p_1, \dots, p_l, p_{l+1}]$ die Positionen, an denen der neue Knoten y in $\beta \cdot L$ eingefügt werden kann. Hierbei gilt: p_1 bedeutet y vor x_1 , p_i ($1 < i \leq l$) bedeutet y zwischen x_{i-1} und x_i und p_{l+1} bedeutet, dass y nach x_{l+1} einzufügen ist. Gibt es für ein p_i ($1 \leq i \leq l + 1$) einen Zeitpunkt $t \in e \cdot T[Id(\mathcal{W})]$ und eine Instanz $I \in \beta \cdot T[Id(\mathcal{W})]$ mit $t \notin I$ und der Eigenschaft:

$$\begin{array}{ll} t < t_1 & \text{falls } i = 1 \\ t_{i-1} < t < t_i & \text{falls } 1 < i \leq l \\ t > t_l & \text{falls } i = l + 1 \end{array}$$

dann lässt sich nach dem Lemma (4.6) eine neue serielle Oberepisode von β erzeugen, die in \mathcal{W} vorkommt. Dabei sieht die entsprechende Instanz im ersten Fall so $[t, t_1, \dots, t_l]$ aus, während sie die Gestalt $[t_1, \dots, t_{i-1}, t, t_i, \dots, t_l]$ im zweiten Fall und die Gestalt $[t_1, \dots, t_l, t]$ im dritten Fall hat.

Die Situation, in der es für ein p_i ($1 \leq i \leq l + 1$) ein $t \in e \cdot T[Id(\mathcal{W})]$ eine Instanz $I \in \beta \cdot T[Id(\mathcal{W})]$ mit $t \in I$ gibt, brauchen wir nach dem Lemma (4.9) nicht zu betrachten. □

Basierend auf dem Beweis des letzten Lemmas ist in **Algorithm (11)** eine Prozedur namens „createAllInstances“ formuliert. Sie generiert aus allen Instanzen des Vorkommens

einer Seriellen Episode β und allen Zeitpunkten des Vorkommens eines Ereignis-Typs e in einem Fenster \mathcal{W} alle Instanzen des Vorkommens derjenigen Episode α , die durch das Einfügen eines neuen Knoten y an die als Parameter gegebene Position p in der Liste $\beta \cdot L$ entstehen kann (Man sieht sich den Beweis des Lemmas (4.10) an).

Algorithm 11 Eine Prozedur zur Erzeugung aller Instanzen des Vorkommens derjenigen Episode, die durch das Einfügen eines neuen Knoten an die Position p in der Liste $\beta \cdot L$ entstehen kann.

```

1: function CREATEALLINSTANCES( $Ins(e, \mathcal{W}), Ins(\beta, \mathcal{W}), p$ )
2:    $allInstances \leftarrow \emptyset$ 
3:   for all  $t \in Ins(e, \mathcal{W})$  do
4:     for all  $[t_1, \dots, t_l] \in Ins(\beta, \mathcal{W})$  do
5:       if  $(p = 1) \wedge (t < t_1)$  then
6:          $allInstances \leftarrow allInstances \cup \{[t, t_1, \dots, t_l]\}$ 
7:       else if  $(1 < p \leq l) \wedge (t_{p-1} < t < t_p)$  then
8:          $allInstances \leftarrow allInstances \cup \{[t_1, \dots, t_{p-1}, t, t_p, \dots, t_l]\}$ 
9:       else if  $(p = l + 1) \wedge (t_l < t)$  then
10:         $allInstances \leftarrow allInstances \cup \{[t_1, \dots, t_l, t]\}$ 
11:      end if
12:    end for
13:  end for
14:  return  $allInstances$ 
15: end function

```

In **Algorithm (12)** ist die Prozedur „createEpisodes“ zur Erzeugung aller häufigen seriellen Episoden aus einer häufigen seriellen Episode β und einem Ereignis-Typ e wie folgt formuliert. Für jede mögliche neu zu erzeugende Episode α (Zeilen 3 und 4) werden alle Zeitfenster, in denen β und e zusammen vorkommen, betrachtet (Zeile 5) und in jedem dieser Zeitfenster werden alle Instanzen des Vorkommens der neu zu erzeugende Episode durch den Aufruf der Prozedur „createAllInstances“ generiert (Zeile 6). Ist die Menge der generierten Instanzen keine leere Menge (Zeile 7), dann wird sie in die Tabelle $\alpha \cdot T$ der neu zu erzeugenden Episode α entsprechend eingetragen (Zeile 8). Dass die Menge der generierten Instanzen keine leere Menge ist, bedeutet, dass die neu zu erzeugende Episode in dem im Moment betrachteten Zeitfenster vorkommt. Nach dem Durchlaufen aller Zeitfenster, in denen β und e zusammen vorkommen, und der Konstruktion der Tabelle $\alpha \cdot T$ der neu zu erzeugenden Episode α wird die Häufigkeit von α geprüft, indem die Anzahl der Tabelle $\alpha \cdot T$ ermittelt wird (Zeile 11). Ist die neu zu erzeugende Episode häufig, dann wird ihre Liste $\alpha \cdot L$ entsprechend dem Parameter p konstruiert (Zeilen 12 - 18) und danach wird die häufige serielle neu erzeugte Episode in die Menge aller neuen häufigen seriellen Episoden eingefügt (Zeile 20).

Man könnte sich die Frage stellen, warum alle Instanzen des Vorkommens der neu zu erzeugenden Episode α in dem betrachteten Fenster \mathcal{W} zu erzeugen sind, obwohl die

Algorithm 12 Eine Prozedur zur Generierung aller häufigen seriellen Episoden aus einem Ereignis-Typ e und einer seriellen Episode β .

```

1: function CREATEEPISODES( $e, \beta, s$ )
2:    $newEpisodes \leftarrow \emptyset$ 
3:   for  $p \leftarrow 1, |\beta| + 1$  do
4:      $\alpha \leftarrow \text{new Episode}()$ 
5:     for all  $id \in \{id \mid id \in e \cdot T \wedge id \in \beta \cdot T\}$  do
6:        $allInstances \leftarrow \text{createAllInstances}(e \cdot T[id], \beta \cdot T[id], p)$ 
7:       if  $allInstances \neq \emptyset$  then
8:          $\alpha \cdot T[id] \leftarrow allInstances$ 
9:       end if
10:    end for
11:    if  $|\alpha \cdot T| > s$  then
12:       $[e_1, \dots, e_l] \leftarrow \beta \cdot L$ 
13:      if  $1 < p \leq l$  then
14:         $\alpha \cdot L \leftarrow [e_1, \dots, e_{p-1}, e, e_p, \dots, e_l]$ 
15:      else if  $p = l + 1$  then
16:         $\alpha \cdot L \leftarrow [e_1, \dots, e_l, e]$ 
17:      else
18:         $\alpha \cdot L \leftarrow [e, e_1, \dots, e_l]$ 
19:      end if
20:       $newEpisodes \leftarrow newEpisodes \cup \{\alpha\}$ 
21:    end if
22:  end for
23:  return  $newEpisodes$ 
24: end function

```

Existenz einer einzigen Instanz in \mathcal{W} genug ist, um das Vorkommen von α in \mathcal{W} sicherzustellen. Die Antwort auf diese Frage liegt im Studieren des folgenden Beispiels.

Sei $\mathcal{W} = ((Y, 2), (X, 4), (Z, 5), (X, 7)), \cdot, \cdot)$ das betrachtete Zeitfenster. Für die Episode $\beta = [Y, X]$ gilt $\beta \cdot T[Id(\mathcal{W})] = \{[2, 4], [2, 7]\}$ und für die Episode $\alpha = [Y, Z, X]$ gilt $\alpha \cdot T[Id(\mathcal{W})] = \{[2, 5, 7]\}$. Wäre bei der Generierung der Episode β aus den Ereignis-Typen Y und X nur die Instanz $I_1 = [2, 4]$ erzeugt worden, dann hätte es keine Möglichkeit gegeben, die Episode α aus der Episode β und dem Ereignis-Typ Z zu generieren, weil das Vorkommen von α in \mathcal{W} nur durch die Betrachtung der Instanz $I_2 = [2, 7]$ von β und des Zeitpunktes $t = 5$ von Z zu beweisen ist.

Es ist also durch dieses Beispiel zu verstehen, warum wir bei der Formulierung der Prozedur „createAllInstances“ die „for all“-Schleife in den Zeilen (3) und (4) und nicht die „exists“-Anweisung angewendet haben, die in Bezug auf den Beweis des Lemmas (4.10) geeignet gewesen wäre.

4.4 Hauptschritte des Algorithmus

Die Hauptschritte des Algorithmus zur Berechnung der Menge $\mathcal{F}^s(\mathcal{S}, win, s)$ sind wie folgt zusammenzufassen.

1. Generierung der Tabellen WT und $e \cdot T$ für alle $e \in E$ mithilfe der in **Algorithm (6)** auf der Seite (49) formulierte Prozedur „createTables“.
2. Generierung der Menge $\mathcal{F}(WT, s)$ mithilfe des FP-growth-Algorithmus.
3. Initialisierung der Datenstruktur „Episoden-Manager“ mithilfe der in **Algorithm (8)** auf der Seite (56) formulierte Prozedur „initEpisodenManager“.
4. Generierung der Menge $\mathcal{F}^s(\mathcal{S}, win, s)$ mithilfe der in **Algorithm (10)** auf der Seite (59) formulierte Hauptprozedur „computeFreqSerEpisodes“.

Die durchgeführten Experimenten mit der Implementierung des Algorithmus werden im Abschnitt (7.2) diskutiert.

5 Kondensierte Repräsentationen der häufigen Mengen und der Assoziationsregeln

Die in Kapiteln 2 und 3 vorgestellten Probleme können nach ([21]) gemeinsam wie folgt beschrieben werden:

Für eine gegebene Datenbank \mathcal{D} , eine gegebene Sprache \mathcal{L} zur Beschreibung von Mustern und eine gegebene Selektion-Funktion $S_{\mathcal{D}} : \mathcal{L} \rightarrow \{0, 1\}$ zur Bewertung, ob ein Muster $p \in \mathcal{L}$ in \mathcal{D} interessant ist, ist die Menge $Th(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}}) = \{p \in \mathcal{L} \mid S_{\mathcal{D}}(p) = 1\}$ zu berechnen. Dabei heißt die Menge $Th(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ die Theorie von \mathcal{D} in Bezug auf \mathcal{L} und $S_{\mathcal{D}}$.

In der Terminologie der induktiven Datenbanken heißt die Selektion-Funktion $S_{\mathcal{D}}$ induktive Anfrage, deren Auswertung die Berechnung der Menge $Th(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ ist. Im Allgemeinen ist eine induktive Anfrage eine boolesche Ausdruck aus einer oder mehreren primitiven Bedingungen und eine induktive Datenbank besteht nicht nur aus Daten sondern auch aus Mustern und Modellen ([22], [23]). Dabei dienen die Muster zur Beschreibung lokaler Zusammenhänge in den Daten und die Modelle sind zur Formulierung globaler Aussagen über die Daten. Die Prozesse der Wissensentdeckung sind in den induktiven Datenbanken als induktive Anfrage definiert. Also dienen die induktiven Anfragen zur Entdeckung, Anwendung und Änderung von interessanten Mustern bzw. Modellen.

Im Kontext des Problems der Entdeckung häufiger serieller Episoden in einer Ereignis-Sequenz \mathcal{S} entspricht die Datenbank \mathcal{D} der Ereignis-Sequenz \mathcal{S} und die Sprache \mathcal{L} ist gleich der Menge aller seriellen Episoden. Hier ist eine Episode α interessant genau dann, wenn sie in der Ereignis-Sequenz \mathcal{S} häufig vorkommt. Mit anderen Worten ist $S_{\mathcal{S}}(\alpha) = 1$ wenn $sup(\alpha, \mathcal{S}, win) > s$ gilt, und $S_{\mathcal{S}}(\alpha) = 0$, wenn $sup(\alpha, \mathcal{S}, win) \leq s$ gilt. Die Theorie der Ereignis-Sequenz \mathcal{S} ist also die Menge $\mathcal{F}(\mathcal{S}, win, s)$.

Im Abschnitt (5.2) dieses Kapitels werden wir das Problem der Entdeckung aller starken Assoziationsregeln in einer Transaktionsdatenbank \mathcal{D} vorstellen. In diesem Zusammenhang ist die Sprache $\mathcal{L} = \{X \rightarrow Y \mid X, Y \subseteq \mathcal{I} \wedge Y \neq \emptyset \wedge X \cap Y = \emptyset\}$. Dabei ist eine Regel in \mathcal{L} interessant (stark) genau dann, wenn ihr Support in der zugrundeliegenden Transaktionsdatenbank \mathcal{D} größer als eine vorgegebene Zahl und ihre Konfidenz in \mathcal{D} auch größer als eine vorgegebene Zahl ist (Die genauen Definitionen finden sich im Unterabschnitt (5.2.1)). Die Theorie der Transaktionsdatenbank \mathcal{D} ist hier die Menge aller starken Assoziationsregeln.

Ein drittes Beispiel für die Theorie $Th(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ ist die Menge aller in einer Transak-

tionendatenbank \mathcal{D} über \mathcal{I} häufigen Mengen, die ein bestimmtes Objekt $a \in \mathcal{I}$ nicht enthalten und mindestens die Kardinalität $k \leq |\mathcal{I}|$ haben. Die Sprache \mathcal{L} zur Beschreibung solcher Muster ist hier gleich der Potenzmenge $2^{\mathcal{I}}$. Für eine Menge $X \in \mathcal{L}$ ist $S_{\mathcal{D}}(X) = 1$ genau dann, wenn $\text{sup}(X, \mathcal{D}) > s$ und $a \notin X$ und $|X| \geq k$ gilt.

Die Berechnung der Menge bzw. der Theorie $\text{Th}(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ kann in Bezug auf die Laufzeit der Berechnung und (oder) auf den Speicherbedarf extrem aufwändig sein. Sie kann sogar undurchführbar sein. Eine Lösung für dieses Problem ist die kondensierte Repräsentation der Menge $\text{Th}(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$. Es handelt sich bei dieser Lösung darum, eine Teilmenge $\mathcal{CR} \subseteq \mathcal{L}$ zu finden, so dass die Ableitung der Menge $\text{Th}(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ aus der Menge \mathcal{CR} möglich und effizient durchführbar ist. „Effizient durchführbar“ bedeutet in diesem Zusammenhang, dass man für die Ableitung der Menge $\text{Th}(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ aus der Menge \mathcal{CR} keinen weiteren Zugriff auf die Datenbank mehr braucht. Die Abbildung (5.1) demonstriert das Prinzip der kondensierten Repräsentation der interessanten Muster $\text{Th}(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$.

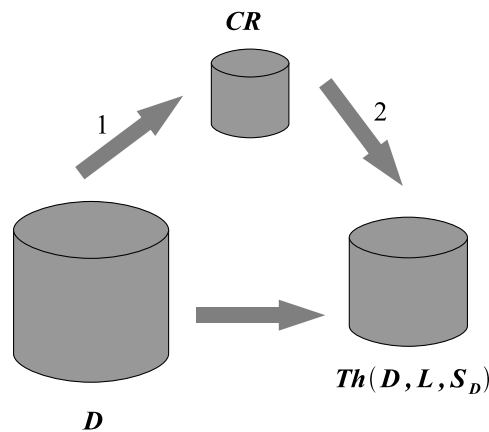


Abbildung 5.1: Diese Abbildung illustriert die Generierung der Menge $\text{Th}(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}})$ aus der Menge \mathcal{CR} ohne weiteren Zugriff auf die Datenbank \mathcal{D} (Pfeil 2). Die kondensierte Repräsentation \mathcal{CR} ist aber direkt aus der Datenbank zu generieren (Pfeil 1).

Die Selektion-Funktion $S_{\mathcal{D}}$ kann nach ([24]) (nur) indirekt durch eine Qualitätsfunktion $\phi_{\mathcal{D}} : \mathcal{L} \rightarrow \mathbb{R}^+$ definiert werden. Diese Funktion ordnet also jedem Muster $p \in \mathcal{L}$ einen Wert $\phi_{\mathcal{D}}(p)$ zu, mit dessen Hilfe festzustellen ist, ob das entsprechende Muster interessant ist. Ein Beispiel für eine Qualitätsfunktion ist der Support einer Menge in einer Transaktionsdatenbank. Im Zusammenhang mit der Qualitätsfunktion muss (soll) eine kondensierte Repräsentation \mathcal{CR} nicht nur die interessanten Muster, sondern auch

ihre Qualitäten ableiten können, ohne auf die Datenbank zugreifen zu müssen. In Bezug auf diese Anforderung sind zwei Arten von Repräsentationen zu unterscheiden. Eine Repräsentation ist exakt, wenn sie die Qualität des von ihr abgeleiteten Musters exakt berechnet. Kann aber eine Repräsentation die Qualität des von ihr abgeleiteten Musters nur ungefähr angeben, dann heißt sie approximative Repräsentation.

In diesem Kapitel werden wir die kondensierte Repräsentation anhand zweier Mengen, nämlich der Menge der häufigen Mengen ($Th(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}}) = \mathcal{F}(\mathcal{D}, s)$) und der Menge der starken Assoziationsregeln ($Th(\mathcal{D}, \mathcal{L}, S_{\mathcal{D}}) = \mathcal{AR}(\mathcal{D}, s, c)$), ausführlich studieren.

5.1 Kondensierte Repräsentationen der häufigen Mengen

5.1.1 Maximale häufige Mengen

Definition 5.1 (Maximale häufige Menge) *Eine Menge $X \subseteq \mathcal{I}$ ist eine maximale häufige Menge in \mathcal{D} genau dann, wenn $X \in \mathcal{F}(\mathcal{D}, s)$ ist und keine Menge $X' \in \mathcal{F}(\mathcal{D}, s)$ existiert, so dass $X \subset X'$ gilt.*

Die Menge aller maximalen häufigen Mengen in \mathcal{D} wird durch $\mathcal{MF}(\mathcal{D}, s)$ gekennzeichnet. D.h. $\mathcal{MF}(\mathcal{D}, s) = \{X \in \mathcal{F}(\mathcal{D}, s) \mid \neg \exists X' \in \mathcal{F}(\mathcal{D}, s) : X \subset X'\}$.

Diese Menge wird in ([21]) in Bezug auf die Eigenschaft der Häufigkeit als positive Grenze¹ des Suchraums vorgestellt. Weil jede Teilmenge einer häufigen Menge nach dem Lemma (2.2) häufig sein muss und alle Mengen aus $\mathcal{MF}(\mathcal{D}, s)$ in Bezug auf die Teilmengen-Beziehung maximale Mengen in $\mathcal{F}(\mathcal{D}, s)$ sind, gilt $\mathcal{F}(\mathcal{D}, s) = \bigcup_{X \in \mathcal{MF}(\mathcal{D}, s)} 2^X$. Dies führt zur Bezeichnung der Menge $\mathcal{MF}(\mathcal{D}, s)$ als repräsentative Menge für $\mathcal{F}(\mathcal{D}, s)$.

Um festzustellen, ob eine Menge $Y \subseteq \mathcal{I}$ eine häufige Menge in \mathcal{D} ist, ist es genug eine Menge $X \in \mathcal{MF}(\mathcal{D}, s)$ mit der Eigenschaft $Y \subseteq X$ zu finden. Dabei gibt es keine Möglichkeit, den Support von Y in \mathcal{D} zu ermitteln. Dies gilt als Nachteil der kondensierten Repräsentation von $\mathcal{F}(\mathcal{D}, s)$ durch die Menge $\mathcal{MF}(\mathcal{D}, s)$. Der Vorteil dieser Repräsentation ist aber der, dass die Kardinalität der Menge $\mathcal{MF}(\mathcal{D}, s)$ exponentiell kleiner als die Kardinalität der Menge $\mathcal{F}(\mathcal{D}, s)$ sein kann, was durch das folgende Beispiel ([24]) demonstriert wird.

Beispiel 5.1 *Betrachten wir die Transaktionsdatenbank $\mathcal{D} = \{(1, \mathcal{I}), (2, \mathcal{I})\}$ über die Objektmenge \mathcal{I} , dann gilt $\mathcal{MF}(\mathcal{D}, 1) = \{\mathcal{I}\}$ und $\mathcal{F}(\mathcal{D}, 1) = 2^{\mathcal{I}}$ für den minimalen Support $s = 1$.*

Für die Entdeckung der maximalen häufigen Mengen in einer Datenbank wurden viele Algorithmen, wie z.B. in ([15], [16], [17], [18]), entwickelt.

5.1.2 Minimale nicht-häufige Mengen

Definition 5.2 (Minimale nicht-häufige Menge) *Eine Menge $X \subseteq \mathcal{I}$ ist eine minimale nicht-häufige Menge in \mathcal{D} genau dann, wenn $X \notin \mathcal{F}(\mathcal{D}, s)$ ist und $X' \in \mathcal{F}(\mathcal{D}, s)$ für jede echte Untermenge X' von X gilt.*

¹Engl.: positive border

Die Menge aller minimalen nicht-häufigen Mengen wird durch $\mathcal{MNF}(\mathcal{D}, s)$ gekennzeichnet. D.h. $\mathcal{MNF}(\mathcal{D}, s) = \{X \subseteq \mathcal{I} \mid X \notin \mathcal{F}(\mathcal{D}, s) \wedge (\forall X' \subseteq \mathcal{I} : X' \subset X \Rightarrow X' \in \mathcal{F}(\mathcal{D}, s))\}$. Diese Menge wird in ([21]) in Bezug auf die Eigenschaft der Häufigkeit als negative Grenze² des Suchraums vorgestellt. Da $\mathcal{F}(\mathcal{D}, s) = \{Y \mid \exists X \in \mathcal{MNF}(\mathcal{D}, s) : Y \subset X\}$ gilt, gilt die Menge $\mathcal{MNF}(\mathcal{D}, s)$ als kondensierte Repräsentation für $\mathcal{F}(\mathcal{D}, s)$. Um also festzustellen, ob eine Menge $Y \subseteq \mathcal{I}$ häufig in \mathcal{D} ist, ist eine Menge $X \in \mathcal{MNF}(\mathcal{D}, s)$ mit der Eigenschaft $Y \subset X$ zu finden.

Diese Repräsentation hat wie die Repräsentation durch die Menge $\mathcal{MF}(\mathcal{D}, s)$ den Nachteil, dass es keine Möglichkeit gibt, den Support der als häufig festgestellten Menge Y zu ermitteln.

Das folgende Beispiel zeigt, dass die Anzahl der minimalen nicht-häufigen Mengen $|\mathcal{I}|$ -Mal größer als die Anzahl der häufigen Mengen sein kann.

Beispiel 5.2 *Betrachten wir $\mathcal{D} = \{(1, a_1), \dots, (|\mathcal{I}|, a_{|\mathcal{I}|}) \mid a_i \in \mathcal{I}\}$ über die Objektmenge $\mathcal{I} = \{a_1, \dots, a_{|\mathcal{I}|}\}$, dann ist $\mathcal{F}(\mathcal{D}, 1) = \{\emptyset\}$ und $\mathcal{MNF}(\mathcal{D}, 1) = \{\{a_i\} \mid a_i \in \mathcal{I}\}$. Also gilt $1 = |\mathcal{F}(\mathcal{D}, 1)| \leq |\mathcal{MNF}(\mathcal{D}, 1)| = |\mathcal{I}|$.*

Es kann $|\mathcal{F}(\mathcal{D}, s)| = |\mathcal{MNF}(\mathcal{D}, s)|$ sein, was durch das folgende Beispiel demonstriert ist.

Beispiel 5.3 *Für $\mathcal{I} = \{a, b, c\}$ betrachten wir die folgende Datenbank:*

$\mathcal{D} = \{(1, \{a\}), (2, \{a\}), (3, \{b\}), (4, \{b\}), (5, \{c\}), (6, \{c\})\}$.

Für den minimalen Support $s = 1$ haben wir $\mathcal{F}(\mathcal{D}, 1) = \{\{a\}, \{b\}, \{c\}\}$ und

$\mathcal{MNF}(\mathcal{D}, 1) = \{\{a, b\}, \{a, c\}, \{b, c\}\}$. *Also gilt $|\mathcal{F}(\mathcal{D}, 1)| = |\mathcal{MNF}(\mathcal{D}, 1)|$.*

Es gibt aber positive Fälle, in denen die Anzahl der minimalen nicht-häufigen Mengen exponentiell kleiner als die Anzahl der häufigen Mengen sein kann. Die demonstriert das folgende Beispiel ([24]).

Beispiel 5.4 *Betrachten wir $\mathcal{D} = \{(1, \mathcal{I} \setminus \{a_1\}), \dots, (|\mathcal{I}|, \mathcal{I} \setminus \{a_{|\mathcal{I}|})\} \mid a_i \in \mathcal{I}\}$ über die Objektmenge $\mathcal{I} = \{a_1, \dots, a_{|\mathcal{I}|}\}$, dann haben wir für den minimalen Support $s = 0$ $\mathcal{MNF}(\mathcal{D}, 0) = \{\mathcal{I}\}$ und $\mathcal{F}(\mathcal{D}, 0) = 2^{\mathcal{I}}$.*

Durch die drei Beispiele (5.2), (5.3) und (5.4) ist zu erkennen, wie schwierig die Abgrenzung der Anzahl der minimalen nicht-häufigen Mengen durch die Anzahl der häufigen Mengen ist.

In ([21]) wurde gezeigt, wie die Menge $\mathcal{MNF}(\mathcal{D}, s)$ aus der Menge $\mathcal{MF}(\mathcal{D}, s)$ erzeugt werden kann.

Falls $\mathcal{MNF}(\mathcal{D}, s)$ keine leere Menge ist, kann nach ([19]) die Anzahl der maximalen häufigen Mengen durch die Anzahl der minimalen nicht-häufigen Mengen von oben wie folgt abgeschätzt werden: $|\mathcal{MF}(\mathcal{D}, s)| \leq (|\mathcal{I}| - s + 1) |\mathcal{MNF}(\mathcal{D}, s)|$. Auf der anderen Seite gilt offensichtlich $|\mathcal{MNF}(\mathcal{D}, s)| \leq |\mathcal{I}| |\mathcal{MF}(\mathcal{D}, s)|$. Man kann also nicht entscheiden, welche der Repräsentationen $\mathcal{MF}(\mathcal{D}, s)$ oder $\mathcal{MNF}(\mathcal{MNF}(\mathcal{D}, s))$ in Bezug auf die Kardinalität besser ist, ohne dass die Struktur der Datenbank \mathcal{D} in Betracht gezogen wird.

²Engl.: negative border

Eine Repräsentation der Menge $\mathcal{F}(\mathcal{D}, s)$, die kleiner als $\mathcal{MF}(\mathcal{D}, s)$ und $\mathcal{MNF}(\mathcal{D}, s)$ ist, kann man in manchen Fällen nach ([25]) bekommen, indem man eine Teilmenge von $\mathcal{MF}(\mathcal{D}, s)$ und eine Teilmenge von $\mathcal{MNF}(\mathcal{D}, s)$ so auswählt, dass die beiden ausgewählten Mengen die Menge $\mathcal{F}(\mathcal{D}, s)$ vollständig bestimmen.

5.1.3 Abgeschlossene Mengen

Die kondensierte Repräsentation basiert hier auf dem Begriff der abgeschlossenen Mengen, der in der Analysis des formalen Konzeptes verwendet wird ([35], [36]). Die Analysis des formalen Konzeptes ist ein Teil der Verband-Theorie, die sich mit dem Studieren der durch die binären Relationen definierten Verband-Strukturen beschäftigt. Die Anwendung dieser Theorie im Bereich der Entdeckung der häufigen Mengen wurde in ([26], [31]) vorgeschlagen und führte zur kondensierten Repräsentation der häufigen Mengen mithilfe der abgeschlossenen Mengen.

Definition 5.3 (Abschluss einer Menge in \mathcal{D}) Der Abschluss³ $cl(X, \mathcal{D})$ einer Menge X in \mathcal{D} ist der Durchschnitt aller Transaktionen in \mathcal{D} , in denen X enthalten ist. Also ist $cl(X, \mathcal{D}) = \bigcap_{(id, T) \in \mathcal{D}: X \subseteq T} T$.

Definition 5.4 (Abgeschlossene Menge in \mathcal{D}) Eine Menge $X \subseteq \mathcal{I}$ ist geschlossen⁴ in \mathcal{D} genau dann, wenn $cl(X, \mathcal{D}) = X$ gilt.

Die Menge aller abgeschlossenen Mengen in \mathcal{D} wird durch $\mathcal{C}(\mathcal{D})$ bezeichnet. das heißt $\mathcal{C}(\mathcal{D}) = \{X \subseteq \mathcal{I} \mid cl(X, \mathcal{D}) = X\}$.

Definition 5.5 (Häufige abgeschlossene Menge in \mathcal{D}) Eine Menge $X \subseteq \mathcal{I}$ ist eine häufige abgeschlossene Menge in \mathcal{D} genau dann, wenn $cl(X, \mathcal{D}) = X$ und $X \in \mathcal{F}(\mathcal{D}, s)$ gilt.

Die Menge aller häufigen abgeschlossenen Mengen in \mathcal{D} wird durch $\mathcal{CF}(\mathcal{D}, s)$ gekennzeichnet. D.h. $\mathcal{CF}(\mathcal{D}, s) = \{X \in \mathcal{F}(\mathcal{D}, s) \mid cl(X, \mathcal{D}) = X\}$.

Eine Menge X heißt maximale häufige abgeschlossene Menge in \mathcal{D} genau dann, wenn $X \in \mathcal{CF}(\mathcal{D}, s)$ ist und keine $X' \in \mathcal{CF}(\mathcal{D}, s)$ existiert, so dass $X \subset X'$ gilt. Die Menge aller maximalen häufigen abgeschlossenen Mengen in \mathcal{D} wird durch $\mathcal{MCF}(\mathcal{D}, s)$ bezeichnet. D.h. $\mathcal{MCF}(\mathcal{D}, s) = \{X \in \mathcal{CF}(\mathcal{D}, s) \mid \neg \exists X' \in \mathcal{CF}(\mathcal{D}, s) : X \subset X'\}$.

Nach ([26], [27], [28], [31]) gelten folgende Eigenschaften:

1. $\forall X \subseteq \mathcal{I} : X \subseteq cl(X, \mathcal{D})$
2. $\forall X \subseteq \mathcal{I} : sup(X, \mathcal{D}) = sup(cl(X, \mathcal{D}), \mathcal{D})$
3. $\forall X \subseteq \mathcal{I} : cl(X, \mathcal{D}) = \bigcap_{X \subseteq Z \wedge Z \in \mathcal{C}(\mathcal{D})} Z$

³Engl.: closure

⁴Engl.: closed

4. $\forall X, Y \subseteq \mathcal{I} : X \in \mathcal{C}(\mathcal{D}) \Leftrightarrow (X \subset Y \Rightarrow \text{sup}(X, \mathcal{D}) > \text{sup}(Y, \mathcal{D}))$
5. $\mathcal{MF}(\mathcal{D}, s) = \mathcal{MCF}(\mathcal{D}, s)$

Basierend auf der vierten Eigenschaft kann eine abgeschlossene Menge X in \mathcal{D} wie folgt definiert werden([24]):

Eine Menge $X \subseteq \mathcal{I}$ ist eine abgeschlossene Menge \mathcal{D} genau dann, wenn folgendes gilt:

$$\forall Y \subseteq \mathcal{I} : X \subset Y \Rightarrow \text{sup}(X, \mathcal{D}) > \text{sup}(Y, \mathcal{D})$$

Die zweite Eigenschaft besagt, dass der Support einer Menge X gleich dem Support ihres Abschlusses ist. Dieser Abschluss ist nach der dritten Eigenschaft gleich der kleinsten abgeschlossenen Menge, die X enthält. Hat man demzufolge die Menge $\mathcal{CF}(\mathcal{D}, s)$ generiert, dann kann man den Abschluss jeder Menge $X \in \mathcal{F}(\mathcal{D}, s)$ und anschließend ihren Support berechnen. Also gilt die Menge $\mathcal{CF}(\mathcal{D}, s)$ als kondensierte Repräsentation der Menge $\mathcal{F}(\mathcal{D}, s)$.

Die Repräsentation durch $\mathcal{CF}(\mathcal{D}, s)$ hat gegenüber der Repräsentation durch $\mathcal{MF}(\mathcal{D}, s)$ bzw. durch $\mathcal{MNF}(\mathcal{D}, s)$ den Vorteil, dass man mit ihrer Hilfe den Support einer häufigen Menge erfahren kann, was in der Repräsentation durch die maximalen häufigen Mengen bzw. durch die minimalen nicht-häufigen Mengen nicht der Fall ist. Da nach der fünften Eigenschaft $\mathcal{MF}(\mathcal{D}, s) = \mathcal{MCF}(\mathcal{D}, s) \subseteq \mathcal{CF}(\mathcal{D}, s)$ gilt, schließen wir, dass die Repräsentation durch $\mathcal{MF}(\mathcal{D}, s)$ kondensierter als die Repräsentation durch $\mathcal{CF}(\mathcal{D}, s)$ ist.

Beispiel 5.5 *Es wird die im linken Teil der Tabelle (5.1) gegebene Datenbank \mathcal{D} über $\mathcal{I} = \{a, b, c, d, e\}$ betrachtet. Für den minimalen Support $s = 1$ sind alle häufigen abgeschlossenen Mengen in \mathcal{D} mit deren Support im rechten Teil der Tabelle (5.1) eingetragen. Für die Menge $X = \{a, c\}$ haben wir: $cl(X, \mathcal{D}) = \{a, c, d\} \cap \{a, b, c, e\} \cap \{a, b, c, e\} = \{a, c\} = X$. Dies bedeutet, dass X eine abgeschlossene Menge in \mathcal{D} ist. Für die Menge $Y = \{c, e\}$ haben wir: $cl(Y, \mathcal{D}) = \{b, c, e\} \cap \{a, b, c, e\} \cap \{a, b, c, e\} = \{b, c, e\} \neq Y$. Dies bedeutet, dass die Menge Y keine abgeschlossene Menge in \mathcal{D} ist. Da $Y \subset \{b, c, e\} \in \mathcal{CF}(\mathcal{D}, 1)$ ist, ist Y eine häufige Menge mit dem Support $\text{sup}(Y, \mathcal{D}) = \text{sup}(\{b, c, e\}, \mathcal{D}) = 3$. Obwohl die Menge $\{a, b, c, e\} \in \mathcal{CF}(\mathcal{D}, 1)$ eine Übermenge von Y ist, wird diese für die Ermittlung des Supports von Y nicht in Betracht gezogen, weil sie nicht die kleinste abgeschlossene häufige Übermenge von Y ist.*

Das folgende Beispiel ([24]) zeigt, dass die Anzahl der häufigen abgeschlossenen Mengen exponentiell kleiner als die Anzahl der häufigen Mengen sein kann.

Beispiel 5.6 *Betrachten wir die Datenbank $\mathcal{D} = \{(1, \mathcal{I}), (2, \mathcal{I})\}$ über \mathcal{I} , dann haben wir $\mathcal{CF}(\mathcal{D}, 1) = \{\mathcal{I}\}$ und $\mathcal{F}(\mathcal{D}, 1) = 2^{\mathcal{I}}$.*

Aber die Anzahl der häufigen abgeschlossenen Mengen kann exponentiell größer als die Anzahl der maximalen häufigen Mengen, was durch das folgende Beispiel demonstriert wird. Dieses Beispiel zeigt auch, dass die Anzahl der häufigen abgeschlossenen Mengen gleich der Anzahl der häufigen Mengen kann.

id	$\mathcal{D}[id]$
1	$\{b, e\}$
2	$\{a, c, d\}$
3	$\{b, c, e\}$
4	$\{a, b, c, e\}$
5	$\{a, b, c, e\}$

$X \in \mathcal{CF}(\mathcal{D}, 1)$	$sup(X, \mathcal{D})$
$\{c\}$	4
$\{b, e\}$	4
$\{a, c\}$	3
$\{b, c, e\}$	3
$\{a, b, c, e\}$	2

Tabelle 5.1: In **der rechten Tabelle** sind alle abgeschlossenen häufigen Mengen in der Datenbank \mathcal{D} aufgelistet, die in **der linken Tabelle** in horizontaler Darstellung gegeben ist.

Beispiel 5.7 Betrachten wir eine Transaktionsdatenbank \mathcal{D} über \mathcal{I} , die wie folgt aufgebaut ist:

- \mathcal{D} enthält mindestens eine Transaktion der Form (\cdot, \mathcal{I})
- Für jede Menge $X \in 2^{\mathcal{I}}$ gibt es genau und nur genau eine Transaktion $(\cdot, X) \in \mathcal{D}$

dann gilt $\mathcal{MF}(\mathcal{D}, 0) = \{\mathcal{I}\}$ und $\mathcal{CF}(\mathcal{D}, 0) = 2^{\mathcal{I}}$.

Zur Generierung der Menge $\mathcal{MF}(\mathcal{D}, s)$ wurde das Konzept „Generator“ eingefügt, wie z.B. in ([26]). Dabei ist der Generator einer abgeschlossenen Menge X in \mathcal{D} wie folgt definiert.

Definition 5.6 Eine Menge $G \subseteq \mathcal{I}$ ist ein Generator einer abgeschlossenen Menge X in \mathcal{D} über \mathcal{I} genau dann, wenn $cl(G, \mathcal{D}) = X$ gilt und keine echte Untermenge G' von G mit $cl(G', \mathcal{D}) = X$ existiert.

Die Menge aller Generatoren einer abgeschlossenen Menge X in \mathcal{D} wird durch $\mathcal{G}(X, \mathcal{D})$ gekennzeichnet. D.h. $\mathcal{G}(X, \mathcal{D}) = \{G \subseteq \mathcal{I} \mid cl(G, \mathcal{D}) = X \wedge \neg \exists G' \subset G : cl(G', \mathcal{D}) = X\}$
 Die Menge aller Generatoren in \mathcal{D} : $\bigcup_{X \in \mathcal{CF}(\mathcal{D})} \mathcal{G}(X, \mathcal{D})$ wird durch $\mathcal{G}(\mathcal{D})$ bezeichnet. Die Menge aller häufigen Generatoren: $\mathcal{G}(\mathcal{D}) \cap \mathcal{F}(\mathcal{D}, s)$ wird durch $\mathcal{GF}(\mathcal{D}, s)$ gekennzeichnet.

Beispiel 5.8 Wir betrachten die in der Tabelle (5.1) gegebene Datenbank \mathcal{D} mit ihren abgeschlossenen Mengen. Da $cl(\{b, c\}, \mathcal{D}) = cl(\{c, e\}, \mathcal{D}) = \{b, c, e\}$, $cl(\{c\}, \mathcal{D}) = \{c\} \neq \{b, c, e\}$ und $cl(\{b\}, \mathcal{D}) = cl(\{e\}, \mathcal{D}) = \{b, e\} \neq \{b, c, e\}$ ist, gilt $\mathcal{G}(\{b, c, e\}, \mathcal{D}) = \{\{b, c\}, \{c, e\}\}$.

Ein Generator $G \in \mathcal{G}(\mathcal{D})$ hat nach ([29])⁵ die folgende Eigenschaft:

$$\forall G, G' \subseteq \mathcal{I} : G \in \mathcal{G}(\mathcal{D}) \Leftrightarrow (G' \subset G \Rightarrow sup(G, \mathcal{D}) < sup(G', \mathcal{D})).$$

Also ist der Support eines Generators kleiner als der Support jeder echten Untermenge von ihm.

Die Konzepte „abgeschlossene Menge“ und „Generator“ spielen eine zentrale Rolle bei den kondensierten Repräsentationen der Assoziationsregeln, die wir im Abschnitt (5.3) diskutieren werden.

⁵Ein Generator ist in ([29]) als „key pattern“ bezeichnet.

5.1.4 Freie Mengen

In diesem Unterabschnitt stellen wir unter anderen das Konzept „ δ -freie Menge“ vor. Dieses Konzept wurde in ([33], [34]) aus dem Konzept „ δ -starke Assoziationsregel“ abgeleitet. Eine δ -starke Assoziationsregel ist eine Assoziationsregel $X \rightarrow Y$ (Siehe die Definition (5.10) auf der Seite (78)), für die $\text{sup}(X, \mathcal{D}) - \text{sup}(X \cup Y, \mathcal{D}) \leq \delta$ ($0 \leq \delta \leq |\mathcal{D}|$) gilt.

Definition 5.7 (δ -Freie Menge) Eine Menge $X \subseteq \mathcal{I}$ ist eine δ -freie Menge in \mathcal{D} über \mathcal{I} , wobei $0 \leq \delta \leq |\mathcal{D}|$ ist, genau dann, wenn folgendes gilt:

$$\forall Y \subseteq \mathcal{I}, \forall a \in \mathcal{I} : (Y \subseteq X \wedge a \in X \setminus Y) \Rightarrow \text{sup}(Y, \mathcal{D}) - \text{sup}(Y \cup \{a\}, \mathcal{D}) > \delta.$$

Die Menge aller δ -freien Mengen in \mathcal{D} wird durch $\text{Free}(\mathcal{D}, \delta)$ gekennzeichnet.

Nach ([33], [34]) gelten die folgenden Eigenschaften:

1. $\forall X, Y \subseteq \mathcal{I} : (Y \subseteq X \wedge X \in \text{Free}(\mathcal{D}, \delta)) \Rightarrow Y \in \text{Free}(\mathcal{D}, \delta)$
2. Für jede $X \subseteq \mathcal{I}$ existiert eine Untermenge Y von X , so dass $Y \in \text{Free}(\mathcal{D}, \delta)$ und $\text{sup}(Y, \mathcal{D}) \geq \text{sup}(X, \mathcal{D}) \geq \text{sup}(Y, \mathcal{D}) - \delta|X|$ gilt.
3. Für jede $X \subseteq \mathcal{I}$ und für jede Untermenge Y von X mit der Eigenschaften $Y \in \text{Free}(\mathcal{D}, \delta)$ und $\text{sup}(Y, \mathcal{D}) = \min\{\text{sup}(Z, \mathcal{D}) \mid Z \subseteq X \wedge Z \in \text{Free}(\mathcal{D}, \delta)\}$ gilt $\text{sup}(Y, \mathcal{D}) \geq \text{sup}(X, \mathcal{D}) \geq \text{sup}(Y, \mathcal{D}) - \delta|X|$.

Die erste Eigenschaft besagt, dass jede Untermenge einer δ -freien Menge δ -frei ist. Dies bedeutet, dass die Menge $\text{Free}(\mathcal{D}, \delta)$ in Bezug auf die Eigenschaft „ δ -frei“ abwärts abgeschlossen ist. Die zweite Eigenschaft besagt, dass der Support einer beliebigen Menge durch den Support einer ihrer Untermengen, die δ -frei ist, approximiert werden kann. Diese δ -freie Untermenge ist in der dritten Eigenschaft spezifiziert.

Die Menge aller häufigen δ -freien Mengen in \mathcal{D} wird mit $\text{FreqFree}(\mathcal{D}, s, \delta)$ bezeichnet. Dies bedeutet $\text{FreqFree}(\mathcal{D}, s, \delta) = \{X \subseteq \mathcal{I} \mid X \in \mathcal{F}(\mathcal{D}, s) \cap \text{Free}(\mathcal{D}, \delta)\}$.

Die Kondensierte Repräsentation der Menge $\mathcal{F}(\mathcal{D}, s)$ basiert hier auf der Menge $\text{FreqFree}(\mathcal{D}, s, \delta)$ und auf der Menge $\text{FreeBd}^-(\mathcal{D}, s, \delta)$, die wie folgt definiert ist:

$$\text{FreeBd}^-(\mathcal{D}, s, \delta) = \{X \subseteq \mathcal{I} \mid X \in \text{Free}(\mathcal{D}, \delta) \wedge X \notin \mathcal{F}(\mathcal{D}, s) \wedge (\forall Y \subseteq \mathcal{I} : Y \subset X \Rightarrow Y \in \text{FreqFree}(\mathcal{D}, s, \delta))\}.$$

Also bezeichnet die Menge $\text{FreeBd}^-(\mathcal{D}, s, \delta)$ die minimalen nicht-häufigen δ -freien Mengen in \mathcal{D} . Existiert für eine beliebige Menge $Y \subseteq \mathcal{I}$ eine Menge $X \in \text{FreeBd}^-(\mathcal{D}, s, \delta)$ mit $X \subseteq Y$, dann ist Y keine häufige Menge. Sonst ist sie eine häufige Menge in \mathcal{D} und ihr Support ist durch den Wert $\min\{\text{sup}(Z, \mathcal{D}) \mid Z \subseteq Y \wedge Z \in \text{FreqFree}(\mathcal{D}, s, \delta)\}$ zu approximieren. Also ist $\text{sup}(Y, \mathcal{D}) \approx \min\{\text{sup}(Z, \mathcal{D}) \mid Z \subseteq Y \wedge Z \in \text{FreqFree}(\mathcal{D}, s, \delta)\}$. Der verursachte Fehler bei dieser Approximation beträgt maximal $\epsilon = \max\{s, \delta|Y|\}$. Setzen wir $\overline{\text{sup}}(Y, \mathcal{D}) = \min\{\text{sup}(Z, \mathcal{D}) \mid Z \subseteq Y \wedge Z \in \text{FreqFree}(\mathcal{D}, s, \delta)\}$, dann ist also $|\text{sup}(Y, \mathcal{D}) - \overline{\text{sup}}(Y, \mathcal{D})| \leq \max\{s, \delta|Y|\}$.

Beispiel 5.9 Seien die Supports aller Mengen $X \subseteq \mathcal{I} = \{a, b, c, d\}$ in einer Transaktionsdatenbank \mathcal{D} über \mathcal{I} in der Tabelle (5.2) gegeben.

Betrachten wir die Menge $X_1 = \{a, b, c\}$, dann ist sie keine 1-freie Menge in \mathcal{D} , da $\text{sup}(\{a, b\}, \mathcal{D}) - \text{sup}(\{a, b, c\}, \mathcal{D}) = 8 - 7 \not\geq \delta = 1$ ist.

Betrachten wir die Menge $X_2 = \{b, d\}$, dann ist sie eine 1-freie Menge in \mathcal{D} , da $\text{sup}(\{b\}, \mathcal{D}) - \text{sup}(\{b, d\}, \mathcal{D}) = 4 - 3 = 1 \geq \delta = 1$, $\text{sup}(\{d\}, \mathcal{D}) - \text{sup}(\{b, d\}, \mathcal{D}) = 2 - 1 = 1 \geq \delta = 1$, $\text{sup}(\emptyset, \mathcal{D}) - \text{sup}(\{b\}, \mathcal{D}) = 7 - 4 = 3 \geq \delta = 1$ und $\text{sup}(\emptyset, \mathcal{D}) - \text{sup}(\{d\}, \mathcal{D}) = 9 - 2 = 7 \geq \delta = 1$ gilt.

Aber für $\delta = 2$ ist X_2 keine 2-freie Menge \mathcal{D} , da $\text{sup}(\{d\}, \mathcal{D}) - \text{sup}(\{b, d\}, \mathcal{D}) = 2 \not\geq 2$ ist. Es gilt: $\text{FreqFree}(\mathcal{D}, 5, 1) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$. Dementsprechend ist $\text{FreeBd}^-(\mathcal{D}, s, \delta) = \emptyset$. Für die Menge $Y = \{b, c, d\}$ haben wir: $\overline{\text{sup}}(Y, \mathcal{D}) = 7 = \text{sup}(Y, \mathcal{D})$. Also gibt es hier keinen Fehler bei der Approximation des Supports von Y .

Für $\delta = 2$ gilt $\text{FreqFree}(\mathcal{D}, 5, 2) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}\}$. Hier haben wir für Y : $\overline{\text{sup}}(Y, \mathcal{D}) = 9$. Der verursachte Fehler beträgt hier also 2 Transaktionen.

X	$\text{sup}(X, \mathcal{D})$	X	$\text{sup}(X, \mathcal{D})$
\emptyset	18	$\{b, c\}$	8
$\{a\}$	11	$\{b, d\}$	7
$\{b\}$	11	$\{c, d\}$	7
$\{c\}$	10	$\{a, b, c\}$	7
$\{d\}$	9	$\{a, b, d\}$	6
$\{a, b\}$	8	$\{a, c, d\}$	6
$\{a, c\}$	7	$\{b, c, d\}$	7
$\{a, d\}$	7	$\{a, b, c, d\}$	6

Tabelle 5.2: Die Supports aller Mengen $X \subseteq \mathcal{I} = \{a, b, c, d\}$ in einer Transaktionendatenbank \mathcal{D} über \mathcal{I} .

Für $\delta = 0$ haben wir die folgende Eigenschaft:

$$\forall X, Y \subseteq \mathcal{I} : X \in \text{Free}(\mathcal{D}, 0) \Leftrightarrow (Y \subset X \Rightarrow \text{sup}(Y, \mathcal{D}) - \text{sup}(X, \mathcal{D}) > 0).$$

Vergleichen wir diese Eigenschaft mit der auf der Seite (73) direkt nach dem Beispiel (5.8) formulierten Eigenschaft eines Generators, dann schließen wir $\text{Free}(\mathcal{D}, 0) = \mathcal{G}(\mathcal{D})$. Jede 0-freie Menge in \mathcal{D} ist also ein Generator in \mathcal{D} und umgekehrt. Dementsprechend können wir die Mengen $\mathcal{GF}(\mathcal{D}, s)$ und $\text{FreeBd}^-(\mathcal{D}, s, 0)$ als exakte kondensierte Repräsentation von $\mathcal{F}(\mathcal{D}, s)$ betrachten. Mit anderen Worten ist die Menge der häufigen Generatoren mit der Menge der minimalen nicht häufigen Generatoren eine exakte Repräsentation der häufigen Mengen. Da jede häufige abgeschlossene Menge mindestens einen häufigen Generator hat, ist die Repräsentation durch $\mathcal{CF}(\mathcal{D}, s)$ kondensierter als die Repräsentation durch $\mathcal{GF}(\mathcal{D}, s)$ und $\text{FreeBd}^-(\mathcal{D}, s, 0)$.

5.1.5 Disjunktion-freie Mengen

Die kondensierte Repräsentation der Menge $\mathcal{F}(\mathcal{D}, s)$ mithilfe des Konzeptes „Disjunktion-freie Menge“ basiert auf der folgenden Idee. Betrachten wir jede der die Menge $\{a\}$

enthaltenden Transaktionen der in der Tabelle (5.3) dargestellten Datenbank, erkennen wir, dass das Objekt c oder das Objekt d jeweils in jeder dieser Transaktionen enthalten ist. Diese Tatsache wird durch die Regel $\{a\} \Rightarrow c \vee d$ ausgedrückt, die einfache disjunktive auf $\{a, c, d\}$ basierende Regel heißt. In dieser Situation ist der Support der Menge $\{a\}$ gleich der Summe der Supports von $\{a, c\}$ und $\{a, d\}$ minus der Support von $\{a, c, d\}$, weil die die Menge $\{a, c, d\}$ enthaltenden Transaktionen zwei Mal in der Summe $sup(\{a, c\}, \mathcal{D}) + sup(\{a, d\}, \mathcal{D})$ gezählt sind. Kennt man also die Supports der Mengen $\{a\}$, $\{a, c\}$ und $\{a, d\}$, dann braucht man den Support von $\{a, c, d\}$ nicht extra zu berechnen, da $sup(\{a, c, d\}, \mathcal{D}) = sup(\{a, c\}, \mathcal{D}) + sup(\{a, d\}, \mathcal{D}) - sup(\{a\}, \mathcal{D})$ gilt.

id	a	b	c	d
1	0	1	1	0
2	0	1	1	1
3	0	0	1	0
4	0	0	0	1
5	1	1	1	0
6	1	1	1	0
7	1	0	1	1
8	1	0	0	1
9	0	1	0	1

Tabelle 5.3: Eine Trasnaktionsdatenbank über $\{a, b, c, d\}$

Definition 5.8 (Einfache disjunktive Regel) Eine einfache disjunktive auf $\emptyset \neq X \subseteq \mathcal{I}$ basierende Regel ist ein Ausdruck der Form: $Y \Rightarrow x_1 \vee x_2$, wobei $Y \subset X \wedge x_1, x_2 \in X \setminus Y$ ist. Die einfache disjunktive auf X basierende Regel $Y \Rightarrow x_1 \vee x_2$ ist gültig in einer Transaktionsdatenbank \mathcal{D} über \mathcal{I} genau dann, wenn x_1 oder x_2 in jeder Y enthaltenden Transaktion vorkommt. D.h.

$Y \Rightarrow x_1 \vee x_2$ gültig in \mathcal{D} gdw. $\{T \in \mathcal{D} | Y \subseteq T\} = \{T \in \mathcal{D} | Y \cup \{x_1\} \subseteq T \vee Y \cup \{x_2\} \subseteq T\}$.

Definition 5.9 (Disjunktion-freie Menge) Eine Menge $\emptyset \neq X \subseteq \mathcal{I}$ ist Disjunktion-freie Menge in \mathcal{D} genau dann, wenn jede einfache disjunktive auf X basierende Regel nicht gültig \mathcal{D} ist.

Die Menge aller Disjunktion-freien Mengen in \mathcal{D} wird durch $DFree(\mathcal{D})$ gekennzeichnet. Nach ([37], [38]) gelten die folgenden Eigenschaften:

1. Jede Untermenge einer Disjunktion-freien Menge ist Disjunktion-frei. Dies bedeutet, dass die Menge $DFree(\mathcal{D})$ in Bezug auf die Eigenschaft „Disjunktion-frei“ abwärts abgeschlossen ist.
2. Eine Menge $X \subseteq \mathcal{I}$ ist keine Disjunktion-freie Menge in \mathcal{D} genau dann, wenn es $x_1, x_2 \in X$ gibt, so dass folgendes gilt:
 $sup(X, \mathcal{D}) = sup(X \setminus \{x_1\}, \mathcal{D}) + sup(X \setminus \{x_2\}, \mathcal{D}) - sup(X \setminus \{x_1, x_2\}, \mathcal{D})$

Die kondensierte Repräsentation der Menge $\mathcal{F}(\mathcal{D}, s)$ besteht hier aus zwei Mengen $FreqDFree(\mathcal{D}, s)$ und $FreqBd^-(\mathcal{D}, s)$, die wie folgt definiert sind:

$$\begin{aligned} FreqDFree(\mathcal{D}, s) &= \{X \subseteq I \mid X \in \mathcal{F}(\mathcal{D}, s) \wedge X \in DFree(\mathcal{D})\} \\ FreqBd^-(\mathcal{D}, s) &= \{X \subseteq \mathcal{I} \mid X \in \mathcal{F}(\mathcal{D}, s) \wedge X \notin DFree(\mathcal{D}) \wedge \\ &\quad (\forall Y \subseteq \mathcal{I} : Y \subset X \Rightarrow Y \in FreqDFree(\mathcal{D}, s))\} \end{aligned}$$

Zur Ermittlung bei dieser Repräsentation, ob eine Menge X in $\mathcal{F}(\mathcal{D}, s)$ ist, ist die folgende Prozedur auszuführen:

1. Falls $X \in FreqDFree(\mathcal{D}, s)$ oder $X \in FreqBd^-(\mathcal{D}, s)$ ist, dann ist X häufig in \mathcal{D} .
2. Falls $X \notin FreqDFree(\mathcal{D}, s)$ ist und für jede Untermenge Y von X gilt: $Y \notin FreqBd^-(\mathcal{D}, s)$, dann ist X keine häufige Menge in \mathcal{D} .
3. Falls $X \notin FreqDFree(\mathcal{D}, s)$ ist und eine echte Untermenge Y von X mit $Y \in FreqBd^-(\mathcal{D}, s)$ existiert, dann gibt es $x_1, x_2 \in X$ mit der folgenden Eigenschaft:

$$sup(X, \mathcal{D}) = sup(X \setminus \{x_1\}, \mathcal{D}) + sup(X \setminus \{x_2\}, \mathcal{D}) - sup(X \setminus \{x_1, x_2\}, \mathcal{D}) \quad (*)$$
In diesem Fall wird die ganze Prozedur (Schritte 1, 2 und 3) jeweils für $X \setminus \{x_1\}$, $X \setminus \{x_2\}$ und $X \setminus \{x_1, x_2\}$ rekursiv wiederholt. Ist eine der Mengen $X \setminus \{x_1\}$, $X \setminus \{x_2\}$ und $X \setminus \{x_1, x_2\}$ keine häufige Menge in \mathcal{D} , dann ist zu schließen, dass X nicht häufig in \mathcal{D} ist. Sonst erhalten wir die Supports dieser Teilmengen, was die Berechnung des Supports von X mithilfe der Gleichung (*) ermöglicht. Dementsprechend ist zu entscheiden, ob $X \in \mathcal{F}(\mathcal{D}, s)$ gilt.

Beispiel 5.10 Wir betrachten die in der Tabelle (5.3) dargestellten Transaktionsdatenbank \mathcal{D} über $\mathcal{I} = \{a, b, c, d\}$. Es gilt für den minimalen Support $s = 0$ folgendes:

$$\begin{aligned} FreqDFree(\mathcal{D}, 0) &= \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}\} \\ FreqBd^-(\mathcal{D}, 0) &= \{\{a, b, c\}, \{c, d\}\} \end{aligned}$$

Betrachten wir die Menge $X_1 = \{a, b, d\}$, dann ist sie keine häufige Menge in \mathcal{D} , weil keine ihrer Teilmengen in der Menge $FreqBd^-(\mathcal{D}, 0)$ enthalten ist. In der Tat ist $sup(\{a, b, d\}, \mathcal{D}) = 0$.

Da die Menge $X_2 = \{a, c, d\}$ eine echte Teilmenge, nämlich die Menge $\{c, d\}$ hat, die in $FreqBd^-(\mathcal{D}, s)$ enthalten ist, haben wir:

$$sup(\{a, c, d\}, \mathcal{D}) = sup(\{a, c\}, \mathcal{D}) + sup(\{a, d\}, \mathcal{D}) - sup(\{a\}, \mathcal{D}) = 3 + 2 - 4 = 1 > 0.$$

Also ist X_2 mit dem Support 1 häufig in \mathcal{D} .

In ([39]) ist das Konzept der einfachen disjunktiven Regel zum Konzept der generalisierten disjunktiven Regel wie folgt erweitert:

Eine generalisierte disjunktive auf $\emptyset \neq X \subseteq \mathcal{I}$ basierende Regel ist ein Ausdruck der Form $X \setminus \{x_1, \dots, x_i, \dots, x_n\} \Rightarrow x_1 \vee \dots \vee x_i \vee \dots \vee x_n$, wobei $\{x_1, \dots, x_i, \dots, x_n\} \subseteq X$ und $0 < n \leq |X|$ gilt. Diese Regel ist gültig in einer Transaktionsdatenbank \mathcal{D} über \mathcal{I} genau dann, wenn folgendes gilt:

$$\{T \in \mathcal{D} \mid X \setminus \{x_1, \dots, x_i, \dots, x_n\} \subseteq T\} = \{T \in \mathcal{D} \mid (X \setminus \{x_1, \dots, x_n\}) \cup \{x_1\} \in T \vee \dots \vee (X \setminus \{x_1, \dots, x_i, \dots, x_n\}) \cup \{x_i\} \in T \vee \dots \vee (X \setminus \{x_1, \dots, x_i, \dots, x_n\}) \cup \{x_n\} \in T\}.$$

Eine Menge $\emptyset \neq X \subseteq \mathcal{I}$ heißt generalisierte Disjunktion-freie Menge⁶ in \mathcal{D} genau dann, wenn jede generalisierte disjunktive auf X basierende Regel nicht gültig in \mathcal{D} ist. Die kondensierte Repräsentation der Menge $\mathcal{F}(\mathcal{D}, s)$, die auf dem Konzept der generalisiert Disjunktion-freien Mengen basiert, ist in ([39]) studiert.

5.2 Assoziationsregeln

Das Konzept der Assoziationsregeln und das Problem ihrer Entdeckung in Transaktionendatenbanken wurden in ([1], [2]) formuliert. Sie werden zum Beispiel zur Warenkorbanalyse im Einzelhandel eingesetzt. Eine wichtige Frage bei solcher Analyse ist zum Beispiel, welche Artikel unter welchen Bedingungen bei einer Transaktion (einem Einkauf) gemeinsam gekauft werden. Beispielweise könnte folgende Aussage interessant sein: Falls in einem Einkauf Bier und Pizza zusammen gekauft werden, dann ist es wahrscheinlich, dass auch Kartoffelchips gekauft werden. Solche Informationen kann der Supermarkt dann zur Planung der Warenanordnung oder zur Entwicklung neuer Marketingstrategien nutzen.

5.2.1 Problembeschreibung

Definition 5.10 (Assoziationsregel) *Eine Assoziationsregel über der Menge $X \cup Y \subseteq \mathcal{I}$ ist ein Ausdruck der Form: $X \rightarrow Y$, wobei $Y \neq \emptyset$ und $X \cap Y = \emptyset$ gilt.*

Eine Transaktion $T \in \mathcal{D}$ erfüllt einer Assoziationsregel $X \rightarrow Y$ genau dann, wenn $X \cup Y \subseteq T$ gilt. Einer Assoziationsregel sind zwei statistische Parameter, nämlich der Support und die Konfidenz zugeordnet. Diese Parameter beziehen sich auf eine bestimmte Transaktionendatenbank und sind wie folgt definiert:

Der Support einer Assoziationsregel $X \rightarrow Y$ in einer Transaktionendatenbank \mathcal{D} über \mathcal{I} ist die Anzahl der Transaktionen in \mathcal{D} , die sie erfüllen. Bezeichnen wir den Support von $X \rightarrow Y$ in \mathcal{D} durch $sup(X \rightarrow Y, \mathcal{D})$, dann ist also $sup(X \rightarrow Y, \mathcal{D}) = sup(X \cup Y, \mathcal{D})$.

Die Konfidenz einer Assoziationsregel $X \rightarrow Y$ in einer Transaktionendatenbank \mathcal{D} über \mathcal{I} ist der relative Anteil derjenigen X enthaltenden Transaktionen in \mathcal{D} , die auch Y enthalten. Bezeichnen wir die Konfidenz von $X \rightarrow Y$ in \mathcal{D} durch $conf(X \rightarrow Y, \mathcal{D})$, dann ist also $conf(X \rightarrow Y, \mathcal{D}) = sup(X \cup Y, \mathcal{D}) / sup(X, \mathcal{D})$.

Der Wert $sup(X \rightarrow Y, \mathcal{D}) / |\mathcal{D}|$ ist also die Wahrscheinlichkeit dafür, dass eine Transaktion in \mathcal{D} die Assoziationsregel $X \rightarrow Y$ erfüllt. Dieser Wert heißt die Häufigkeit der Assoziationsregel in \mathcal{D} und wird durch $fr(X \rightarrow Y, \mathcal{D})$ gekennzeichnet. Dementsprechend ist die Konfidenz von $X \rightarrow Y$ die bedingte Wahrscheinlichkeit dafür, dass eine die Menge X mit der Wahrscheinlichkeit $sup(X, \mathcal{D}) / |\mathcal{D}|$ enthaltende Transaktion die Menge Y enthält. Der Support und die Konfidenz einer Assoziationsregel sind also Parameter, an denen die Relevanz der Regel beurteilt wird. Typischerweise sollen diese beiden Parameter relativ hoch sein, so dass sich das Problem der Entdeckung aller Assoziationsregeln in einer Transaktionendatenbank folgendermaßen beschreiben lässt:

⁶Engl.: generalized disjunction-free set

Für zwei gegebene Parameter $s \in \mathbb{N}$ und $c \in \mathbb{R}^+$ sind alle Assoziationsregeln $X \rightarrow Y$ in \mathcal{D} zu finden, die $\text{sup}(X \rightarrow Y, \mathcal{D}) > s$ und $\text{conf}(X \rightarrow Y, \mathcal{D}) > c$ erfüllen. Die Menge aller solchen Assoziationsregeln wird durch $\mathcal{AR}(\mathcal{D}, s, c)$ gekennzeichnet.

Das heißt $\mathcal{AR}(\mathcal{D}, s, c) = \{X \rightarrow Y \mid \text{sup}(X \rightarrow Y, \mathcal{D}) > s \wedge \text{conf}(X \rightarrow Y, \mathcal{D}) > c\}$.

Jede Regel aus $\mathcal{AR}(\mathcal{D}, s, c)$ wird manchmal als starke Assoziationsregel bezeichnet. Das Problem der Berechnung der Menge $\mathcal{AR}(\mathcal{D}, s, c)$ lässt sich in zwei Teilprobleme aufspalten:

1. Generierung der Menge $\mathcal{F}(\mathcal{D}, s)$.
2. Generierung der Menge $\mathcal{AR}(\mathcal{D}, s, c) = \{X \setminus X' \rightarrow X' \mid X \in \mathcal{F}(\mathcal{D}, s) \wedge \emptyset \neq X' \subseteq X \wedge \text{sup}(X, \mathcal{D})/\text{sup}(X \setminus X', \mathcal{D}) > c\}$.

Im Hinblick auf das erste Teilproblem, das im Kapitel (2) ausführlich behandelt wurde, kann man die Entdeckung aller starken Assoziationsregeln als eine Anwendung der häufigen Mengen in einer Transaktionsdatenbank betrachten. Deshalb kümmern wir uns im nächsten Unterabschnitt nur um das zweite Teilproblem.

5.2.2 Algorithmus zur Generierung der Assoziationsregeln

Die Basis für die Formulierung eines Algorithmus zu Generierung aller starken Assoziationsregeln besteht aus der folgenden Tatsache: Für alle $X_1, X_2, X \in \mathcal{F}(\mathcal{D}, s)$ haben wir folgendes:

$$\begin{aligned}
 X_1 \subset X_2 \subseteq X &\Rightarrow \emptyset \subseteq X \setminus X_2 \subset X \setminus X_1 \\
 &\Rightarrow \text{sup}(\emptyset, \mathcal{D}) \geq \text{sup}(X \setminus X_2, \mathcal{D}) \geq \text{sup}(X_1, \mathcal{D}) \\
 &\Rightarrow 1/\text{sup}(\emptyset, \mathcal{D}) \leq 1/\text{sup}(X \setminus X_2, \mathcal{D}) \leq 1/\text{sup}(X \setminus X_1, \mathcal{D}) \\
 &\Rightarrow \text{sup}(X, \mathcal{D})/\text{sup}(\emptyset, \mathcal{D}) \leq \text{sup}(X, \mathcal{D})/\text{sup}(X \setminus X_2, \mathcal{D}) \leq \\
 &\quad \text{sup}(X, \mathcal{D})/\text{sup}(X \setminus X_1, \mathcal{D}) \\
 &\Rightarrow \text{conf}(\emptyset \rightarrow X, \mathcal{D}) \leq \text{conf}(X \setminus X_2 \rightarrow X_2, \mathcal{D}) \leq \text{conf}(X \setminus X_1 \rightarrow X_1, \mathcal{D})
 \end{aligned}$$

Gilt $\text{conf}(X \setminus X_1 \rightarrow X_1, \mathcal{D}) \leq c$, dann gilt also $\text{conf}(X \setminus X_2 \rightarrow X_2, \mathcal{D}) \leq c$ für jede echte Übermenge X_2 von X_1 . Dies bedeutet mit anderen Worten folgendes: Bevor eine Assoziationsregel $X \setminus X_2 \rightarrow X_2$ mit $X_2 \subseteq X \in \mathcal{F}(\mathcal{D}, s)$ als starke Assoziationsregel in Frage kommen kann, muss zuerst jede Assoziationsregel $X \setminus X_1 \rightarrow X_1$ mit der Eigenschaft $X_1 \subset X_2$ in $\mathcal{AR}(\mathcal{D}, s, c)$ enthalten sein. Falls dies nicht zutrifft, dann ist sofort zu schließen, dass $X \setminus X_2 \rightarrow X_2 \notin \mathcal{AR}(\mathcal{D}, s, c)$ ist. Diese Überlegungen führen zum Prinzip der Konklusion-Kandidaten. Dabei ist eine Menge $X' \subseteq X \in \mathcal{F}(\mathcal{D}, s)$ ein Konklusion-Kandidat genau dann, wenn $X \setminus Y \rightarrow Y \in \mathcal{AR}(\mathcal{D}, s, c)$ für jede echte Teilmenge Y von X' gilt.

Für die Formulierung des Algorithmus zu Generierung der Menge $\mathcal{AR}(\mathcal{D}, s, c)$ definieren wir für jede Menge $X \in \mathcal{F}(\mathcal{D}, s)$ zwei Mengen \mathcal{C}_k und \mathcal{F}_k wie folgt: \mathcal{C}_k ist die Menge aller Konklusion-Kandidaten $X' \subseteq X$ der Länge k und \mathcal{F}_k ist die Menge aller Mengen X' in \mathcal{C}_k , für die $\text{conf}(X \setminus X' \rightarrow X', \mathcal{D}) > c$ gilt. Die Menge \mathcal{C}_{k+1} ist aus der Menge \mathcal{F}_k mithilfe der in **Algorithm (14)** formulierten Prozedur zu erzeugen. Dabei wird angenommen,

dass die Elemente jeder Menge $X' \in \mathcal{F}_k$ lexikographisch geordnet sind. Die in **Algorithm (13)** formulierte Prozedur erzeugt die Menge aller starken Assoziationsregeln basierend auf dem gerade oben definierten Prinzip der Konklusion-Kandidaten.

Algorithm 13 Eine Prozedur zur Generierung der Menge $\mathcal{AR}(\mathcal{D}, s, c)$

```

1: function COMPUTEARULES( $\mathcal{F}(\mathcal{D}, s), c$ )
2:    $\mathcal{AR}(\mathcal{D}, s, c) \leftarrow \emptyset$ 
3:   for all  $X \in \mathcal{F}(\mathcal{D}, s)$  do
4:      $k \leftarrow 1$ 
5:      $\mathcal{C}_k \leftarrow \{\{x\} \mid x \in X\}$ 
6:     while  $\mathcal{C}_k \neq \emptyset$  do
7:        $\mathcal{F}_k \leftarrow \{X' \in \mathcal{C}_k \mid \text{sup}(X, \mathcal{D}) / \text{sup}(X \setminus X', \mathcal{D}) > c\}$ 
8:        $\mathcal{AR}(\mathcal{D}, s, c) \leftarrow \mathcal{AR}(\mathcal{D}, s, c) \cup \{X \setminus X' \Rightarrow X' \mid X' \in \mathcal{F}_k\}$ 
9:        $k \leftarrow k + 1$ 
10:       $\mathcal{C}_k \leftarrow \text{generateCandidate}(\mathcal{F}_{k-1})$ 
11:     end while
12:   end for
13:   return  $\mathcal{AR}(\mathcal{D}, s, c)$ 
14: end function

```

Algorithm 14 Eine Prozedur zur Kandidaten-Generierung.

```

1: function GENERATECANDIDATE( $\mathcal{F}_k$ )
2:    $\mathcal{F}_{k+1} \leftarrow \emptyset$ 
3:   for all  $\{x_1, \dots, x_{k-1}, x_k\}, \{x_1, \dots, x_{k-1}, x'_k\} \in \mathcal{F}_k$  do
4:     if  $x_k <_L x'_k$  then  $\triangleright <_L =$  lexikographische Ordnung
5:        $X \leftarrow \{x_1, \dots, x_{k-1}, x_k, x'_k\}$ 
6:     else
7:        $X \leftarrow \{x_1, \dots, x_{k-1}, x'_k, x_k\}$ 
8:     end if
9:     if alle  $k$ -elementigen Teilmengen von  $X$  sind in  $\mathcal{F}_k$  then
10:       $\mathcal{F}_{k+1} \leftarrow \mathcal{F}_{k+1} \cup \{X\}$ 
11:    end if
12:   end for
13:   return  $\mathcal{F}_{k+1}$ 
14: end function

```

5.3 Kondensierte Repräsentationen der Assoziationsregeln

Für eine Menge $X \in \mathcal{F}(\mathcal{D}, s)$ der Länge k kann es $2^k - 1$ Assoziationsregeln $X \setminus X' \rightarrow X'$ geben, wobei $\emptyset \neq X' \subseteq X$ gilt. Dies ergibt sich aus der Tatsache, dass jede nicht leere Teilmenge von X die Konklusion der Regel sein kann. Infolgedessen kann die Anzahl der Assoziationsregeln in einer Datenbank von der Größenordnung $\mathcal{O}(|\mathcal{F}(\mathcal{D}, s)| \times 2^L)$ sein,

wobei L die Länge der größten häufigen Menge ist. Dies kann oft in dichten Transaktionsdatenbanken zutreffen, besonderes wenn der minimale Support s und die minimale Konfidenz c relativ klein sind. Die Konfrontation des Benutzers mit so vielen Informationen begrenzt die Nützlichkeit der Assoziationsregeln. Die kondensierten Repräsentationen der Menge $\mathcal{AR}(\mathcal{D}, s, c)$ mit den entsprechenden Inferenz-Mechanismen stellen eine mögliche Lösung für dieses Problem dar.

Die Inferenz-Methoden und die Klassen der kondensierten Repräsentationen werden in den Unterabschnitten (5.3.1) und (5.3.2) nach ([40]) vorgestellt. Die auf dem Konzept der repräsentativen Assoziationsregeln basierende Repräsentation ([41], [42], [43]) ist im Unterabschnitt (5.3.3) vorgestellt. Den Begriff der minimalen nicht redundanten Assoziationsregeln ([45], [46]) stellen wir im Unterabschnitt (5.3.4) vor. Die kondensierte Repräsentation nach ([28], [27], [26], [47]) wird im Unterabschnitt (5.3.5) präsentiert.

In manchen kondensierten Repräsentationen der Assoziationsregeln wird zwischen zwei Klassen von Assoziationsregeln unterschieden ([28] [45], [46]). Die erste Klasse besteht aus den Assoziationsregeln, deren Konfidenz gleich 1 ist. Diese Menge wird durch $\mathcal{AR}_{=1}(\mathcal{D}, s, c)$ gekennzeichnet. D. h. $\mathcal{AR}_{=1}(\mathcal{D}, s, c) = \{r \in \mathcal{AR}(\mathcal{D}, s, c) \mid \text{conf}(r, \mathcal{D}) = 1\}$. Jede Regel dieser Klasse heißt exakte Assoziationsregel. Die zweite Klasse wird durch $\mathcal{AR}_{<1}(\mathcal{D}, s, c)$ und besteht aus allen starken Regeln, deren Konfidenz kleiner als 1 ist. Also ist $\mathcal{AR}_{<1}(\mathcal{D}, s, c) = \{r \in \mathcal{AR}(\mathcal{D}, s, c) \mid \text{conf}(r, \mathcal{D}) < 1\}$. Jeder Regel in dieser Menge heißt approximierter Assoziationsregel.

Zur Vereinfachung der Schreibweise werden wir im Rest dieses Abschnittes die Mengennamen ohne Parameter verwenden. Zum Beispiel wird die Menge $\mathcal{F}(\mathcal{D}, s)$ einfach als \mathcal{F} geschrieben. Außerdem werden $\text{sup}(r, \mathcal{D})$ und $\text{conf}(r, \mathcal{D})$ für eine Regel $r \in \mathcal{AR}$ als $\text{sup}(r)$ bzw. $\text{conf}(r)$ geschrieben.

5.3.1 Inferenz-Methoden für Assoziationsregeln

Definition 5.11 (Inferenz-Methode) *Jede Methode zur Ableitung von Assoziationsregeln aus anderen Assoziationsregeln heißt Inferenz-Methode für Assoziationsregeln.*

Basierend auf ([40]) beschreiben wir im folgenden Inferenz-Mechanismen für Assoziationsregeln.

Armstrongs Axiome (AA): Armstrongs Axiome sind gültig nur für die exakten Assoziationsregeln, also für die Regeln der Menge $\mathcal{AR}_{=1}$, und besagen:

1. $\text{conf}(X \rightarrow X) = 1$
2. $\text{conf}(X \rightarrow Y) = 1 \Rightarrow \text{conf}(X \cup Z \rightarrow Y) = 1$
3. $\text{conf}(X \rightarrow Y) = 1 \wedge \text{conf}(Y \cup Z \rightarrow W) = 1 \Rightarrow \text{conf}(X \cup Z \rightarrow W) = 1$

Transitivität der Konfidenz (CTP): Falls $X \subset Y \subset Z \subseteq \mathcal{I}$ gilt, dann können der Support und die Konfidenz der Regel $X \rightarrow Z \setminus X$ aus den Supports und den Konfidenzen der Regeln $X \rightarrow Y \setminus X$ und $Y \rightarrow Z \setminus Y$ wie folgt abgeleitet werden:

1. $sup(X \rightarrow X \setminus Z) = sup(Y \rightarrow Z \setminus Y)$
2. $conf(X \rightarrow Z \setminus X) = conf(X \rightarrow Y \setminus X) \cdot conf(Y \rightarrow Z \setminus Y)$

Abdeckungs-Operator (C): Der Abdeckungs-Operator⁷ C wurde in ([41]) eingeführt und für eine Assoziationsregel $X \rightarrow Y$ wie folgt definiert:

$$C(X \rightarrow Y) = \{X \cup Z \rightarrow V \mid Z \cup V \subseteq Y \wedge Z \cap V = \emptyset \wedge V \neq \emptyset\}$$

Jeder Regel $X' \rightarrow Y' \in C(X \rightarrow Y)$ ist per Definition von der Regel $X \rightarrow Y$ abgedeckt. Der Abdeckungs-Operator hat folgende Eigenschaften:

1. Ist $r' \in C(r)$, dann ist $sup(r') \geq sup(r)$ und $conf(r') \geq conf(r)$.
2. Die Regel $X' \rightarrow Y'$ ist in $C(X \rightarrow Y)$ enthalten genau dann, wenn $X' \cup Y' \subseteq X \cup Y$ und $X \subseteq X'$ gilt.

Abschluss-Abschluss-Operator (CCI): Der Abschluss-Abschluss-Operator⁸ basiert auf zwei Eigenschaften des Abschluss einer Menge in einer Transaktionendatenbank (Abschnitt (5.1.3)). Die erste besagt, dass der Support einer Menge X gleich dem Support ihres Abschlusses $cl(X)$ ist. Die zweite besagt, dass der Abschluss $cl(X)$ einer Menge X gleich der kleinsten abgeschlossenen Menge ist, die X enthält. Demzufolge können der Support und die Konfidenz einer Regel $X \rightarrow Y \setminus X$ wie folgt berechnet werden ([40]):

1. $sup(X \rightarrow Y \setminus X) = sup(cl(Y))$
2. $conf(X \rightarrow Y \setminus X) = sup(cl(Y))/sup(cl(X))$

Im Laufe dieses Abschnittes gibt es Beispiele für diese Inferenz-Methoden.

5.3.2 Klassen der kondensierten Repräsentation der Assoziationsregeln

Definition 5.12 (Repräsentation der Assoziationsregeln) Eine kondensierte Repräsentation einer Menge von Assoziationsregeln ist ein Paar (\mathcal{R}, \models) , wobei $\mathcal{R} \subseteq \mathcal{AR}$ und \models eine Inferenz-Methode ist, die neue Assoziationsregeln aus der Menge \mathcal{R} ableitet.

Die kondensierten Repräsentationen der Assoziationsregeln sind nach ([40]) in die folgenden Klassen aufzuteilen:

Vollständige Repräsentation: (\mathcal{R}, \models) ist eine vollständige Repräsentation der Menge $\mathcal{AR}' \subseteq \mathcal{AR}$ genau dann, wenn die Inferenz-Methode \models alle Regeln in \mathcal{AR}' aus den Regeln der Menge \mathcal{R} ableiten kann.

Korrekte Repräsentation: (\mathcal{R}, \models) ist eine korrekte Repräsentation der Menge $\mathcal{AR}' \subseteq \mathcal{AR}$ genau dann, wenn jede aus der Menge \mathcal{R} durch \models abgeleitete Regel in \mathcal{AR}' enthalten ist.

⁷Engl.: Cover operator

⁸Engl.: Closure-Closure operator

Informative Repräsentation: (\mathcal{R}, \models) ist eine informative Repräsentation der Menge $\mathcal{AR}' \subseteq \mathcal{AR}$ genau dann, wenn sich der Support und die Konfidenz jeder aus der Menge \mathcal{R} durch \models abgeleiteten Regel berechnen lassen.

Jede vollständige und informative Repräsentation ist auch korrekt, weil anhand der Informativität der Repräsentation entschieden werden kann, ob die abgeleitete Regel stark ist. Die vollständige aber weder korrekte noch informative Repräsentation einer Menge \mathcal{AR}' ist am wenigsten wünschenswert, weil man aus der solchen Repräsentation andere Regeln ableiten kann, die nicht \mathcal{AR}' enthalten sind. Mit anderen Worten man leitet aus solchen Repräsentationen eine echte Übermenge der repräsentierten Menge \mathcal{AR}' ab.

5.3.3 Repräsentative Assoziationsregeln

Definition 5.13 (Repräsentative Assoziationsregel) Eine Regel $r \in \mathcal{AR}$ ist eine repräsentative Regel genau dann, wenn es keine andere Regel $r' \in \mathcal{AR}$ gibt, so dass $r \in C(r')$ gilt (C ist der Abdeckungs-Operator).

Die Menge aller repräsentativen Regeln in \mathcal{AR} wird durch \mathcal{RR} gekennzeichnet. Das heißt $\mathcal{RR} = \{r \in \mathcal{AR} \mid \neg \exists r' \in \mathcal{AR} : r \neq r' \wedge r \in C(r')\}$.

In der Literatur wurden die folgenden Algorithmen zur Berechnung der Menge \mathcal{RR} formuliert: GenAllRepresentatives in ([41]), FastGenAllRepresentatives in ([42]), Generate-RAR in ([44]) und GenRR in ([43]).

Wir werden im folgenden die Kern-Idee jedes dieser Algorithmen zeigen.

Der Algorithmus GenAllRepresentatives generiert die Menge \mathcal{RR} aus der Menge $\mathcal{F}(\mathcal{D}, s)$ basierend auf der folgenden Eigenschaft:

$$\mathcal{RR} = \{X \rightarrow Y \in \mathcal{AR} \mid \neg \exists (X' \rightarrow Y') \in \mathcal{AR} : (X = X' \wedge X \cup Y \subset X' \cup Y') \vee (X' \subset X \wedge X \cup Y = X' \cup Y')\}. \quad (1)$$

Diese Eigenschaft besagt also, dass eine Regel $X \rightarrow Z \setminus X \in \mathcal{AR}$ genau dann repräsentativ ist, wenn es keine andere Regel $X \rightarrow Z' \setminus X$ mit $Z \subset Z'$ und keine andere Regel $X' \rightarrow Z \setminus X'$ mit $X' \subset X$ gibt.

Der Algorithmus FastGenAllRepresentatives generiert die Menge der repräsentativen Regeln auch aus der Menge $\mathcal{F}(\mathcal{D}, s)$ aber basierend auf den folgenden Eigenschaften:

$$\forall X \rightarrow Z \setminus X \in \mathcal{AR} : \text{sup}(Z) = \text{maxSuperSup}(Z) \Rightarrow X \rightarrow Z \setminus X \notin \mathcal{RR}. \quad (2)$$

$$\forall X \rightarrow Z \setminus X \in \mathcal{AR} : \text{maxSuperSup}(Z)/\text{sup}(X) > c \Rightarrow X \rightarrow Z \setminus X \notin \mathcal{RR}. \quad (3)$$

Dabei ist $\text{maxSuperSup}(Z) = \max(\{\text{sup}(Y) \mid Y \in \mathcal{CF}(\mathcal{D}, s) \wedge Z \subset Y\} \cup \{0\})$. Zur Erinnerung ist $\mathcal{CF}(\mathcal{D}, s)$ die Menge aller abgeschlossenen häufigen Mengen in \mathcal{D} (Abschnitt (5.1.3)). Aus der Definition von $\text{maxSuperSup}(X)$ und aus der folgenden im Abschnitt (5.1.3) angegebenen Eigenschaft: $X \in \mathcal{CF}(\mathcal{D}, s) \Leftrightarrow \forall Y \supset X : \text{sup}(X) > \text{sup}(Y)$, ist folgendes zu schließen: $X \in \mathcal{CF}(\mathcal{D}, s) \Leftrightarrow \text{maxSuperSup}(X) \neq \text{sup}(X)$. Daraus und aus der Eigenschaft (2) ist zu schließen, dass man keine repräsentative Regel aus einer Menge $Z \in \mathcal{F}(\mathcal{D}, s)$ generieren kann, die keine abgeschlossene Menge ist. Also braucht man nur die Menge $\mathcal{CF}(\mathcal{D}, s)$ zur Generierung der Menge \mathcal{RR} . Dies wurde in ([44]) erkannt und führte zur Formulierung des Algorithmus Generate-RR, der aus der Menge $\mathcal{CF}(\mathcal{D}, s)$ die Menge \mathcal{RR} erzeugt. In Generate-RR ist zur Berechnung der Konfidenz der Regel

$X \rightarrow Z \setminus X$, wobei $Z \in \mathcal{CF}(\mathcal{D}, s)$ ist, die Formel $conf(X \rightarrow Z \setminus X) = sup(Z)/sup(cl(X))$ anzuwenden. Dabei ist der Abschluss $cl(X)$ von X wie folgt zu berechnen:

$$cl(X) = \bigcap_{Z \in \mathcal{CF}: X \subseteq Z} Z.$$

In ([43]) wurde die Frage gestellt, ob jede abgeschlossene häufige Menge zur Generierung einer repräsentativen Regel beitragen kann. Die Antwort auf die Frage ist negativ, weil sich nur aus den abgeschlossenen häufigen Mengen, die in der Menge $\mathcal{RS} \subseteq \mathcal{CF}$ enthalten sind, die repräsentativen Regeln erzeugen lassen. Die Menge \mathcal{RS} ist wie folgt definiert:

$\mathcal{RS} = \{Z \in \mathcal{CF} \mid sup(Z)/minSubSup(Z) > c \wedge maxSuperSup(Z)/minSubSup(Z) \leq c\}$
 Dabei ist $minSubSup(Z) = \min(\{sup(Y) \mid Y \in \mathcal{CF}(\mathcal{D}, s) \wedge Y \subset Z\} \cup \{|\mathcal{D}|\})$. Dementsprechend gilt für $Z \in \mathcal{F}(\mathcal{D}, s)$ und $X \subset Z$ die folgende Eigenschaft, mit deren Hilfe der Algorithmus GenRR in [43] formuliert ist:

$X \rightarrow Z \setminus X$ ist in \mathcal{RR} genau dann, wenn die folgenden vier Bedingungen zusammen erfüllt sind:

1. $Z \in \mathcal{RS}$ und $X \in \mathcal{GF}$
2. $sup(Z)/sup(X) > c$
3. $maxSuperSup(Z)/sup(X) \leq c$
4. $sup(Z)/minSubSup(X) \leq c$

Zur Erinnerung ist die Menge \mathcal{GF} die Menge aller häufigen Generatoren in \mathcal{D} (Abschnitt (5.1.3)). Also braucht der Algorithmus GenRR zur Generierung der Menge \mathcal{RR} die Mengen $\mathcal{RS} \subseteq \mathcal{CF}$ und \mathcal{GF} . Zusammenfassend gilt für jede repräsentative Regel $X \rightarrow Z \setminus X$, dass Z eine abgeschlossene Menge und X ein Generator von Z ist.

Nun können wir die auf den repräsentativen Regeln basierende Repräsentation der Menge \mathcal{AR} vorstellen. Weil der Abdeckungs-Operator C folgendes erfüllt:

- Für jede Regel $r \in \mathcal{AR}$ gibt es eine Regel $r' \in \mathcal{RR}$, so dass $r \in C(r')$ gilt.
- Ist $r \in C(r')$, dann gilt $sup(r) \geq sup(r')$ und $conf(r) \geq conf(r')$

ist (\mathcal{RR}, C) eine vollständige und korrekte Repräsentation der Menge \mathcal{AR} .

Beispiel 5.11 Wir betrachten die im linken Teil der Tabelle (5.4) dargestellte Transaktionsdatenbank \mathcal{D} . Für $s = 2$ und $c = 77\%$ ist $|\mathcal{AR}(\mathcal{D}, 2, 77\%)| = 112$ Regeln, während $|\mathcal{RR}(\mathcal{D}, 2, 77\%)| = 4$ Regeln ist. Dies bedeutet, dass die Anzahl der starken Regeln in \mathcal{D} durch die Repräsentation $(\mathcal{RR}(\mathcal{D}, 2, 77\%), C)$ ungefähr um 86% reduziert ist. Die Menge \mathcal{RR} ist im rechten Teil der Tabelle (5.4) dargestellt.

Um herauszufinden, ob $\{a\} \rightarrow \{b, c\} \in \mathcal{AR}(\mathcal{D}, 2, 77\%)$ ist, müssen wir nach den Eigenschaften der Inferenz-Methode C (Unterabschnitt (5.3.1)) eine Regel $X \rightarrow Y$ in $\mathcal{RR}(\mathcal{D}, 2, 77\%)$ finden, für die $X \subseteq \{a\}$ und $\{a, b, c\} \subseteq X \cup Y$ gilt. Da es solche Regel in $\mathcal{RR}(\mathcal{D}, 2, 77\%)$ nicht gibt, schließen wir, dass $sup(\{a\} \rightarrow \{b, c\}) \leq 2$ oder $conf(\{a\} \rightarrow \{b, c\}) \leq 77\%$ ist.

Da es für die Regel $\{b\} \rightarrow \{e\}$ die Regel $(\emptyset = X) \rightarrow (Y = \{b, c, d, e\})$ in $\mathcal{RR}(\mathcal{D}, 2, 77\%)$ mit $X \subseteq \{b\}$ und $\{b, e\} \subseteq X \cup Y$ gibt, schließen wir $\{b\} \rightarrow \{e\} \in \mathcal{AR}(\mathcal{D}, 2, 77\%)$. Aber leider können wir wider den Support noch die Konfidenz dieser Regel erfahren, weil die Repräsentation (\mathcal{RR}, C) nicht informativ ist.

id	$\mathcal{D}[id]$
1	$\{a, b, c, d, e\}$
2	$\{a, b, c, d, e, f\}$
3	$\{a, b, c, d, e, g, h\}$
4	$\{a, b, e\}$
5	$\{b, c, d, e, g, h\}$

$r \in \mathcal{RR}(\mathcal{D}, 2, 77\%)$	$sup(r, \mathcal{D})$	$conf(r, \mathcal{D})$
$\emptyset \rightarrow \{a, b, e\}$	4	4/5
$\emptyset \rightarrow \{b, c, d, e\}$	4	4/5
$\{a, c\} \rightarrow \{b, d, e\}$	3	1
$\{a, d\} \rightarrow \{b, c, e\}$	3	1

Tabelle 5.4: Die linke Tabelle stellt eine Transaktionsdatenbank \mathcal{D} über $\mathcal{I} = \{a, b, c, d, e, f, g, h\}$ dar. Für $s = 2$ und $c = 77\%$ stellt die rechte Tabelle alle repräsentativen Regeln in \mathcal{D} dar.

5.3.4 Minimale nicht redundante Assoziationsregeln

Definition 5.14 (Minimale nicht redundante Assoziationsregel) Eine Regel $X \rightarrow Z \setminus X$ heißt *minimale nicht redundante Regel* in \mathcal{AR} genau dann, wenn es keine andere Regel $X \rightarrow Z' \setminus X'$ in \mathcal{AR} mit den folgenden Eigenschaften gibt:

- $sup(X \rightarrow Z \setminus X) = sup(X' \rightarrow Z' \setminus X')$ und $conf(X \rightarrow Z \setminus X) = conf(X' \rightarrow Z' \setminus X')$
- $X' \subseteq X$ und $Z \subseteq Z'$

Die Menge aller minimalen nicht redundanten Regeln in \mathcal{AR} bzw. in \mathcal{D} wird durch \mathcal{MNR} gekennzeichnet. Nach ([40]) gilt folgendes:

$$\mathcal{MNR} = \{r \in \mathcal{AR} \mid \neg \exists r' \in \mathcal{AR} : r \neq r' \wedge r \in C(r') \wedge sup(r) = sup(r') \wedge conf(r) = conf(r')\}.$$

Eine minimale nicht redundante Regel kann also mit einer anderen Regel, die den selben Support und die selbe Konfidenz hat, nicht abgedeckt werden aber es kann möglich sein, sie mit einer anderen Regel abzudecken, die einen verschiedenen Support oder eine verschiedene Konfidenz hat. Auf der anderen Seite kann eine repräsentative Regel auf keinem Fall von einer anderen Regel abgedeckt werden. Daraus ist zu schließen, dass $\mathcal{RR} \subseteq \mathcal{MNR}$ gilt.

Aus der folgenden in ([40]) bewiesenen Tatsache:

Für jede Regel $r \in \mathcal{AR}$ gibt es eine andere Regel $r' \in \mathcal{MNR}$, so dass $r \in C(r')$, $sup(r) = sup(r')$ und $conf(r) = conf(r')$ gilt,

und aus der folgenden Eigenschaft des Abdeckungs-Operators C :

Für jede $r \in C(r')$ gilt $sup(r) \geq sup(r')$ und $conf(r) \geq conf(r')$,

schließen wir, dass (\mathcal{MNR}, C) eine vollständige und korrekte Repräsentation der Menge

$\mathcal{AR}(\mathcal{D}, s, c)$ ist. Ziehen wir die beiden folgenden Gleichungen in Betracht:

$$\forall r \in \mathcal{AR} : \text{sup}(r) = \max\{\text{sup}(r') \mid r \in C(r') \wedge r' \in \mathcal{MNR}\} \text{ und}$$

$$\forall r \in \mathcal{AR} : \text{conf}(r) = \max\{\text{conf}(r') \mid r \in C(r') \wedge r' \in \mathcal{MNR}\}$$

dann ist die Repräsentation (\mathcal{MNR}, C) auch informativ.

Da $\mathcal{RR} \subseteq \mathcal{MNR}$ gilt, ist also die Repräsentation (\mathcal{RR}, C) kondensierter als die Repräsentation (\mathcal{MNR}, C) aber dafür ist (\mathcal{MNR}, C) informativ.

Beispiel 5.12 Wir betrachten die im linken Teil der Tabelle (5.4) dargestellten Transaktionendatenbank \mathcal{D} . Für den minimalen Support $s = 2$ und die minimale Konfidenz $c = 77\%$ sind alle minimalen nicht redundanten Regeln in \mathcal{D} aufgelistet. Es ist zu bemerken, dass $|\mathcal{RR}(\mathcal{D}, 2, 77\%)| = 4 < 8 = |\mathcal{MNR}(\mathcal{D}, 2, 77\%)|$ ist. Dabei ist die Menge $\mathcal{RR}(\mathcal{D}, 2, 77\%)$ im rechten Teil der Tabelle (5.4) dargestellt.

Wir möchten mithilfe der Repräsentation $(\mathcal{MNR}(\mathcal{D}, 2, 77\%), C)$ der Menge $\mathcal{AR}(\mathcal{D}, 2, 77\%)$ wissen, ob die Regel $\{b, c\} \rightarrow \{d\}$ in $\mathcal{AR}(\mathcal{D}, 2, 77\%)$ ist, und gleichzeitig ihren Support und ihre Konfidenz berechnen. Da es z.B. die Regel $(\{c\} = X) \rightarrow (Y = \{b, d, e\})$ mit $X \subseteq \{b, c\}$ und $\{b, c, d\} \subseteq X \cup Y$ in $\mathcal{MNR}(\mathcal{D}, 2, 77\%)$ gibt, schließen wir, dass $\{b, c\} \rightarrow \{d\}$ eine starke Regel ist. Um den Support und die Konfidenz dieser Regel zu berechnen, müssen wir alle Regeln in $\mathcal{MNR}(\mathcal{D}, 2, 77\%)$ finden, die sie abdecken. Diese Regeln sind r_3 und r_5 in der Tabelle (5.5). Dementsprechend gilt:

$$\text{sup}(\{b, c\} \rightarrow \{d\}, \mathcal{D}) = \max\{\text{sup}(r_3, \mathcal{D}), \text{sup}(r_5, \mathcal{D})\} = 4.$$

$$\text{conf}(\{b, c\} \rightarrow \{d\}, \mathcal{D}) = \max\{\text{conf}(r_3, \mathcal{D}), \text{conf}(r_5, \mathcal{D})\} = 1.$$

Hätten wir die Repräsentation $(\mathcal{RR}(\mathcal{D}, 2, 77\%), C)$ gewählt, hätten wir den Support und die Konfidenz der Regel $\{b, c\} \rightarrow \{d\}$ nicht erfahren können.

$r \in \mathcal{MNR}(\mathcal{D}, 2, 77\%)$	$\text{sup}(r, \mathcal{D})$	$\text{conf}(r, \mathcal{D})$
$r_1 : \emptyset \rightarrow \{b, e\}$	5	1
$r_2 : \emptyset \rightarrow \{a, b, e\}$	4	4/5
$r_3 : \emptyset \rightarrow \{b, c, d, e\}$	4	4/5
$r_4 : \{a\} \rightarrow \{b, e\}$	4	1
$r_5 : \{c\} \rightarrow \{b, d, e\}$	4	1
$r_6 : \{d\} \rightarrow \{b, c, e\}$	4	1
$r_7 : \{a, c\} \rightarrow \{b, d, e\}$	3	1
$r_8 : \{a, d\} \rightarrow \{b, c, e\}$	3	1

Tabelle 5.5: Für $s = 2$ und $c = 77\%$ präsentiert diese Tabelle die minimalen nicht redundanten Regeln in der im linken Teil der Tabelle (5.4) dargestellten Transaktionendatenbank \mathcal{D} .

In ([45]) und in ([46]) ist die Menge \mathcal{MNR} zwar direkt definiert aber nicht direkt behandelt bzw. berechnet. Stattdessen sind dort die Mengen \mathcal{GB} und \mathcal{IB} behandelt. Die Menge \mathcal{GB} heißt generische Basis für $\mathcal{AR}_{=1}$ und ist wie folgt definiert:

$$\mathcal{GB} = \{X \rightarrow Y \setminus X \mid X \neq Y \wedge Y \in \mathcal{CF} \wedge X \in \mathcal{G}(Y)\}.$$

Zur Erinnerung ist $\mathcal{G}(Y)$ die Menge aller Generatoren von Y in der zugrundeliegenden Datenbank (Unterabschnitt (5.1.3)). Die Menge \mathcal{IB} heißt informative Basis für die Menge $\mathcal{AR}_{<1}$ und ist wie folgt definiert:

$$\mathcal{IB} = \{X \rightarrow Y \setminus X \mid Y \in \mathcal{CF} \wedge X \in \mathcal{G} \wedge cl(X) \subset Y \wedge conf(X \rightarrow Y \setminus X) > c\}.$$

Zur Erinnerung ist \mathcal{G} die Menge aller Generatoren in der zugrundeliegenden Datenbank (Unterabschnitt (5.1.3)). Die Menge der starken exakten Regeln $\mathcal{AR}_{=1}$ und die Menge der starken approximierten Regeln $\mathcal{AR}_{<1}$ sind auf der Seite (81) definiert.

Es wurde in ([45]) folgendes gezeigt, dass (\mathcal{GB}, CCI) eine vollständige und informative Repräsentation der Menge $\mathcal{AR}_{=1}$ ist und dass $\mathcal{GB} \subseteq \mathcal{MNR}$ und $\mathcal{IB} \subseteq \mathcal{MNR}$ gilt. Außerdem ist in ([40]) bewiesen, dass $\mathcal{MNR} = \mathcal{GB} \cup \mathcal{IB}$ ist. In ([45]) wurde behauptet, dass (\mathcal{IB}, CCI) eine vollständige und informative Repräsentation der Menge $\mathcal{AR}_{<1}$ sei. Aber diese Behauptung ist in ([40]) widerlegt. Das folgende Beispiel bestätigt dies und zeigt gleichzeitig, wie die Inferenz-Methode CCI anzuwenden ist.

Beispiel 5.13 Wir betrachten die Repräsentation $(\mathcal{IB}(\mathcal{D}, 2, 77\%), CCI)$ der Menge $\mathcal{AR}_{<1}(\mathcal{D}, 2, 77\%)$, wobei die Menge $\mathcal{IB}(\mathcal{D}, 2, 77\%)$ im linken Teil der Tabelle (5.6) aufgelistet ist.

Wir möchten basierend auf dieser Repräsentation herausfinden, ob die Regel $\{b\} \rightarrow \{e\}$ in $\mathcal{AR}(\mathcal{D}, 2, 77\%)$ ist. Nach der Inferenz-Methode CCI (Unterabschnitt (5.3.1)) gilt:

$$sup(\{b\} \rightarrow \{e\}) = sup(cl(\{b, e\}), \mathcal{D}) \text{ und}$$

$conf(\{b\} \rightarrow \{e\}) = sup(cl(\{b, e\}), \mathcal{D}) / sup(cl(\{b\}), \mathcal{D})$. Weil jede Regel $X \rightarrow Y \setminus X$ in \mathcal{IB} nach der Definition der informativen Basis eine häufige abgeschlossene Menge, nämlich die Menge Y liefert, können wir aus der Menge $\mathcal{IB}(\mathcal{D}, 2, 77\%)$ nur zwei häufige abgeschlossene Mengen $Y_1 = \{a, b, e\}$ und $Y_2 = \{b, c, d, e\}$ herleiten.

Weil weder $Y_1 \subseteq Y_2$ noch $Y_2 \subseteq Y_1$ gilt, kann die Inferenz-Methode CCI den Abschluss sowohl der Menge $\{b, e\}$ als auch der Menge $\{b\}$ nicht eindeutig bestimmen. Mit anderen Worten kann die Inferenz-Methode CCI die kleinste Menge von den beiden Mengen Y_1 und Y_2 nicht bestimmen, um z.B. den Abschluss der Menge $\{b, e\}$ als die kleinste häufige abgeschlossene Menge, die $\{b, e\}$ enthält, zu berechnen. Also kann die Repräsentation $(\mathcal{IB}(\mathcal{D}, 2, 77\%), CCI)$ darüber nicht entscheiden, ob $\{b\} \rightarrow \{e\}$ in $\mathcal{AR}_{<1}(\mathcal{D}, 2, 77\%)$ ist.

$r \in \mathcal{IB}(\mathcal{D}, 2, 77\%)$	$sup(r, \mathcal{D})$	$conf(r, \mathcal{D})$
$\emptyset \rightarrow \{a, b, e\}$	4	4/5
$\emptyset \rightarrow \{b, c, d, e\}$	4	4/5

$X \in \mathcal{CF}(\mathcal{D}, 2)$	$sup(X, \mathcal{D})$
\emptyset	5
$\{b, e\}$	5
$\{a, b, e\}$	4
$\{b, c, d, e\}$	4
$\{a, b, c, d, e\}$	3

Tabelle 5.6: Für $s = 2$ und $c = 77\%$ zeigt der linke Teil dieser Tabelle die informative Basis der Datenbank, die im linken Teil der Tabelle (5.4) dargestellt ist. Der rechte Teil präsentiert die Menge $\mathcal{CF}(\mathcal{D}, 2)$

Jede Regel in \mathcal{IB} kann nach ([45]) aus der Menge \mathcal{RI} durch die Anwendung der Inferenz-Methode *CTP* abgeleitet werden, wobei die Menge \mathcal{RI} wie folgt definiert ist:

$\mathcal{RI} = \{X \rightarrow Y \setminus X \in \mathcal{IB} \mid cl(X) \text{ ist eine maximale echte Teilmenge von } Y \in \mathcal{CF}\}$
und transitive Reduktion der informativen Basis \mathcal{IB} heißt. Also ist (\mathcal{RI}, CTP) eine kondensierte Repräsentation der Menge \mathcal{IB} .

Weil nach ([40]) folgendes gilt:

$$\forall r \in \mathcal{AR} : conf(r) = 1 \Rightarrow sup(r) = \max\{sup(r') \mid r' \in \mathcal{GB} \wedge r \in C(r')\}$$

$$\{r \in \mathcal{RR} \mid conf(r) = 1\} \subseteq \mathcal{GB}$$

schließen wir, dass (\mathcal{GB}, C) eine korrekte, vollständige und informative Repräsentation der starken exakten Regeln $\mathcal{AR}_{=1}$ ist

Es gilt auch nach ([40]), dass (\mathcal{IB}, C) eine vollständige Repräsentation der Menge $\mathcal{AR}_{<1}$ ist.

5.3.5 Die kondensierte Repräsentation der Assoziationsregeln nach [47]

Definition 5.15 (pseudo-abgeschlossen) Eine Menge $X \subseteq \mathcal{I}$ ist pseudo-abgeschlossen⁹ in einer Transaktionsdatenbank \mathcal{D} über \mathcal{I} genau dann, wenn folgendes gilt:

- $cl(X) \neq X$. (X ist also keine abgeschlossene Menge in \mathcal{D})
- Für jede echte Untermenge Y von X , die pseudo-abgeschlossen ist, gilt $cl(Y) \subset X$.

Die Menge aller häufigen pseudo-abgeschlossenen Mengen in \mathcal{D} wird durch \mathcal{PCF} gekennzeichnet. D.h. $\mathcal{PCF} = \{X \in \mathcal{F}(\mathcal{D}, s) \mid cl(X) \neq X \wedge (\forall Y \in \mathcal{PCF} : Y \subset X \Rightarrow cl(Y) \subset X)\}$. In ([47]) ist eine Basis \mathcal{DG} ¹⁰ für die Menge der starken exakten Assoziationsregeln $\mathcal{AR}_{=1}$ wie folgt definiert: $\mathcal{DG} = \{X \rightarrow cl(X) \setminus X \mid X \in \mathcal{PCF}\}$.

Weil $sup(cl(X), \mathcal{D}) = sup(X, \mathcal{D})$ für jede $X \subseteq \mathcal{I}$ gilt, ist $sup(r) = 1$ für jede Regel $r \in \mathcal{DG}$. Die Repräsentation (\mathcal{DG}, AA) ist eine vollständige Repräsentation der Menge $\mathcal{AR}_{=1}$. Aufgrund der Armstrongs Axiome *AA* ist die Konfidenz jeder aus dieser Repräsentation abgeleiteten Regel gleich 1. Außerdem gilt nach ([47]), dass \mathcal{DG} die kleinste Menge in \mathcal{AR} ist, aus der alle starken exakten Regeln abgeleitet werden können.

Beispiel 5.14 Betrachten wir die im linken Teil der Tabelle (5.4) dargestellte Transaktionsdatenbank \mathcal{D} , dann haben wir für den minimalen Support $s = 2$:

$$\mathcal{PCF}(\mathcal{D}, 2) = \{\emptyset, \{b, c, e\}, \{b, d, e\}\}.$$

Außerdem haben wir $cl(\emptyset) = \{b, e\}$ und $cl(\{b, c, e\}) = cl(\{b, d, e\}) = \{b, c, d, e\}$. Dementsprechend gilt $\mathcal{DG} = \{\emptyset \rightarrow \{b, e\}, \{b, c, e\} \rightarrow \{d\}, \{b, d, e\} \rightarrow \{c\}\}$. Wir möchten prüfen, ob die Regel $\{a, c\} \rightarrow \{b, d, e\}$ mithilfe der Repräsentation (\mathcal{DG}, AA) eine exakte Regel ist.

Aus der Regel $\{b, c, e\} \rightarrow \{d\} \in \mathcal{DG}$ schließen wir, dass $sup(\{b, c, d, e\}, \mathcal{D}) = 1$. Daraus ergibt sich, dass die Konfidenz der Regel $\{c\} \rightarrow \{b, d, e\}$ gleich 1 ist. Wenden wir das zweite Axiom auf diese Regel an, dann erhalten wir, dass $conf(\{a, c\} \rightarrow \{b, d, e\}) = 1$ ist. Also ist die Regel $\{a, c\} \rightarrow \{b, d, e\}$ exakt. Ob sie eine starke Regel ist, können wir

⁹Engl.: pseudo-closed

¹⁰Duquenne-Guigues basis

nur mithilfe von (\mathcal{DG}, AA) nicht entscheiden. Denn diese Repräsentation ist weder eine korrekte noch informative Repräsentation der Menge $\mathcal{AR}_{=1}$.

In ([47]) ist eine Basis \mathcal{PB}^{11} für die Menge der starken approximierten Regeln $\mathcal{AR}_{<1}$ wie folgt definiert: $\mathcal{PB} = \{X \rightarrow Y \setminus X \mid X, Y \in \mathcal{CF}(\mathcal{D}, s) \wedge X \subset Y \wedge \text{conf}(X \rightarrow Y \setminus X) > c\}$. Diese Basis kann durch (\mathcal{LB}, CTP) repräsentiert werden, wobei die Menge \mathcal{LB} Luxemburger transitive Reduktion von \mathcal{PB} heißt und wie folgt definiert ist:

$$\mathcal{LB} = \{X \rightarrow Y \setminus Y \in \mathcal{PB} \mid X \text{ ist eine maximale echte Teilmenge von } Y \in \mathcal{CF}\}.$$

Die Repräsentation (\mathcal{PB}, CCI) ist eine vollständige Repräsentation der Menge $\mathcal{AR}_{<1}$. In ([47]) wurde behauptet, dass diese Repräsentation auch korrekt und informativ sei. Dies ist in ([40]) widerlegt.

Basierend auf der folgenden in ([40]) bewiesenen Eigenschaft:

Für $X \subset Y$ ist $cl(X) = Y$ genau dann, wenn $X \rightarrow Y \setminus X$ ableitbar aus (\mathcal{DG}, AA) und $Y \in \mathcal{CF}$ ist.

und basierend darauf, dass (\mathcal{DG}, AA) eine vollständige Repräsentation von $\mathcal{AR}_{=1}$ und (\mathcal{PB}, CCI) eine vollständige Repräsentation von $\mathcal{AR}_{<1}$ ist, ist $(\mathcal{DG} \cup \mathcal{PB}, \{AA, CCI\})$ eine vollständige, korrekte und informative Repräsentation der Menge aller starken approximierten Regeln $\mathcal{AR}_{<1}$.

Um bei dieser Repräsentation festzustellen, ob die Regel $X \rightarrow Y \setminus X$ in $\mathcal{AR}_{<1}$ enthalten ist, ist eine Regel $X' \rightarrow Y' \setminus X'$ in \mathcal{PB} zu finden, so dass $X \rightarrow X' \setminus X$ und $Y \rightarrow Y' \setminus Y$ jeweils aus (\mathcal{DG}, AA) ableitbar ist. In diesem Fall gilt $cl(X) = X'$ und $cl(Y) = Y'$. Dann ist nach der Inferenz-Methode CCI zu schließen, dass $\text{sup}(X \rightarrow Y \setminus X) = \text{sup}(X' \rightarrow Y' \setminus X')$ und $\text{conf}(X \rightarrow Y \setminus X) = \text{conf}(X' \rightarrow Y' \setminus X')$ gilt. Existiert solche Regel $X' \rightarrow Y' \setminus X'$ in \mathcal{PB} nicht, dann ist $X \rightarrow Y \setminus X \notin \mathcal{AR}_{<1}$.

Wir fassen in der Tabelle (5.7) alle kondensierten Repräsentationen der Assoziationsregeln zusammen, die wir in diesem Abschnitt vorgestellt haben.

Regeln	Repräsentation	vollständig	korrekt	informativ
$\mathcal{AR}(\mathcal{D}, s, c)$	$(\mathcal{RR}(\mathcal{D}, s, c), C)$	ja	ja	nein
	$(\mathcal{MNR}(\mathcal{D}, s, c), C)$	ja	ja	ja
$\mathcal{AR}_{=1}(\mathcal{D}, s, c)$	$(\mathcal{GB}(\mathcal{D}, s, c), C)$	ja	ja	ja
	$(\mathcal{GB}(\mathcal{D}, s, c), CCI)$	ja	ja	ja
	$(\mathcal{DG}(\mathcal{D}, s, c), AA)$	ja	nein	nein
$\mathcal{AR}_{<1}(\mathcal{D}, s, c)$	$(\mathcal{IB}(\mathcal{D}, s, c), C)$	ja	nein	nein
	$(\mathcal{PB}(\mathcal{D}, s, c), CCI)$	ja	nein	nein
	$((\mathcal{PB} \cup \mathcal{DG})(\mathcal{D}, s, c), \{AA, CCI\})$	ja	ja	ja

Tabelle 5.7: Zusammenfassung der kondensierten Repräsentationen der Assoziationsregeln

¹¹Proper basis

6 Repräsentative Episode-Regeln

Episode-Regeln sind ein Mittel für die Beschreibung und Vorhersage des Verhaltens der Ereignis-Sequenzen. Sie informieren uns zum Beispiel über das Verhalten einer Ereignis-Sequenz \mathcal{S} in der folgenden Form: Folgt der Ereignis-Typ B dem Ereignis-Typ A sehr oft in \mathcal{S} , dann ist es sehr wahrscheinlich, dass der Ereignis-Typ F nach B (direkt oder indirekt) vorkommt. Dieser Sachverhalt wird syntaktisch durch $[A, B] \rightarrow [A, B, F]$ ausgedrückt. In Abhängigkeit davon, wie die Ausdrücke „sehr oft“ und „sehr wahrscheinlich“ quantitativ spezifiziert sind, werden die Episode-Regeln in interessante und in uninteressante Regeln eingeteilt. Im Abschnitt (6.1) werden wir nach ([9]) das Konzept „Episode-Regel“ formal definieren und einen Algorithmus zur Entdeckung aller interessanten Episode-Regeln in einer Ereignis-Sequenz vorstellen.

Die Anzahl der Episode-Regeln ist in vielen Fällen zu groß, wodurch die Analyse der entsprechenden Ereignis-Sequenz mit ihrer Hilfe sehr schwierig wird. Die Entwicklung einer Lösung für dieses Problem ist eines der Ziele dieser Diplomarbeit.

Um Ideen für das Finden einer Lösung zu bekommen, haben wir im Kapitel 5 das Konzept der kondensierten Repräsentation der interessanten Muster in Datenbanken vorgestellt und seine Techniken in Bezug auf die häufigen Mengen und auf die Assoziationsregeln ausführlich studiert. Im Hinblick auf unser Ziel sprechen zwei Gründe für diese Studie. Zum einen ist die Rolle der Assoziationsregeln in Transaktionsdatenbanken ähnlich der Rolle der Episode-Regeln in Ereignis-Sequenzen. Zum anderen ist die Nützlichkeit der Assoziationsregeln bei der Analyse der zugrundeliegenden Transaktionsdatenbank auch negativ abhängig von der Größe ihrer Anzahl.

In der Tat werden wir im Abschnitt (6.2) ein neues Konzept, nämlich das Konzept „repräsentative Episode-Regeln“, entwickeln, das dem im Abschnitt (5.3.3) vorgestellten Konzept „repräsentative Assoziationsregeln“ ähnlich ist. Es wird auch gezeigt, dass die Menge der repräsentativen Episode-Regeln eine korrekte und vollständige Repräsentation aller interessanten Episode-Regeln darstellt. Außerdem werden wir im Abschnitt (6.3) einen Algorithmus zur Entdeckung aller repräsentativen Episode-Regeln in einer Ereignis-Sequenz formulieren.

6.1 Episode-Regeln

Definition 6.1 (Episode-Regel) Für zwei Episoden α und β heißt der Ausdruck $\beta \rightarrow \alpha$ Episode-Regel genau dann, wenn $\beta \neq \alpha$ und β eine Unterepisode von α ist.

Jeder Episode-Regel $\beta \rightarrow \alpha$ werden in Bezug auf eine Ereignis-Sequenz \mathcal{S} und auf eine Zeitfensterbreite win zwei Werte zugeordnet. Der erste Wert ist $sup(\beta \rightarrow \alpha, \mathcal{S}, win) = sup(\alpha, \mathcal{S}, win)$ und heißt Support der Regel in \mathcal{S} in Bezug auf die Zeitfensterbreite win .

Der zweite Wert ist $\text{conf}(\beta \rightarrow \alpha, \mathcal{S}, \text{win}) = \text{sup}(\alpha, \mathcal{S}, \text{win}) / \text{sup}(\beta, \mathcal{S}, \text{win})$ und heißt Konfidenz der Regel in \mathcal{S} in Bezug auf die Zeitfensterbreite win .

Für zwei vorgegebene Zahlen $s \in \mathbb{N}$ und $c \in \mathbb{R}^+$ gilt eine Episode-Regel $\beta \rightarrow \alpha$ als interessant in einer Ereignis-Sequenz \mathcal{S} und in Bezug auf eine Zeitfensterbreite win genau dann, wenn $\text{sup}(\beta \rightarrow \alpha, \mathcal{S}, \text{win}) > s$ und $\text{conf}(\beta \rightarrow \alpha, \mathcal{S}, \text{win}) > c$ gilt. Die Menge aller interessanten Episode-Regeln in einer Ereignis-Sequenz \mathcal{S} wird durch $\mathcal{ER}(\mathcal{S}, \text{win}, s, c)$ gekennzeichnet. In **Algorithm (15)** ist eine Prozedur namens „computeEpisodeRules“ zur Berechnung der Menge $\mathcal{ER}(\mathcal{S}, \text{win}, s, c)$ formuliert. An dieser Prozedur ist zu erkennen, dass der Schwerpunkt der Berechnung aller Episode-Regeln die Berechnung der Menge $\mathcal{F}(\mathcal{S}, \text{win}, s)$, also der Menge aller häufigen Episoden, ist. Algorithmen zur Berechnung der Menge $\mathcal{F}(\mathcal{S}, \text{win}, s)$ wurden ausführlich im Abschnitt (3.1.2) und im Kapitel 4 studiert.

Algorithm 15 Eine Prozedur zur Berechnung der Menge $\mathcal{ER}(\mathcal{S}, \text{win}, s, c)$

```

1: function COMPUTEEPISODERULES( $\mathcal{S}, \text{win}, s, c$ )
2:   Berechne die Menge  $\mathcal{F}(\mathcal{S}, \text{win}, s)$ 
3:    $\mathcal{ER}(\mathcal{S}, \text{win}, s, c) \leftarrow \emptyset$ 
4:   for all  $\alpha \in \mathcal{F}(\mathcal{S}, \text{win}, s)$  do
5:     for all  $\beta \in U_\alpha \setminus \{\alpha\}$  do           ▷  $U_\alpha$  ist die Menge aller Unterepisoden von  $\alpha$ 
6:       if  $\text{sup}(\alpha, \mathcal{S}, \text{win}) / \text{sup}(\beta, \mathcal{S}, \text{win}) > c$  then
7:          $\mathcal{ER}(\mathcal{S}, \text{win}, s, c) \leftarrow \mathcal{ER}(\mathcal{S}, \text{win}, s, c) \cup \{\beta \rightarrow \alpha\}$ 
8:       end if
9:     end for
10:  end for
11:  return  $\mathcal{ER}(\mathcal{S}, \text{win}, s, c)$ 
12: end function

```

6.2 Repräsentative Episode-Regeln

Dieser Abschnitt widmet sich der Entwicklung einer kondensierten Repräsentation der Menge $\mathcal{ER}(\mathcal{S}, \text{win}, s, c)$. Der Kern der Arbeit besteht aus den folgenden Punkten:

- Die Unterepisoden-Beziehung wird mithilfe des Lemmas (6.1) durch die Teilmengen-Beziehung ausgedrückt.
- Eine Art der Vereinigung von zwei Episoden wird durch die Definition des Operators \sqcup (Definition (6.2)) ermöglicht. Basierend auf diesem Operator wird der Operator RC definiert (Definition (6.3)). Der Operator RC hat hier die Rolle zu spielen, die der Abdeckungs-Operator C bei der repräsentativen Assoziationsregeln spielt (Siehe die Abschnitte (5.3.1) und (5.3.3)).
- Der neue Begriff „repräsentative Episode-Regel“ wird durch den Operator RC definiert. Die Existenz solcher Episode-Regeln wird im Lemma (6.5) bewiesen.

- Es wird gezeigt, dass die Menge aller repräsentativen Episode-Regeln $\mathcal{RE}\mathcal{R}(\mathcal{S}, win, s, c)$ mit dem Operator RC als Inferenz-Methode eine korrekte und vollständige Repräsentation der Menge aller interessanten Episode-Regeln ist.

Zur Vereinfachung der Schreibweise werden die Mengen-Namen im Rest dieses Kapitels ohne Parameter verwendet, da es aus dem Kontext hervorgehen soll, um welche Parameter es sich handelt. Zum Beispiel wird die Menge $\mathcal{F}(\mathcal{S}, win, s)$ einfach als \mathcal{F} geschrieben. Außerdem werden $sup(\beta \rightarrow \alpha, \mathcal{S}, win)$ und $conf(\beta \rightarrow \alpha, \mathcal{S}, win)$ für eine Regel $(\beta \rightarrow \alpha) \in \mathcal{ER}$ als $sup(\beta \rightarrow \alpha)$ bzw. $conf(\beta \rightarrow \alpha)$ geschrieben.

Für eine Episode $\alpha \in \mathcal{F}$ wird die Menge aller Unterepisoden von α durch U_α gekennzeichnet. D.h. $U_\alpha = \{\beta \in \mathcal{F} \mid \beta \text{ ist eine Unterepisode von } \alpha\}$.

Es wird angenommen, dass die Menge \mathcal{F} mindestens zwei Episoden α und β enthält, für die $\alpha \neq \beta$ und $\beta \in U_\alpha$ gilt.

Das folgende Lemma zeigt, dass sich die Unterepisoden-Beziehung auf die Teilmengen-Beziehung zurückführen lässt.

Lemma 6.1 *Eine Episode $\beta = (V_\beta, R_\beta, f_\beta)$ ist in U_α genau dann, wenn $V_\beta \subseteq V_\alpha$, $R_\beta \subseteq R_\alpha$ und $\forall x \in V_\beta : f_\beta(x) = f_\alpha(x)$ gilt.*

Beweis: (\Rightarrow)

Sei $\beta = (V_\beta, R_\beta, f_\beta) \in U_\alpha$, dann existiert nach der Definition (3.7) eine injektive Funktion $g : V_\beta \rightarrow V_\alpha$ mit den folgenden Eigenschaften:

$$\forall x \in V_\beta : f_\beta(x) = f_\alpha(g(x)) \quad (1)$$

$$\forall x, y \in V_\beta : (x, y) \in R_\beta \Rightarrow (g(x), g(y)) \in R_\alpha \quad (2)$$

Da die Funktion g injektiv ist, kann man die Menge V_β mit der Menge $g(V_\beta) = \{x \in V_\alpha \mid \exists x' \in V_\beta : g(x') = x\} \subseteq V_\alpha$ identifizieren, indem man jedes $x \in V_\beta$ durch $g(x) \in g(V_\beta)$ ersetzt. Dies bedeutet einfach, dass man die Knoten aus V_β umbenannt hat. Nach der Ausführung dieses Schrittes gilt:

$$\forall x \in V_\beta : f_\beta(x) = f_\alpha(g(x)) = f_\alpha(x) \quad \text{wegen (1)}$$

$$\forall x, y \in V_\beta : (x, y) \in R_\beta \Rightarrow (g(x), g(y)) = (x, y) \in R_\alpha \quad \text{wegen (2)}$$

Also $R_\beta \subseteq R_\alpha$ und damit ist der erste Teil des Lemmas bewiesen.

(\Leftarrow)

Sei $\beta = (V_\beta, R_\beta, f_\beta)$ eine Episode, für die $V_\beta \subseteq V_\alpha$, $R_\beta \subseteq R_\alpha$ und $\forall x \in V_\beta : f_\beta(x) = f_\alpha(x)$ gilt. Wir definieren die Funktion $g : V_\beta \rightarrow V_\alpha$ wie folgt:

$\forall x \in V_\beta : g(x) = x$. Dann ist g offensichtlich eine injektive Funktion, die die Bedingung (1) erfüllt. Aus $R_\beta \subseteq R_\alpha$ ergibt sich die Gültigkeit der Bedingung (2). Da die injektive Funktion g die Bedingungen (1) und (2) erfüllt, ist $\beta \in U_\alpha$. \square

Der folgende Operator ist als Vereinigung von zwei Episoden zu verstehen.

Definition 6.2 (der Operator \sqcup) *Für eine Episode γ definieren wir den Operator \sqcup über der Menge U_γ wie folgt:*

$$\forall \alpha, \beta \in U_\gamma : \alpha \sqcup \beta = (V_\alpha \cup V_\beta, R_\alpha \cup R_\beta, f_{\gamma|_{V_\alpha \cup V_\beta}})$$

wobei $f_{\gamma|_{V_\alpha \cup V_\beta}} : V_\alpha \cup V_\beta \rightarrow E$ mit $\forall x \in V_\alpha \cup V_\beta : f_{\gamma|_{V_\alpha \cup V_\beta}}(x) = f_\gamma(x)$ ist.

Beispiel 6.1 Für eine Episode α mit $V_\alpha = \{A, B\}$, $R_\alpha = \{(A, A), (B, B), (B, A)\}$ und $f_\alpha = f_{id}$ und eine Episode β mit $V_\beta = \{A, C\}$, $R_\beta = \{(A, A), (C, C), (C, A)\}$ und $f_\beta = f_{id}$ gilt für die Episode $\delta = \alpha \sqcup \beta$: $V_\delta = \{A, B, C\}$, $R_\delta = \{(A, A), (B, B), (C, C), (B, A), (C, A)\}$ und $f_\delta = f_{id}$.

Das folgende Lemma zeigt ein paar Eigenschaften des Operators \sqcup .

Lemma 6.2 Der Operator \sqcup erfüllt folgendes:

1. Sind $\alpha, \beta \in U_\gamma$, dann ist $\alpha \sqcup \beta \in U_\gamma$. Dies bedeutet, dass die Menge U_γ abgeschlossen gegen den Operator \sqcup ist.
2. Ist $\beta \in U_\alpha$ und ist $\alpha \in U_\gamma$, dann ist $\beta \in U_\gamma$. Dies bedeutet, dass die Unterepisoden-Beziehung eine transitive Beziehung ist.
3. Ist $\gamma = \alpha \sqcup \beta$, dann sind $\alpha, \beta \in U_\gamma$.

Beweis: Die Korrektheit von (1) und (2) ergibt sich direkt aus dem Lemma (6.1) und der Definition (6.2).

(3): Nehmen wir an, dass $\alpha, \beta \in U_\delta$ für eine Episode δ gilt, dann haben wir nach der Definition des Operators \sqcup :

$$V_\gamma = V_\alpha \cup V_\beta \quad (i)$$

$$R_\gamma = R_\alpha \cup R_\beta \quad (ii)$$

$$f_\gamma : V_\alpha \cup V_\beta \rightarrow E \quad \text{mit} \quad f_\gamma(x) = f_{\delta|_{V_\alpha \cup V_\beta}}(x) \quad (iii)$$

Aus (i) und (ii) haben wir $V_\alpha \in V_\gamma$ und $R_\alpha \in U_\gamma$. Außerdem haben wir:

$f_{\gamma|_{V_\alpha}} = f_{\delta|_{V_\alpha}} = f_\alpha$, da $\alpha \in U_\delta$ ist. Nach dem Lemma (6.1) schließen wir also $\alpha \in U_\gamma$. Aus den selben Argumenten gilt $\beta \in U_\gamma$. \square

Definition 6.3 (der Operator RC) Der Operator RC ist für eine Episode-Regel $(\gamma \rightarrow \delta)$ wie folgt definiert:

$$RC(\gamma \rightarrow \delta) = \{\gamma \sqcup \beta \rightarrow \alpha \mid \gamma \in U_\alpha \wedge \beta \in U_\alpha \wedge \alpha \in U_\delta\}$$

Beispiel 6.2 Für die Episode-Regel $[A, C] \rightarrow [A, B, C, A]$ gilt:

$$\begin{aligned} RC([A, C] \rightarrow [A, B, C, A]) &= \{[A, C] \rightarrow [A, B, C], \\ &\quad [A, C] \rightarrow [A, C, A], \\ &\quad [A, C] \rightarrow [A, B, C, A], \\ &\quad [A, B, C] \rightarrow [A, B, C, A], \\ &\quad [A, C, A] \rightarrow [A, B, C, A]\}. \end{aligned}$$

Das folgende Beispiel zeigt, dass eine Episode-Regel von mehr als einer anderen Episode-Regel durch die Anwendung des RC -Operators generiert werden kann

Beispiel 6.3 Für die Episode-Regel $[A, C] \rightarrow [A, C, D]$ gilt zum Beispiel:

$$([A, C] \rightarrow [A, C, D]) \in RC([A] \rightarrow [A, C, D])$$

$$([A, C] \rightarrow [A, C, D]) \in RC([A, C] \rightarrow [A, C, D, C])$$

Das folgende Lemma zeigt eine wichtige Eigenschaft des RC -Operators in Bezug auf die interessanten Episode-Regeln, nämlich, dass die Anwendung des RC -Operators auf eine interessante Episode-Regel nur interessante Episode-Regeln liefert.

Lemma 6.3 Für jede Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{ER}$ gilt $RC(\gamma \rightarrow \delta) \subseteq \mathcal{ER}$.

Beweis: Sei $(\gamma' \rightarrow \delta') \in RC(\gamma \rightarrow \delta)$.

Da $(\gamma' \rightarrow \delta') \in RC(\gamma \rightarrow \delta)$ ist, existieren nach der Definition (6.3) $\alpha \in U_\delta$ und $\beta \in U_\alpha$ die folgendes erfüllen:

$$\gamma \in U_\alpha \quad (1)$$

$$\gamma' = \gamma \sqcup \beta \quad (2)$$

$$\delta' = \alpha \quad (3)$$

Aus (1), (2), $\beta \in U_\alpha$ und dem Lemma (6.2/1) ergibt sich $\gamma' \in U_\alpha$. Daraus und aus (3) ist es zu schließen, dass $(\gamma' \rightarrow \delta')$ eine Episode-Regel ist. Da $\delta' = \alpha \in U_\delta$ ist, haben wir nach dem Lemma(3.3):

$$\sup(\delta', \mathcal{S}, \text{win}) \geq \sup(\delta, \mathcal{S}, \text{win}) > s \quad (4)$$

Aus (2) und dem Lemma (6.2/3) schließen wir $\gamma \in U_{\gamma'}$. Daraus ergibt sich :

$$\sup(\gamma, \mathcal{S}, \text{win}) \geq \sup(\gamma', \mathcal{S}, \text{win}) \quad \Rightarrow$$

$$1/\sup(\gamma', \mathcal{S}, \text{win}) \geq 1/\sup(\gamma, \mathcal{S}, \text{win}) \quad \Rightarrow$$

$$\sup(\delta', \mathcal{S}, \text{win})/\sup(\gamma', \mathcal{S}, \text{win}) \geq \sup(\delta, \mathcal{S}, \text{win})/\sup(\gamma, \mathcal{S}, \text{win}) \quad \Rightarrow$$

$$\text{conf}(\gamma' \rightarrow \delta') \geq \text{conf}(\gamma \rightarrow \delta) > c \quad (5)$$

Also ist $\gamma' \rightarrow \delta'$ eine Episode-Regel, die (4) und (5) erfüllt. Also ist $\gamma' \rightarrow \delta' \in \mathcal{ER}$. \square

Wann eine Episode-Regel durch die Anwendung des RC -Operators auf eine andere Episode-Regel erzeugt werden kann, zeigt das folgende Lemma, das eine zentrale Rolle im Rest dieses Abschnittes spielt.

Lemma 6.4 Für die Regeln $\gamma' \rightarrow \delta'$ und $\gamma \rightarrow \delta$ gilt :

$$(\gamma' \rightarrow \delta') \in RC(\gamma \rightarrow \delta) \Leftrightarrow \delta' \in U_\delta \wedge \gamma \in U_{\gamma'}$$

Beweis:(\Rightarrow)

Nach der Definition vom RC -Operator gilt folgendes:

$$\delta' \in U_\delta \quad (1)$$

$$\gamma \in U_{\delta'} \quad (2)$$

$$\exists \beta : \beta \in U_{\delta'} \wedge \gamma' = \gamma \sqcup \beta \quad (3)$$

Aus (2), (3) und dem Lemma (6.2/3) ergibt sich $\gamma \in U_{\gamma'}$. Daraus und aus (1) ergibt sich die Korrektheit des ersten Teiles.

(\Leftarrow)

Definieren wir die Episode β wie folgt:

$$V_\beta = V_{\gamma'} \setminus V_\gamma$$

$$R_\beta = R_{\gamma'} \setminus R_\gamma$$

$$f_\beta = f_{\gamma'|V_\beta}$$

dann haben wir:

$$\gamma' = \gamma \sqcup \beta \quad (4) \quad (\text{nach Definition(6.3)})$$

$$\beta \in U_{\gamma'} \quad (5) \quad (\text{nach Lemma (6.1)})$$

Aus (5), $\gamma' \in U_{\delta'}$ und dem Lemma (6.2/2) ergibt sich $\beta \in U_{\delta'}$. Aus $\gamma \in U_{\gamma'}$, $\gamma' \in U_{\delta'}$ und aus dem Lemma (6.2/2) ergibt sich $\gamma \in U_{\delta'}$. Also schließen wir aus (4), $\gamma \in U_{\delta'}$, $\beta \in U_{\delta'}$ und $\delta' \in U_{\delta}$, dass $(\gamma' \rightarrow \delta') \in RC(\gamma \rightarrow \delta)$ gilt. \square

Basierend auf dem RC -Operator definieren wir den Begriff „repräsentative Episode-Regel“.

Definition 6.4 (Repräsentative Episode-Regel) *Eine Episode-Regel $\gamma \rightarrow \delta$ heißt repräsentative Episode-Regel genau dann, wenn keine Episode-Regel $\beta \rightarrow \alpha$ existiert, so dass $(\gamma \rightarrow \delta) \neq (\beta \rightarrow \alpha) \wedge (\gamma \rightarrow \delta) \in RC(\beta \rightarrow \alpha)$ gilt.*

Eine repräsentative Episode-Regel ist also eine Episode-Regel, die sich durch die Anwendung des RC -Operators auf eine andere Episode-Regel nicht generieren lässt. Sie heißt repräsentativ, weil sie alle Episode-Regeln repräsentiert, die man durch die Anwendung des RC -Operators auf sie generieren kann. Also repräsentiert eine repräsentative Episode-Regel $\gamma \rightarrow \delta$ alle in der Menge $RC(\gamma \rightarrow \delta)$ enthaltenen Episode-Regeln.

Das folgende Lemma zeigt die Existenz repräsentativer Episode-Regeln.

Lemma 6.5 *Es gibt mindestens zwei Episoden $\gamma, \delta \in \mathcal{F}$, so dass $\gamma \rightarrow \delta$ eine repräsentative Episode-Regel ist.*

Beweis: Wir definieren die Relation $P \subseteq \mathcal{F} \times \mathcal{F}$ wie folgt:

$$\forall \alpha, \beta \in \mathcal{F} : (\beta, \alpha) \in P \Leftrightarrow \beta \in U_{\alpha}$$

Nach dem Lemma (6.2/2) ist die Relation P transitiv. Außerdem gilt

$$\forall \alpha \in \mathcal{F} : (\alpha, \alpha) \in P$$

$$\forall \alpha, \beta \in \mathcal{F} : (\beta, \alpha) \in P \wedge (\alpha, \beta) \in P \Rightarrow \beta = \alpha$$

offensichtlich. Also ist P eine Ordnungsrelation. Es gilt:

$$|\mathcal{F}| < \infty \quad (1)$$

$$\forall \delta \in \mathcal{F} : 1 < |V_{\delta}| < \infty \quad (2) \quad (\text{Definition einer Episode})$$

Aus (1), (2) und daraus, dass P eine Ordnungsrelation ist, ergibt sich, dass eine Episode $\delta \in \mathcal{F}$ existiert, die folgendes erfüllt:

$$\neg \exists \delta_1 \in \mathcal{F} : (\delta, \delta_1) \in P \quad (3)$$

Sei γ die kleinste Unterepisode von δ . D.h. γ erfüllt folgendes:

$$\gamma \neq \delta \wedge (\gamma, \delta) \in P \wedge \neg \exists \gamma_1 : (\gamma_1, \gamma) \in P \wedge (\gamma_1, \delta) \in P \quad (4)$$

Nehmen wir an, dass eine Episode-Regel $(\gamma_1 \rightarrow \delta_1) \in ER$ existiert, für die folgendes gilt:

$$(\gamma_1 \rightarrow \delta_1) \neq (\gamma \rightarrow \delta) \wedge (\gamma \rightarrow \delta) \in RC(\gamma_1 \rightarrow \delta_1)$$

dann muss nach dem Lemma (6.4) $\delta \in U_{\delta_1} \wedge \gamma_1 \in U_{\gamma}$ gelten. Dies ist ein Widerspruch zu (3) und (4). D.h. unsere Annahme ist falsch. Dies bedeutet nach der Definition (6.4), dass $\gamma \rightarrow \delta$ eine repräsentative Regel ist. \square

In der folgenden Definition wird das Konzept „repräsentative Episode-Regeln“ auf interessante Episode-Regeln eingeschränkt.

Definition 6.5 (Interessante repräsentative Episode-Regel) *Eine Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{ER}$ heißt interessante repräsentative Episode-Regel genau dann, wenn es*

keine Episode-Regel $(\gamma' \rightarrow \delta') \in \mathcal{ER}$ gibt, so dass $(\gamma' \rightarrow \delta') \neq (\gamma \rightarrow \delta)$ und $(\gamma \rightarrow \delta) \in RC(\gamma' \rightarrow \delta')$ gilt.

Die Menge aller interessanten repräsentativen Episode-Regeln wird durch $\mathcal{REER}(\mathcal{S}, win, s, c)$ oder einfach durch \mathcal{REER} gekennzeichnet.

Das folgende Lemma liefert uns ein sehr wichtiges Resultat, nämlich, dass sich jede interessante Episode-Regel durch die Anwendung des RC -Operators auf eine repräsentative interessante Episode-Regel erzeugen lässt.

Lemma 6.6 *Für jede Episode-Regel $(\beta \rightarrow \alpha) \in \mathcal{ER}$ gibt es eine repräsentative Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{REER}$, so dass $(\beta \rightarrow \alpha) \in RC(\gamma \rightarrow \delta)$ gilt.*

Beweis: Falls $(\beta \rightarrow \alpha) \in \mathcal{REER}$ ist, dann trifft das Lemma zu, da $(\beta \rightarrow \alpha) \in RC(\beta \rightarrow \alpha)$ gilt. Also betrachten wir den Fall: $(\beta \rightarrow \alpha) \notin \mathcal{REER}$.

In der Abhängigkeit von β definieren wir die Mengen $\overline{\mathcal{ER}}_\beta$ und \mathcal{ER}_β^- wie folgt:

$$\overline{\mathcal{ER}}_\beta = \{(\beta \rightarrow \alpha) \in \mathcal{ER} \mid \neg \exists (\beta \rightarrow \delta) \in \mathcal{ER} : \alpha \in U_\delta \setminus \{\delta\}\}$$

$$\mathcal{ER}_\beta^- = \{(\beta \rightarrow \alpha) \in \mathcal{ER} \mid \forall (\beta \rightarrow \delta) \in \mathcal{ER} : \alpha \in U_\delta \setminus \{\delta\} \Rightarrow sup(\beta \rightarrow \alpha) = sup(\beta \rightarrow \delta)\}$$

Fall 1: $(\beta \rightarrow \alpha) \in \overline{\mathcal{ER}}_\beta$

Wenn wir in Bezug auf die Unterepisoden-Beziehung die kleinste Episode $\gamma \in U_\delta$ mit der Eigenschaft $sup(\alpha, \mathcal{S}, win)/sup(\gamma, \mathcal{S}, win) > c$ wählen, dann gilt nach dem Lemma (6.4) $(\beta \rightarrow \alpha) \in RC(\gamma \rightarrow \alpha)$ und $(\gamma \rightarrow \alpha) \in \mathcal{REER}$.

Fall 2: $(\beta \rightarrow \alpha) \notin \overline{\mathcal{ER}}_\beta$

In diesem Fall gibt es eine Episode-Regel $(\beta \rightarrow \delta) \in \mathcal{ER}$ mit $\alpha \in U_\delta \setminus \{\delta\}$. Also ist die Menge $\mathcal{F}_\alpha = \{\delta \in \mathcal{F} \mid \alpha \in U_\delta \wedge \alpha \neq \delta \wedge (\beta \rightarrow \delta) \in \mathcal{ER}\} \neq \emptyset$.

In der Abhängigkeit von $sup(\beta \rightarrow \alpha)$ ist zwischen zwei Fällen zu unterscheiden:

Fall 2.1: $(\beta \rightarrow \alpha) \in \mathcal{ER}_\beta^-$

Wir wählen in Bezug auf die Unterepisoden-Beziehung die größte Episode δ in \mathcal{F}_α und die kleinste Episode γ in U_β mit der Eigenschaft: $sup(\delta, \mathcal{S}, win)/sup(\gamma, \mathcal{S}, win)$. Nach unserem Fall gilt $sup(\delta, \mathcal{S}, win) = sup(\alpha, \mathcal{S}, win)$ und somit $(\gamma \rightarrow \delta) \in \mathcal{ER}$. Es gilt nach dem Lemma (6.4) $(\beta \rightarrow \alpha) \in RC(\gamma \rightarrow \delta)$

Annahme: $(\gamma \rightarrow \delta) \notin \mathcal{REER}$.

Dies bedeutet, dass es nach der Definition (6.5) eine Episode-Regel $(\gamma' \rightarrow \delta') \in \mathcal{ER}$ mit $(\gamma' \rightarrow \delta') \neq (\gamma \rightarrow \delta)$ und $(\gamma \rightarrow \delta) \in RC(\gamma' \rightarrow \delta')$ gibt. Nach dem Lemma (6.4) gilt:

$$\delta \in U_{\delta'} \quad (1)$$

$$\gamma' \in U_\gamma \quad (2)$$

Da $(\gamma' \rightarrow \delta') \neq (\gamma \rightarrow \delta)$ ist, muss $\delta \neq \delta'$ oder $\gamma \neq \gamma'$ gelten.

Falls $\delta \neq \delta'$ ist, dann gilt nach der Wahl von δ und (1): $sup(\alpha, \mathcal{S}, win) > sup(\delta, \mathcal{S}, win)$ und somit ist $(\beta \rightarrow \alpha) \notin \mathcal{ER}_\beta^-$. Also muss $\delta = \delta'$ sein.

Falls $\gamma \neq \gamma'$ ist, dann gilt mit $\delta = \delta'$ und nach der Wahl von γ' und (2):

$con(\gamma' \rightarrow \delta') = sup(\delta, \mathcal{S}, win)/sup(\gamma', \mathcal{S}, win) \leq c$ und somit ist $(\gamma' \rightarrow \delta') \notin \mathcal{ER}$.

Es kann also nur $(\gamma' \rightarrow \delta') = (\gamma \rightarrow \delta)$ sein und somit ist unsere Annahme falsch. Also gilt $(\gamma \rightarrow \delta) \in \mathcal{REER}$.

Fall 2.2: $(\beta \rightarrow \alpha) \notin \mathcal{ER}_\beta^-$

In diesem Fall existiert also eine Episode-Regel $(\beta \rightarrow \delta) \in \mathcal{ER}$ mit $sup(\beta \rightarrow \alpha) \neq$

$sup(\beta \rightarrow \delta)$ und $\alpha \in U_\delta$.

Falls $(\beta \rightarrow \delta) \in \overline{\mathcal{ER}}_\beta$ oder $(\beta \rightarrow \delta) \in \mathcal{ER}_\beta^-$ ist, dann existiert nach dem Fall 1 bzw. dem Fall 2.1 eine Episode-Regel $(\beta' \rightarrow \delta') \in \mathcal{RE}\mathcal{R}$, so dass $(\beta \rightarrow \delta) \in RC(\beta' \rightarrow \delta')$ gilt.

Aus $\alpha \in U_\delta$ und $\delta \in U_{\delta'}$ ist $\alpha \in U_{\delta'}$ zu schließen. Daraus und aus $\beta' \in U_\beta$ gilt $(\beta \rightarrow \alpha) \in RC(\beta' \rightarrow \delta')$.

Falls die Regel $(\beta \rightarrow \delta) \notin \mathcal{ER}_\beta^-$ ist, wenden wir wiederum den Fall 2.2 auf sie an.

Weil die Menge der Oberepisoden von α endlich ist, erhalten wir nach vielen endlichen Schritten folgendes:

$$\begin{aligned} (\beta_n \rightarrow \delta_n) &\in \mathcal{RE}\mathcal{R} \\ (\beta_{n-1} \rightarrow \delta_{n-1}) &\in RC(\beta_n \rightarrow \delta_n) \\ (\beta_{n-2} \rightarrow \delta_{n-2}) &\in RC(\beta_{n-1} \rightarrow \delta_{n-1}) \\ &\vdots \\ (\beta \rightarrow \delta) &\in RC(\beta_1 \rightarrow \delta_1) \end{aligned}$$

Nach dem Lemma (6.2)/2 und dem Lemma (6.4) gilt $(\beta \rightarrow \alpha) \in RC(\beta_n \rightarrow \delta_n)$. \square

Nun können wir basierend auf dem Lemma (6.3) und dem Lemma (6.6) unser zentrales Resultat formulieren.

Lemma 6.7 ($\mathcal{RE}\mathcal{R}, RC$) *ist eine korrekte und vollständige Repräsentation der Menge aller interessanten Episode-Regeln \mathcal{ER} . Das heißt:*

$$\mathcal{ER} = \bigcup_{(\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R}} RC(\gamma \rightarrow \delta)$$

Beweis: Für jede beliebige Episode-Regel $(\beta \rightarrow \alpha) \in \mathcal{ER}$ existiert nach dem Lemma (6.6) eine repräsentative Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R}$, so dass $(\beta \rightarrow \alpha) \in RC(\gamma \rightarrow \delta)$ gilt. Daraus schließen wir folgendes:

$$\mathcal{ER} \subseteq \bigcup_{(\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R}} RC(\gamma \rightarrow \delta) \quad (1)$$

Auf der andern Seite haben wir nach dem Lemma (6.3) folgendes:

$$\forall (\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R} : RC(\gamma \rightarrow \delta) \subseteq \mathcal{ER}$$

Daraus und aus der Tatsache, dass die Menge $\mathcal{RE}\mathcal{R}$ eine endliche Menge ist, ergibt sich folgendes:

$$\mathcal{ER} \supseteq \bigcup_{(\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R}} RC(\gamma \rightarrow \delta) \quad (2)$$

Aus (1) und (2) ergibt sich die Korrektheit des Lemmas. \square

Also kann man nach dem obigen Lemma alle interessanten Episode-Regeln mithilfe des RC -Operators erzeugen, falls man alle interessanten repräsentativen Episode-Regeln hat. Außerdem kann man nach dem Lemma (6.3) den Support und die Konfidenz jeder Episode-Regel $(\beta \rightarrow \alpha) \in \mathcal{ER} \setminus \mathcal{RE}\mathcal{R}$ wie folgt abschätzen:

$$\begin{aligned} sup(\beta \rightarrow \alpha) &= \max(\{sup(\delta \rightarrow \gamma) \mid (\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R} \wedge (\beta \rightarrow \alpha) \in RC(\gamma \rightarrow \delta)\}) \\ conf(\beta \rightarrow \alpha) &= \max(\{conf(\delta \rightarrow \gamma) \mid (\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R} \wedge (\beta \rightarrow \alpha) \in RC(\gamma \rightarrow \delta)\}) \end{aligned}$$

6.3 Algorithmus zur Berechnung repräsentativer Episode-Regeln

In diesem Abschnitt werden ein paar Eigenschaften der repräsentativen Episode-Regeln untersucht. Danach wird anhand dieser Eigenschaften ein Algorithmus zur Generierung der Menge $\mathcal{RE}\mathcal{R}$ aus der Menge \mathcal{F} formuliert. Die Arbeitsweise dieses Algorithmus ist ähnlich der Arbeitsweise des Algorithmus zur Generierung repräsentativer Assoziationsregeln ([41], [42]).

Definition 6.6 (maxSup) Für eine Episode $\alpha \in \mathcal{F}$ ist der Wert $(\maxSup)_\alpha$ wie folgt definiert:

$$(\maxSup)_\alpha = \begin{cases} \max(\{sup(\delta, \mathcal{S}, win) \mid \delta \in \mathcal{F} \wedge \alpha \in U_\delta \setminus \{\delta\}\}) & ; \exists \delta : \alpha \in U_\delta \setminus \{\delta\} \\ 0 & ; \neg \exists \delta \in \mathcal{F} : \alpha \in U_\delta \setminus \{\delta\} \end{cases}$$

Offensichtlich gilt $sup(\alpha, \mathcal{S}, win) \geq (\maxSup)_\alpha$ für jede Episode $\alpha \in \mathcal{F}$.

Lemma 6.8 Gilt $sup(\delta, \mathcal{S}, win) = (\maxSup)_\delta$ für eine Episode $\delta \in \mathcal{F}$, dann gilt $(\gamma \rightarrow \delta) \notin \mathcal{RE}\mathcal{R}$ für jede Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{ER}$.

Beweis: Aus $sup(\delta, \mathcal{S}, win) = (\maxSup)_\delta$ und nach der Definition (6.6) ist zu schließen, dass es mindestens eine echte Oberepisode $\delta' \in \mathcal{F}$ von δ mit $sup(\delta, \mathcal{S}, win) = sup(\delta', \mathcal{S}, win)$ gibt. Dies bedeutet nach der Definition (6.3), dass für jede Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{ER}$ folgendes gilt: $(\gamma \rightarrow \delta) \in RC(\gamma \rightarrow \delta')$. Dies bedeutet nach der Definition (6.5), dass $(\gamma \rightarrow \delta) \notin \mathcal{RE}\mathcal{R}$ ist. \square

Lemma 6.9 Eine Episode-Regel $(\gamma \rightarrow \delta) \in \mathcal{ER}$ ist in $\mathcal{ER}\mathcal{E}$ enthalten, wenn die folgenden Bedingungen erfüllt sind:

1. $sup(\delta, \mathcal{S}, win) > (\maxSup)_\delta$
2. $(\maxSup)_\delta / sup(\gamma, \mathcal{S}, win) \leq c$
3. $\neg \exists \gamma' \in \mathcal{F} : \gamma' \in U_\gamma \setminus \{\gamma\} \wedge (\gamma' \rightarrow \delta) \in \mathcal{ER}$

Beweis: Nach der Definition (6.6) besagen die erste und die zweite Bedingung zusammen, dass es keine echte Oberepisode $\delta' \in \mathcal{F}$ von δ gibt, so dass $(\gamma \rightarrow \delta') \in \mathcal{ER}$ gilt. Dies und die dritte Bedingung des Lemmas besagen zusammen folgendes:

$$\neg \exists \delta', \gamma' \in \mathcal{F} : \delta \in U_{\delta'} \setminus \{\delta'\} \wedge \gamma' \in U_\gamma \setminus \{\gamma\} \wedge (\gamma' \rightarrow \delta') \in \mathcal{ER}$$

Also gibt es keine Episode-Regel $(\gamma' \rightarrow \delta') \in \mathcal{ER}$ mit $(\gamma \rightarrow \delta) \neq (\gamma' \rightarrow \delta')$ und $(\gamma \rightarrow \delta) \in RC(\gamma' \rightarrow \delta')$. Also gilt nach der Definition (6.5) $(\gamma \rightarrow \delta) \in \mathcal{RE}\mathcal{R}$. \square

Basierend auf den Lemmata (6.8) und (6.9) ist in **Algorithm (16)** eine Prozedur zur Generierung der Menge $\mathcal{RE}\mathcal{R}$ aus der Menge \mathcal{F} formuliert. Dabei sind folgende Vereinbarungen getroffen: Die Länge der längsten Episode in \mathcal{F} ist durch L bezeichnet. Das heißt $L = \max(\{|\delta| \mid \delta \in \mathcal{F}\})$. Die Menge aller Episoden der Länge l mit $1 \leq l \leq L$ ist

Algorithm 16 Eine Prozedur zur Berechnung der Menge $\mathcal{RE}\mathcal{R}(\mathcal{S}, win, s, c)$ aus der Menge $\mathcal{F}(\mathcal{S}, win, s)$

```

1: function COMPUTERER( $\mathcal{F}, c$ )
2:    $\mathcal{RE}\mathcal{R} \leftarrow \emptyset$ 
3:   for  $l \leftarrow L, 2$  do
4:     for all  $\delta \in \mathcal{F}[l]$  do
5:        $(maxSup)_\delta \leftarrow 0$ 
6:       if  $l < L$  then
7:          $(maxSup)_\delta \leftarrow \max(\{sup(\delta', \mathcal{S}, win) \mid \delta' \in \mathcal{F}[l+1] \wedge \delta \in U_{\delta'}\} \cup \{0\})$ 
8:       end if
9:       if  $sup(\delta, \mathcal{S}, win) > (maxSup)_\delta$  then
10:         $k \leftarrow 1$ 
11:         $U_\delta^k \leftarrow \{\gamma \in U_\delta \mid |\gamma| = k\}$ 
12:        while  $U_\delta^k \neq \emptyset \wedge k < l$  do
13:          for all  $\gamma \in U_\delta^k$  do
14:            if  $sup(\delta, \mathcal{S}, win)/sup(\gamma, \mathcal{S}, win) > c$  then
15:              if  $(maxSup)_\delta/sup(\gamma, \mathcal{S}, win) \leq c$  then
16:                 $\mathcal{RE}\mathcal{R} \leftarrow \mathcal{RE}\mathcal{R} \cup \{\gamma \rightarrow \delta\}$ 
17:              end if
18:               $U_\delta^k \leftarrow U_\delta^k \setminus \{\gamma\}$ 
19:            end if
20:          end for
21:           $k \leftarrow k + 1$ 
22:           $U_\delta^k \leftarrow \{\gamma \in U_\delta \mid |\gamma| = k \wedge (\forall \gamma' \in U_\gamma : |\gamma'| = k - 1 \Rightarrow \gamma' \in U_\delta^{k-1})\}$ 
23:        end while
24:      end if
25:    end for
26:  end for
27:  return  $\mathcal{RE}\mathcal{R}$ 
28: end function

```

durch $\mathcal{F}[l]$ gekennzeichnet. Die Menge U_δ^k bezeichnet eine Menge von Unterepisoden von δ , deren Länge k ist.

Die Prozedur durchläuft die Menge \mathcal{F} in Blöcken, wobei jeder Block die Menge $\mathcal{F}[l]$ darstellt. Für jede Episode $\delta \in \mathcal{F}[l]$ mit $1 < l \leq L$ werden alle möglichen interessanten repräsentativen Episode-Regeln ($\gamma \rightarrow \delta$) wie folgt generiert: Zuerst wird der Wert $(maxSup)_\delta$ nach der Definition (6.6) berechnet (Zeilen 5-8). Ist $sup(\delta, \mathcal{S}, win) = (maxSup)_\delta$, dann gibt es nach dem Lemma (6.8) keine Episode-Regel ($\gamma \rightarrow \delta$) in $\mathcal{RE}\mathcal{R}$ und deshalb braucht die Prozedur die Episode δ nicht mehr zu betrachten.

Ist $sup(\delta, \mathcal{S}, win) > (maxSup)_\delta$, dann werden Unterepisoden γ von δ nach ihren Längen k mit $k = 1, 2, \dots, |\delta| - 1$ in der folgenden Weise betrachtet:

Für $\underline{k = 1}$ werden alle Unterepisoden γ mit $|\gamma| = 1$ in die Menge U_δ^1 aufgenommen (Zeile

11). Danach wird für jede Episode $\gamma \in U_\delta^1$ geprüft, ob $sup(\delta, \mathcal{S}, win)/sup(\gamma, \mathcal{S}, win) > c$, was $(\gamma \rightarrow \delta) \in \mathcal{ER}$ bedeutet, und $(maxSup)_\delta/sup(\gamma, \mathcal{S}, win) \leq c$ gelten. Erfüllt γ diese beiden Bedingungen, dann gilt nach dem Lemma (6.9) $(\gamma \rightarrow \delta) \in \mathcal{RER}$, wenn man $sup(\delta, \mathcal{S}, win) > (maxSup)_\delta$ für die Erfüllung der ersten Bedingung und $|\gamma| = 1$ für die Erfüllung der dritten Bedingung des Lemmas vor Augen hält. Anschließend wird die Episode γ aus der Menge U_δ^k entfernt (Zeile 18), falls für sie $(\gamma \rightarrow \delta) \in \mathcal{ER}$ gilt.

Für $k > 1$ enthält die Menge U_δ^{k-1} nach dem Verlassen des Rumpfes der **for all**-Schleife (Zeile 13) in der Zeile 20 diejenigen Unterepisoden γ' von δ , deren Länge $k - 1$ ist, und für die $(\gamma' \rightarrow \delta) \notin \mathcal{ER}$ gilt. Alle Unterepisoden von δ , deren Länge k ist, und für die gilt, dass jede ihrer Unterepisoden der Länge $k - 1$ in U_δ^{k-1} enthalten ist, werden in die Menge U_δ^k aufgenommen. Also enthält die Menge U_δ^k jetzt alle Unterepisoden γ von δ mit der Eigenschaften: $|\gamma| = k$ und $\forall \gamma' \in U_\gamma \setminus \{\gamma\} : (\gamma' \rightarrow \delta) \notin \mathcal{ER}$.

Für jede Episode γ in U_δ^k wird wiederum (**for all**-Schleife in der Zeile 13) geprüft, ob sie die Bedingungen $sup(\delta, \mathcal{S}, win)/sup(\gamma, \mathcal{S}, win)$ und $(maxSup)_\delta/sup(\gamma, \mathcal{S}, win) \leq c$ erfüllt. Kann sie diese beiden Bedingungen erfüllen, dann gilt nach der Konstruktion der Menge U_δ^γ und dem Lemma (6.9), dass $(\gamma \rightarrow \delta) \in \mathcal{RER}$ ist.

Also werden alle möglichen interessanten repräsentativen Episode-Regeln $(\gamma \rightarrow \delta)$ für jede Episode $\delta \in \mathcal{F}$ mit $|\delta| > 1$ systematisch auf die gerade erklärte Weise generiert. Deshalb erzeugt unsere Prozedur „computeRER“ alle interessanten repräsentativen Episode-Regeln, also die Menge $\mathcal{RER}(\mathcal{S}, win, s, c)$.

Die durchgeführten Experimenten mit diesem Algorithmus werden im Abschnitt (7.3) diskutiert.

7 Implementierung und Experimente

Alle im Rahmen dieser Diplomarbeit implementierten Algorithmen stehen zur Verfügung als YALE-Plugin, das im Abschnitt (7.4) beschrieben wird.

Die im Laufe dieses Kapitels berichteten Experimente wurden auf einem Rechner mit dem Betriebssystem Windows XP durchgeführt. Der Rechner hat einen 1.5 Gigahertz Prozessor und verfügt über 512 Megabyte Hauptspeicher.

7.1 FP-growth -Algorithmus

Im Abschnitt (2.2.3) wurde der FP-growth-Algorithmus ausführlich studiert. Außerdem wurde auf der Seite (18) und mithilfe der in **Algorithm (2)** auf der Seite (19) formulierten Prozedur gezeigt, wie die anfängliche FP-tree-Datenstruktur der zugrundeliegenden Transaktionsdatenbank \mathcal{D} zu konstruieren bzw. zu implementieren ist. Der wesentliche Kern der Implementierung des FP-growth-Algorithmus besteht aber in der Antwort auf die Frage, wie sich die bedingte Datenbank $\mathcal{D}|a$ und die bedingte FP-tree-Struktur $\text{FP-tree}|a$ für ein häufiges Objekt $a \in \mathcal{I}$ konstruieren lassen. Mit anderen Worten ist die folgende Frage zu beantworten: Wie kann man die Zeilen (5) und (6) der in **Algorithm (3)** auf der Seite (22) formulierten Prozedur implementieren?

Eine Lösung dieser Frage besteht darin, dass man die bedingte FP-tree-Struktur $\text{FP-tree}|a$ direkt aus der momentan aktuellen FP-tree-Struktur ableitet, ohne die bedingte Datenbank $\mathcal{D}|a$ explizit zu berechnen.

Ist $HT = [(a_1, s_1, L_1), \dots, (a_i, s_i, L_i), \dots, (a_h, s_h, L_h)]$ die Header-Tabelle der momentan aktuellen FP-tree-Struktur und ist die bedingte FP-tree-Struktur $\text{FP-tree}|a_i$ zu konstruieren, dann sieht ihre Konstruktion, also die Konstruktion von $\text{FP-tree}|a_i$, in unserer Java-Implementierung folgendermaßen aus:

1. Eine neue Header-Tabelle $HT_i = [(a_1, s'_1, L'_1), \dots, (a_j, s'_j, L'_j), \dots, (a_{i-1}, s'_{i-1}, L'_{i-1})]$ wird erzeugt, wobei $s_j = 0$ und $L_j = \emptyset$ für alle $1 \leq j \leq i - 1$ gilt.
Betrachten wir zum Beispiel die in der Abbildung (2.2) dargestellten FP-tree-Struktur, dann gilt $HT_e = [(d, s_d, L_d), (b, s_b, L_b), (c, s_c, L_c), (a, s_a, L_a)]$ für $a_i = e$, wobei $s_d = s_b = s_c = s_a = 0$ und $L_d = L_b = L_c = L_a = \emptyset$ gilt.
2. Indem die mit a_i in der Header-Tabelle HT assoziierte Liste L_i durchlaufen wird, wird jeder Präfix-Pfad $P|a_i = (L, count_i)$ von a_i (s. Definition (2.8)) erreicht und durchlaufen. Für jeden Präfix-Pfad $P|a_i = (L, count_i)$ wird der Wert s'_j um den Wert $count_i$ erhöht ($s'_j \leftarrow s'_j + count_i$), falls $a_j \in L$ mit $1 \leq j \leq i - 1$ gilt.
Zum Beispiel haben wir nach der Ausführung dieses Schrittes für HT_e folgendes: $s_d = 2, s_b = 2, s_c = 2$ und $s_a = 1$ (siehe nochmal die Abbildung (2.2)).
Nach der Ausführung dieses Schrittes gilt $s'_j = \text{sup}(a_j, \mathcal{D}|a_i)$ mit $1 \leq j \leq i - 1$.

3. Alle Einträge (a_j, s'_j, L'_j) , für die $s'_j \leq s$ gilt, wobei s der minimale Support ist, werden aus der neuen Header-Tabelle HT_i entfernt.

Ist zum Beispiel $s = 1$, dann sieht HT_e nach der Ausführung dieses Schrittes wie folgt aus: $HT_e = [(d, 2, L_d), (b, 2, L_b), (c, 2, L_c)]$.

4. In diesem Schritt wird angenommen, dass jeder Knoten $node$ der FP-tree-Struktur einen Zeiger $node \cdot copy$ enthält, der vom Typ „Knoten“ ist und mit dem Wert $null$ initialisiert ist.

Jeder Präfix-Pfad $P|\{a_i\} = (L, count_i)$ wird in diesem Schritt durchlaufen, wobei bei jedem Durchlauf folgendes auszuführen ist:

- Enthält der Präfix-Pfad einen Knoten $node$, der mit einem Objekt $a_j \in HT_i$ markiert ist und für den der Zeiger $node \cdot copy$ gleich $null$ ist, dann wird vom diesen Knoten eine Kopie $node'$ gemacht, die dem Zeiger $node \cdot copy$ zuzuweisen ist ($node \cdot copy \leftarrow node'$). Der neue Knoten $node'$ wird in die Liste L'_j aufgenommen und seiner Komponente $node' \cdot count$ wird der Wert $count_i$ zugewiesen ($node' \cdot count \leftarrow count_i$).
- Enthält der Präfix-Pfad einen Knoten $node$, der mit einem Objekt $a_j \in HT_i$ markiert ist und für den der Zeiger $node \cdot copy$ auf eine Kopie $node'$ von $node$ zeigt, dann wird lediglich die Komponente $node' \cdot count$ um den Wert $count_i$ erhöht ($node' \cdot count \leftarrow node' \cdot count + count_i$).

Alle Kopie-Knoten und die Header-Tabelle HT_i stellen zusammen nach der Ausführung dieses Schrittes die bedingte FP-tree-Struktur $FP-tree|\{a_i\}$ dar.

Zum Beispiel besteht die bedingte FP-tree-Struktur $FP-tree|\{e\}$ (s. Abbildung (2.7)) nach der Ausführung dieses Schrittes aus der Header-Tabelle HT_e und den Pfaden: $[(d, 2), (b, 1)]$, $[(d, 2), (c, 1)]$ und $[(b, 1), (c, 1)]$.

5. In diesem Schritt sind die Knoten der bedingten FP-tree-Struktur $FP-tree|\{a_i\}$ von den Knoten der aktuellen FP-tree-Struktur zu trennen. Dazu wird für jedes Objekt a_j mit $(1 \leq j \leq i - 1)$ die mit ihm in der Header-Tabelle HT assoziierte Liste L_j durchlaufen. Dabei wird der Zeiger $copy$ jedes in L_j enthaltenen Knotens $node \in L_j$ auf den Wert $null$ gesetzt ($node \cdot copy \leftarrow null$).

Um also die bedingte FP-tree-Struktur $FP-tree|\{a_i\}$ aus der aktuellen FP-tree-Struktur zu konstruieren, werden zuerst alle in $\mathcal{D}|\{a_i\}$ häufigen einelementigen Mengen indirekt in den Schritten 1, 2 und 3 bestimmt und danach werden in den Schritten 4 und 5 alle Knoten der $FP-tree|\{a_i\}$ aus den mit den zuvor in den Schritten 1, 2 und 3 herausgefundenen Objekten markierten Knoten der aktuellen FP-tree-Struktur entsprechend generiert.

Um unsere Java-Implementierung zu testen, haben wir aus [51] eine C++-Implementierung¹ des FP-growth-Algorithmus auf den in der Einleitung dieses Kapitels beschriebe-

¹Man hat hier Herrn Bart Goethals dafür zu danken, dass er diesen Code auf seiner Internetseite zur Verfügung gestellt hat.

nen Rechner heruntergeladen und mit dem Compiler GNU-gcc kompiliert.

Datenbank	#Objekte	#Trans.	max. Länge	min. Länge	durchschnitt. Länge
Chess	75	3196	37	37	37
Connect	129	67556	43	43	43
Mushroom	119	8124	23	23	23
Pumsb	2113	49046	74	74	74
Pumsb-star	2088	49046	63	49	50
Retail	16470	88162	76	1	10
T10I4D100K	870	100000	29	1	10

Tabelle 7.1: Diese Datenbanken wurden in den Experimenten verwendet, um die Laufzeit der Java-Implementierung vom FP-growth-Algorithmus mit der Laufzeit der C++-Implementierung des selben Algorithmus zu vergleichen. Diese Datenbanken sind in [52] verfügbar.

Die in der Tabelle (7.1) mit ihren charakteristischen Merkmalen aufgelisteten Datenbanken wurden als Testdatenbanken verwendet. Sie sind im FIMI-Repository verfügbar ([52]). Die beiden Implementierungen haben für jede dieser Datenbanken und für verschiedene Werte des minimalen Supports jeweils die gleichen häufigen Mengen ausgegeben. Die Ergebnisse des experimentellen Vergleiches der Laufzeit der Java-Implementierung mit der Laufzeit der C++-Implementierung sind in den Abbildungen (7.1) und (7.2) dargestellt.

Es ist an diesen Diagrammen festzustellen, dass die beiden Implementierungen bei den allen in der Tabelle (7.1) aufgelisteten Datenbanken und für relativ große minimale Supports fast eine identische Laufzeit haben. Aber für relativ kleine minimale Supports verhalten sie sich in Bezug auf die Laufzeit unterschiedlich. In diesem Zusammenhang ist die Laufzeit der Java-Implementierung bei den Datenbanken „Chess“, „Connect“ und „Pumsb“ besser als die Laufzeit der C++-Implementierung, während bei den anderen Datenbanken der umgekehrte Fall gilt.

Dies lässt sich auf die andere Technik zurückführen, die von Bart Goethals ([51]) in seiner C++-Implementierung angewendet wurde. Der wesentliche Unterschied zwischen unserer Java-Implementierung und der C++-Implementierung liegt darin, dass in der C++-Implementierung die bedingte Datenbank $\mathcal{D}|a_i$ explizit generiert und danach die bedingte FP-tree-Struktur $\text{FP-tree}|a_i$ konstruiert wird. Dies wird folgendermaßen durchgeführt:

1. Alle Präfix-Pfade $P|a_i$ werden konstruiert, in dem die mit dem Objekt a_i in der Header-Tabelle der aktuellen FP-tree-Struktur assoziierten Liste durchlaufen wird. Betrachten wir zum Beispiel die Abbildung (2.2) auf der Seite (2.2), dann sind alle Präfix-Pfade des Objektes a : $([d, b], 2)$, $([d, c], 1)$ und $([d], 1)$
2. Eine neue FP-tree-Struktur $\text{FP-tree}|a_i$ wird initialisiert. Jeder Präfix-Pfad $P|a_i$

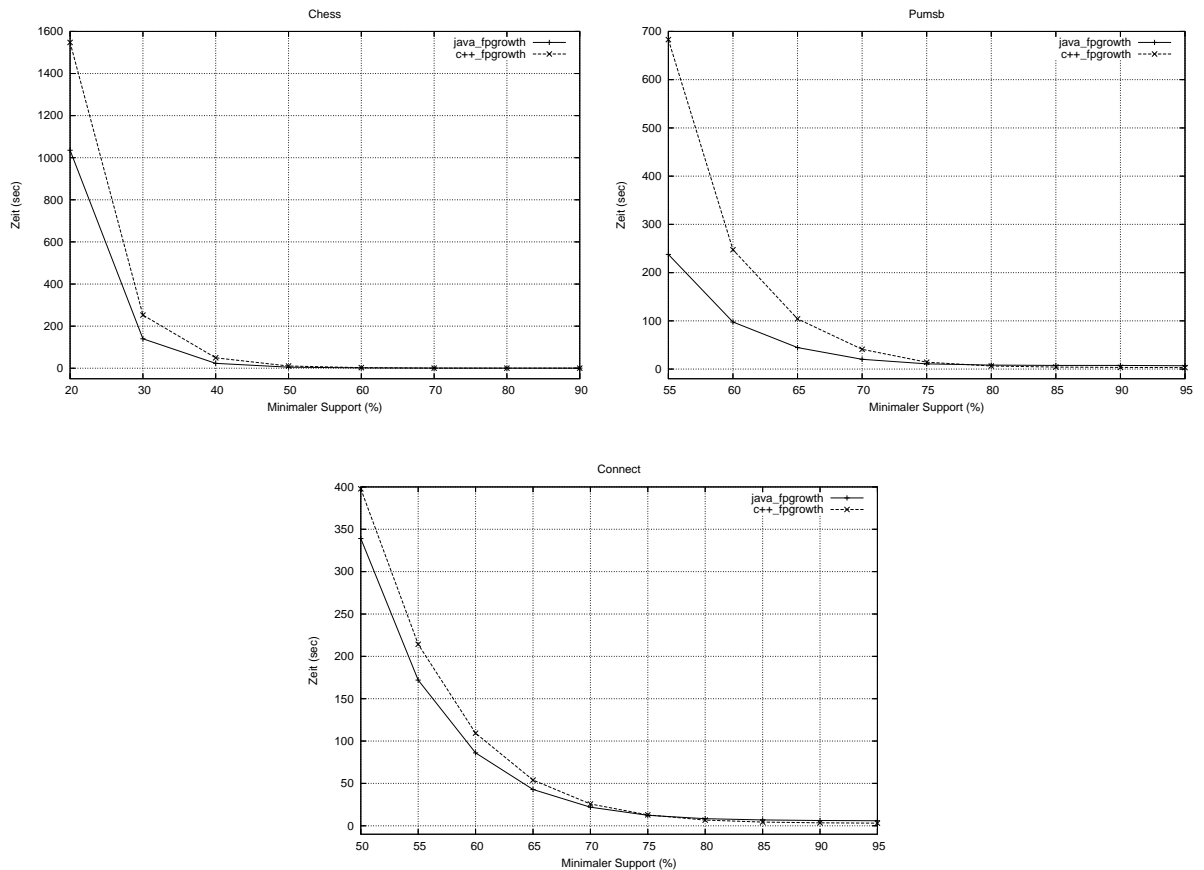


Abbildung 7.1: Die Java-Implementierung zeigt hier für relativ kleine minimale Supports eine bessere Laufzeit als C++-Implementierung, während die beiden Implementierungen für relativ große minimale Supports die selbe Laufzeit haben.

$= (L, count_i)$ wird mithilfe einer leicht modifizierten Variante der in **Algorithm (2)** auf der Seite (19) formulierten Prozedur „insert“ in die neue konstruierte FP-tree-Struktur $FP-tree|a_i$ eingefügt. Dabei entsteht die modifizierte Variante der Prozedur „insert“ dadurch, dass man in dieser Prozedur die Anweisung in der Zeile (5) durch die Anweisung: $node \cdot count \leftarrow node \cdot count + count_i$ und die Anweisung der Zeile (12) durch die Anweisung: $newNode \cdot count \leftarrow count_i$ ersetzt.

Nach der Ausführung dieses Schrittes und im Hinblick auf die Präfix-Pfade des Objektes a (siehe die Abbildung (2.2)) hat die Header-Tabelle von $FP-tree|a$ die Gestalt: $[(d, 4, [d]), (b, 2, [b]), (c, 1, [c])]$ und der Baum von $FP-tree|a$ besteht aus den Pfaden: $[root, (d/4), (b/2)]$ und $[root, (d/4), (c/1)]$.

3. In diesem Schritt werden alle nicht häufigen Objekten aus der neu konstruierten Struktur $FP-tree|a_i$ entfernt. Dazu werden für jeden Eintrag (a_j, s_j, L_j) in der

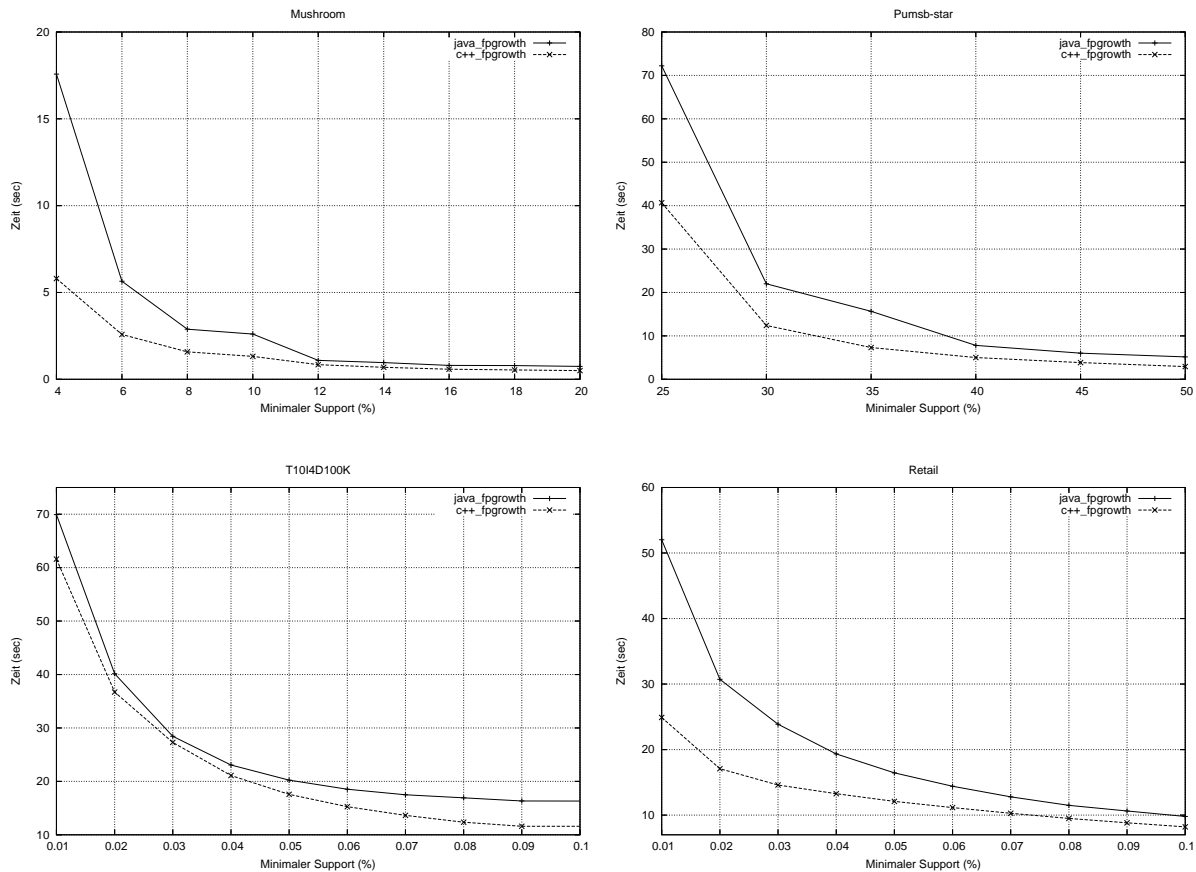


Abbildung 7.2: Die C++-Implementierung zeigt hier für relativ kleine minimale Supports eine bessere Laufzeit als Java-Implementierung, während die beiden Implementierungen für relativ große minimale Supports ungefähr die selbe Laufzeit haben.

Header-Tabelle von $\text{FP-tree}\{a_i\}$, für den $s_j \leq s$ (s ist der minimale Support) gilt, alle in der Liste L_j enthaltenen Knoten von der Struktur $\text{FP-tree}\{a_i\}$ gelöscht und danach wird der Eintrag selbst aus der Header-Tabelle entfernt.

Für den minimalen Support $s = 1$ ist also der Eintrag $(c, 1, [c])$ aus der Header-Tabelle von $\text{FP-tree}\{a\}$ zu entfernen, sobald der Knoten $(c/1)$ aus dem Baum gelöscht wurde. Also besteht der Struktur $\text{FP-tree}\{a\}$ nach der Ausführung dieses Schrittes nur aus einem einzigen Pfad, nämlich dem Pfad: $[root, (d/4), (b/2)]$.

Also unterscheiden sich die beiden Implementierung wesentlich in der Technik, nach der die Zeilen (5) und (6) der in **Algorithm (3)** auf der Seite (22) formulierten Prozedur implementiert worden sind. Der Grund dafür, dass wir in Bezug auf die Laufzeit kein eindeutiges Ergebnis bekommen haben, lässt sich auf die Eigenschaften der verwendeten

Datenbanken zurückführen. Mit anderen Worten stellen die Eigenschaften der Datenbanken „Chess“, „Pumbs“ und „Connect“ Vorteile für die Java-Implementierung dar, während sie Nachteile für die C++-Implementierung darstellen. Der umgekehrte Fall gilt für die Eigenschaften der anderen Datenbanken.

7.2 Entdeckung häufiger Episoden in Ereignis-Sequenzen

Die Implementierung des im Kapitel (4) formulierten Algorithmus zur Entdeckung häufiger Episoden in einer Ereignis-Sequenz hat folgende Bedingungen zu erfüllen:

1. Für eine Episode α , für die bekannt ist, dass sie in einer Ereignis-Sequenz $\mathcal{S} = (S, T_s, T_e)$ mit dem Support $sup(\alpha, \mathcal{S}, win)$ vorkommt, muss die Implementierung für die Eingaben \mathcal{S} und s mit $0 \leq s < sup(\alpha, \mathcal{S}, win)$ die Episode α als häufige Episode mit der Häufigkeit $sup(\alpha, \mathcal{S}, win)/(T_e - T_s + win - 1)$ liefern.
2. Für einen festgelegten minimalen Support muss die Laufzeit monoton in der Zeitfensterbreite win sein.
3. Für einen festgelegten minimalen Support und eine festgelegte Zeitfensterbreite muss die Laufzeit zur Entdeckung der (injektiven) seriellen Episoden länger als die Laufzeit zur Entdeckung der (injektiven) parallelen Episoden sein.
4. Für einen festgelegten minimalen Support und eine festgelegte Zeitfensterbreite muss die Entdeckung der seriellen (parallelen) Episoden mindestens so lange dauern, wie die Entdeckung der injektiven seriellen (parallelen) dauert.
5. Für einen festgelegten minimalen Support und eine festgelegte Zeitfensterbreite muss die Anzahl der gelieferten seriellen (parallelen) Episoden mindestens so groß sein, wie die Anzahl der gelieferten injektiven seriellen (parallelen) Episoden ist.

Um die erste Anforderung überprüfen zu können, war die künstliche Generierung einer Ereignis-Sequenz erforderlich. Zur Vorstellung der Technik, durch die eine Ereignis-Sequenz $\mathcal{S} = (S, T_s, T_e)$ künstlich generiert wurde, sind folgende Vereinbarungen zu treffen: Die Anzahl der Ereignis-Typen, die an einem Zeitpunkt t mit $(T_s \leq t < T_e)$ vorkommen, wird durch n_t gekennzeichnet. Die Anzahl der Zeitfenster, in denen eine Episode α vorkommt, wird durch w_α gekennzeichnet. Dementsprechend sieht die künstliche Generierung einer Ereignis-Sequenz, in der zum Beispiel die Episoden $\alpha_1 = [A, B, C, D]$, $\alpha_2 = [A, B, A, C, D]$ und $\alpha_3 = [X, Y, Z]$ vorkommen, folgendermaßen aus:

- Zuerst sind T_s und T_e zu initialisieren. Zum Beispiel kann T_s mit dem Wert 1 und T_e mit dem Wert 10000 initialisiert werden.
- Für jeden Wert t mit $(T_s \leq t < T_e)$ wird die Variable n_t zufällig mit einer Zahl initialisiert, die 0 sein darf und kleiner als ein bestimmter Wert (z.B. 10) ist. Danach werden n_t zufällige Zahlen z_1, z_2, \dots, z_{n_t} generiert. Die Tupeln $(z_1, t), (z_2, t), \dots, (z_{n_t}, t)$ werden als Ereignisse betrachtet und in die Liste S eingefügt.

- Die Variablen w_{α_1} , w_{α_2} und w_{α_3} werden zufällig so initialisiert, dass $1 < w_{\alpha_3} < w_{\alpha_2} < w_{\alpha_1} \leq T_e - T_s + win - 1$ gilt. Man kann dabei die Zeitfensterbreite win zufällig wählen oder einfach festlegen.
- Für jeden Wert w_{α_i} mit $i = 1, 2, 3$ werden w_{α_i} Zeitfenster aus der Menge aller Zeitfenster der bisjetzt konstruierten Ereignis-Sequenz zufällig ausgewählt. In jedes Zeitfenster \mathcal{W} der w_{α_i} zufällig ausgewählten Zeitfenster werden alle in der Episode α_i vorkommenden Symbole so eingefügt, dass die Episode α_i in diesem Zeitfenster \mathcal{W} vorkommt. Nach der Ausführung dieses Schrittes gilt also $sup(\alpha_i, \mathcal{S}, win) = w_{\alpha_i}$ für $i = 1, 2, 3$.

Für die oben künstlich generierte Ereignis-Sequenz und für den minimalen Support s mit $w_{\alpha_2} \leq s < w_{\alpha_1}$ hat die Implementierung nur die Episode α_1 mit der Häufigkeit $w_{\alpha_1}/(T_e - T_s + win - 1)$ ausgegeben, während sie für den minimalen Support s mit $w_{\alpha_1} \leq s < w_{\alpha_2}$ die Episode α_2 mit der Häufigkeit $w_{\alpha_2}/(T_e - T_s + win - 1)$ und die Episode α_1 mit der Häufigkeit $w_{\alpha_1}/(T_e - T_s + win - 1)$ geliefert hat. Im dritten Fall, in dem $0 \leq s < w_{\alpha_3}$ gilt, hat die Implementierung alle drei Episoden α_1 , α_2 und α_3 mit den entsprechenden Häufigkeiten als häufige Episoden entdeckt. Für mehrere andere Ereignis-Sequenzen, die wie oben künstlich generiert wurden, hat die Implementierung den erwünschten Sachverhalt gezeigt, was bedeutet, dass die Implementierung des Algorithmus zur Entdeckung häufiger Episoden die erste Bedingung erfüllt.

Zur Überprüfung der Bedingungen 2, 3, 4 und 5 wurde die Datenbank „msnbc“ verwendet, die in [53] zur Verfügung gestellt ist. Diese Datenbank stellt die Log-Datei des Web-Servers der Firma „msnbc.com“² dar, wobei jede Zeile einem User entspricht und die Namen aller von ihm besuchten Web-Seiten in derjenigen Reihenfolge enthält, in der der User die Web-Seiten besucht hatte.

Eine Ereignis-Sequenz namens „msnSeq1000“ wurde aus dieser Log-Datei erzeugt, indem 1000 Zeilen zufällig aus der Datei ausgewählt und aneinander angehängt wurden. Dabei wurde zwischen je zwei aneinander angehängten Zeilen ein Abstand gelassen, der gleich der durchschnittlichen Länge der 1000 ausgewählten Zeilen ist.

Für den relativen minimalen Support 0.005 und für verschiedene Zeitfensterbreiten sind die Ergebnisse der mit der Ereignis-Sequenz „msnSeq1000“ durchgeführten Experimente in den Abbildungen (7.3) und (7.4) dargestellt. Durch die Betrachtung der Diagramme der Abbildung (7.3) lässt sich feststellen, dass die Implementierung die Anforderungen 2, 3, und 4 erfüllt, während durch die Betrachtung der Diagramme der Abbildung (7.4) die Erfüllung der Anforderung 5 festgestellt werden kann.

Um die Abhängigkeit der maximalen Länge der häufigen Episoden von der Zeitfensterbreite zu demonstrieren, zeigt die Tabelle (7.2) für den relativen minimalen Support 0.005 und für jede der in dieser Tabelle angegebenen Zeitfensterbreiten die jeweilige maximale Länge der seriellen Episoden bzw. der injektiven seriellen Episoden, die in der Sequenz „msnSeq1000“ häufig sind.

²Man hat der Firma „msnbc.com“ dafür zu danken, dass sie diese Log-Server-Datei zur Verfügung gestellt hat.

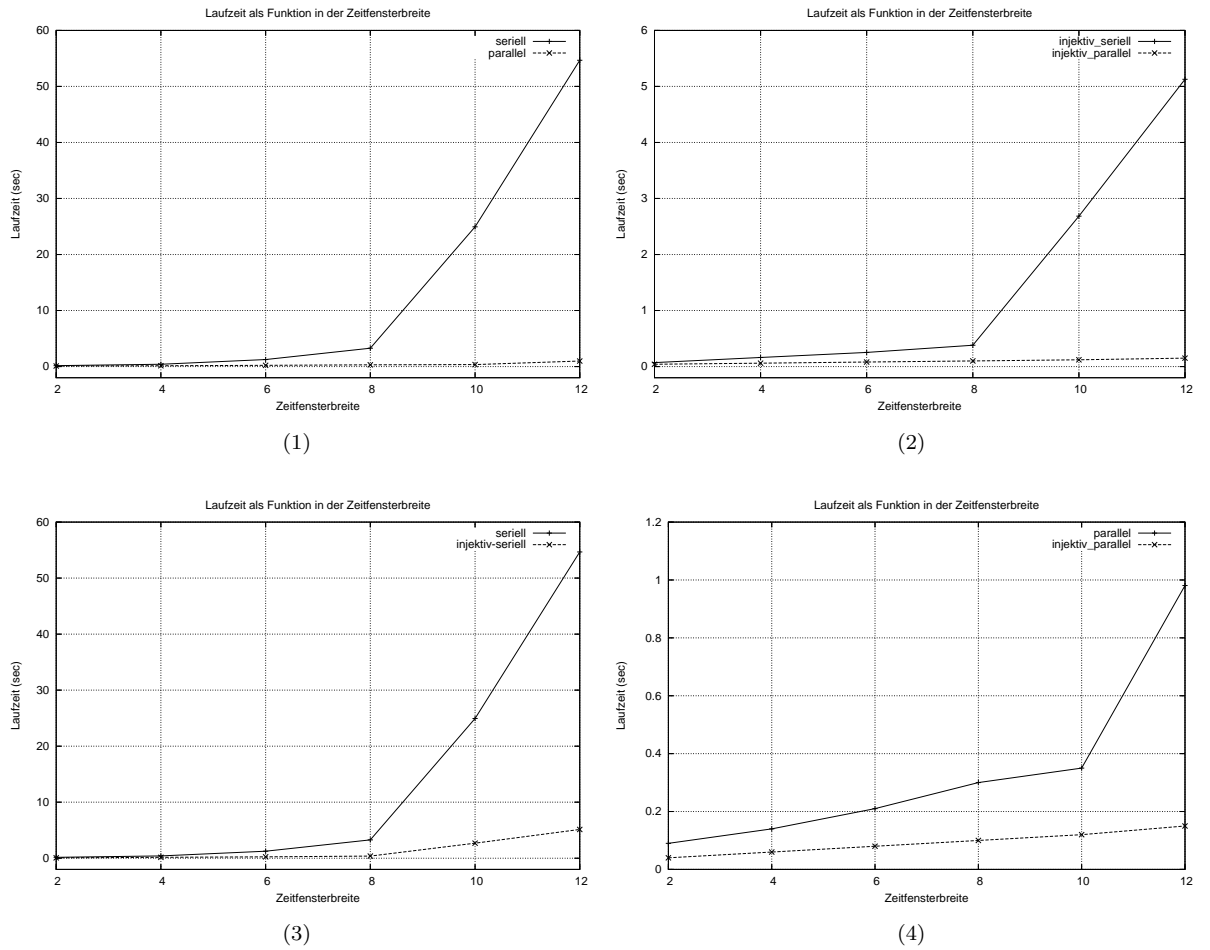


Abbildung 7.3: Diese Diagramme zeigen, dass die Laufzeit des Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen monoton in der Zeitfensterbreite ist.

Das Diagramm 1 (2) zeigt, dass die Laufzeit zur Generierung der (injektiven) seriellen Episoden länger als die Laufzeit zur Generierung der (injektiven) parallelen Episoden ist.

Das Diagramm 3 (4) zeigt, dass die Laufzeit zur Generierung der seriellen (parallelen) Episoden länger als die Laufzeit zur Generierung der injektiven seriellen (parallelen) Episoden ist.

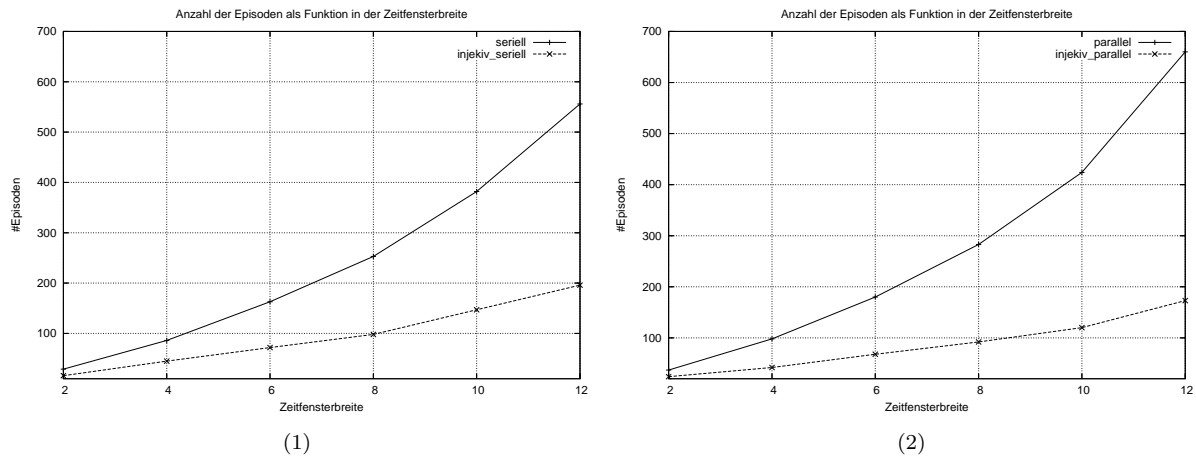


Abbildung 7.4: Das Diagramm 1 zeigt, dass die Anzahl der seriellen Episoden größer als die Anzahl der injektiven seriellen Episoden ist. Das Diagramm 2 zeigt, dass die Anzahl der parallelen Episoden größer als die Anzahl der injektiven parallelen Episoden ist.

win	max. Länge s.E.	max. Länge i.s.E.
12	10	4
10	9	4
8	8	3
6	6	3
4	4	3
2	2	2

Tabelle 7.2: Diese Tabelle zeigt für den relativen minimalen Support 0.005 und für jede der Zeitfensterbreiten die maximale Länge der seriellen Episoden (s.E.) bzw. die maximale Länge der injektiven seriellen Episoden (i.s.E.), die in der Sequenz „msnSeq1000“ häufig sind.

Als Beispiele für die gefundenen häufigen Episoden in der Sequenz „msnSeq1000“ sind in der Tabelle (7.3) alle häufigen injektiven seriellen Episoden der Länge 3 und in der Tabelle (7.4) eine Teilmenge der häufigen seriellen Episode der Länge 3 aufgelistet. Diese Episoden sind für den minimalen Support 0.001 und für die Zeitfensterbreite $win = 7$ in „msnSeq1000“ häufig.

Eine Java-Implementierung des Algorithmus ist auf der Internetseite des Lehrstuhl für künstliche Intelligenz der Universität Dortmund und unter der Kategorie Software mit der entsprechenden Dokumentation zur Verfügung gestellt.

Episode	Sup.
[frontpage, news, local]	60
[frontpage, misc, local]	48
[frontpage, news, sports]	47
[sports, frontpage, news]	46
[frontpage, news, on-air]	45
[frontpage, news, health]	39
[news, frontpage, on-air]	39
[business, frontpage, news]	37
[frontpage, local, news]	36
[frontpage, on-air, news]	36
[frontpage, living, news]	36
[frontpage, local, misc]	36
[frontpage, living, business]	36
[frontpage, health, news]	36
[frontpage, business, news]	36
[living, frontpage, news]	34
[misc, msn-sports, sports]	34
[news, misc, local]	33
[frontpage, on-air, misc]	33
[frontpage, news, tech]	31
[local, frontpage, news]	30
[frontpage, on-air, health]	29
[misc, on-air, summary]	28
[misc, frontpage, local]	28
[business, news, frontpage]	28
[news, frontpage, health]	25
[news, local, misc]	25
[local, news, frontpage]	25
[misc, local, on-air]	25
[news, on-air, frontpage]	25
[news, frontpage, sports]	24

Tabelle 7.3: Diese Tabelle zeigt beispielsweise alle injektiven seriellen Episoden der Länge 3, die für den relativen minimalen Support 0.001 und für die Zeitfensterbreite $win = 7$ häufig in der Ereignis-Sequenz „msnSeq1000“ sind.

Episode	Sup.
[frontpage, frontpage, news]	242
[misc, misc, on-air]	223
[frontpage, news, frontpage]	221
[opinion, opinion, opinion]	201
[misc, misc, frontpage]	183
[health, health, health]	172
[frontpage, frontpage, business]	170
[news, news, frontpage]	167
[on-air, misc, on-air]	161
[news, frontpage, frontpage]	156
[frontpage, living, frontpage]	153
[sports, sports, frontpage]	151
[frontpage, frontpage, sports]	148
[misc, local, local]	140
[frontpage, misc, frontpage]	139
[on-air, on-air, misc]	138
[frontpage, tech, frontpage]	134
[frontpage, business, frontpage]	134
[msn-sports, msn-sports, sports]	131
[frontpage, sports, frontpage]	123
[frontpage, business, business]	122
[frontpage, frontpage, health]	121
[tech, frontpage, frontpage]	116
[local, misc, local]	115
[frontpage, on-air, frontpage]	113
[frontpage, health, frontpage]	113
[frontpage, frontpage, on-air]	112
[frontpage, frontpage, living]	108
[misc, misc, local]	108
[living, frontpage, frontpage]	108
[misc, on-air, misc]	106
[frontpage, frontpage, local]	102
[news, frontpage, news]	102

Tabelle 7.4: Diese Tabelle zeigt beispielsweise eine Teilmenge der seriellen Episoden der Länge 3, die für den relativen minimalen Support 0.001 und für die Zeitfensterbreite $win = 7$ häufig in der Ereignis-Sequenz „msnSeq1000“ sind.

7.3 Repräsentative Episode-Regeln

Um festzustellen, um wieviel die Repräsentation durch die repräsentativen Episode-Regeln die Anzahl der Episode-Regeln reduzieren kann, wurden Experimente mit der Ereignis-Sequenzen „msnSeq1000“ und „msnSeq2000“ durchgeführt. Dabei wurde die Ereignis-Sequenz „msnSeq2000“ aus der Log-Datei „msnbc“ durch die selbe Methode erzeugt, die zur Erzeugung der Ereignis-Sequenz „msnSeq1000“ angewendet und im vorherigen Abschnitt erklärt wurde.

Die Ergebnisse dieser Experimente sind in den Abbildungen (7.5), (7.6) und (7.7) dargestellt und besagen folgendes:

- Für relativ kleine Werte der minimalen Konfidenz kann die Repräsentation durch die repräsentativen Episode-Regeln die Anzahl der Episode-Regeln um bis zu 65% reduzieren, weniger kann auch sein.
- Die Reduzierung bei den aus seriellen bzw. parallelen Episoden generierten Episode-Regeln ist größer als die Reduzierung bei den aus injektiven seriellen bzw. injektiven parallelen Episoden generierten Episode-Regeln. Dies lässt sich dadurch begründen, dass die Anzahl der seriellen bzw. parallelen Episoden mindestens so groß ist, wie die Anzahl der injektiven seriellen bzw. parallelen Episoden ist (s. die Abbildung (7.4) im vorherigen Abschnitt).

Um den Verlauf der Kurven, die jeweils für verschiedene Werte der minimalen Konfidenz die prozentuale Reduzierung der Episode-Regeln durch die repräsentativen Episode-Regeln darstellen, verstehen zu können, untersuchen wir für zwei Werte c_1 und c_2 der minimalen Konfidenz, wobei $0 < c_1 < c_2 \leq 1$ gilt, die folgenden Fälle:

Fall 1: Es gilt:

$$|\mathcal{ER}(\mathcal{S}, win, s, c_1)| - |\mathcal{RER}(\mathcal{S}, win, s, c_1)| \geq |\mathcal{ER}(\mathcal{S}, win, s, c_2)| - |\mathcal{RER}(\mathcal{S}, win, s, c_2)|.$$

Daraus schließen wir, dass die prozentuale Reduzierung für die minimale Konfidenz c_1 mindestens so groß ist, wie die prozentuale Reduzierung für die minimale Konfidenz c_2 ist. Dies bedeutet, dass die entsprechende Kurve vom Punkt c_1 zum Punkt c_2 absteigend läuft.

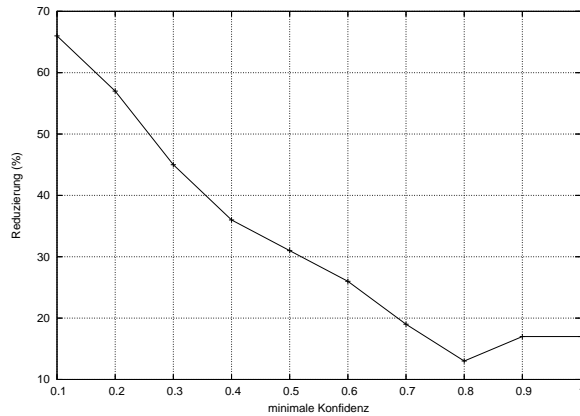
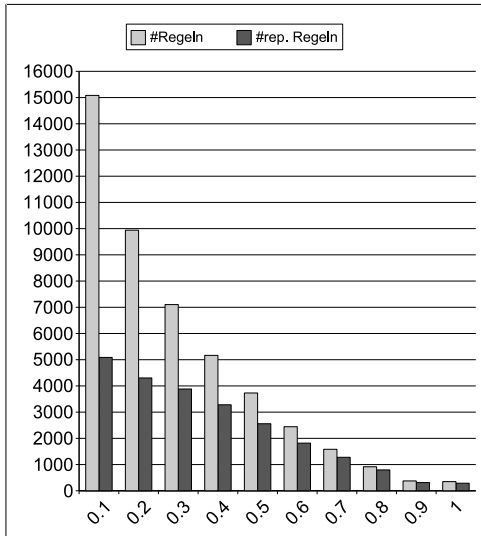
Fall 2: Es gilt:

$$|\mathcal{ER}(\mathcal{S}, win, s, c_1)| - |\mathcal{RER}(\mathcal{S}, win, s, c_1)| \leq |\mathcal{ER}(\mathcal{S}, win, s, c_2)| - |\mathcal{RER}(\mathcal{S}, win, s, c_2)|.$$

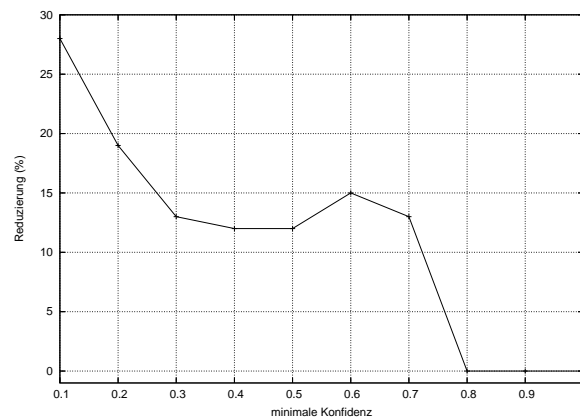
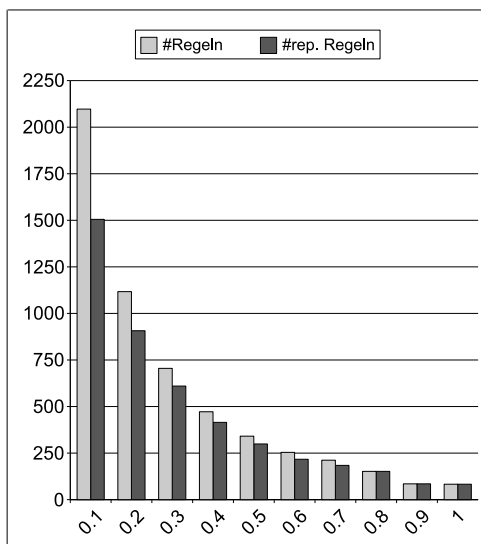
Daraus ist zu schließen, dass die prozentuale Reduzierung für die minimale Konfidenz c_1 höchstens so groß ist, wie die prozentuale Reduzierung für die minimale Konfidenz c_2 ist. Dies bedeutet, dass die entsprechende Kurve vom Punkt c_1 zum Punkt c_2 aufsteigend läuft.

Die oben untersuchten Fälle erklären also, warum die Kurven der in den Abbildungen (7.5), (7.6) und (7.7) dargestellten Diagramme in der Abhängigkeit vom Wert der minimalen Konfidenz keinen eindeutigen Verlauf (aufsteigend bzw. absteigend) haben.

Beispiele für die entdeckten repräsentativen Episode-Regeln in der Sequenz „msnSeq1000“ sind in den Abbildungen (7.8) und (7.9) dargestellt.

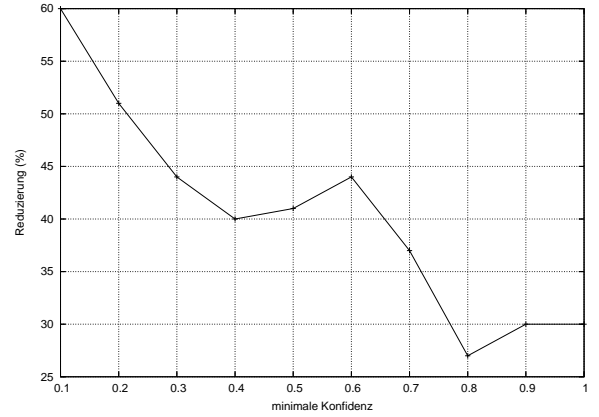
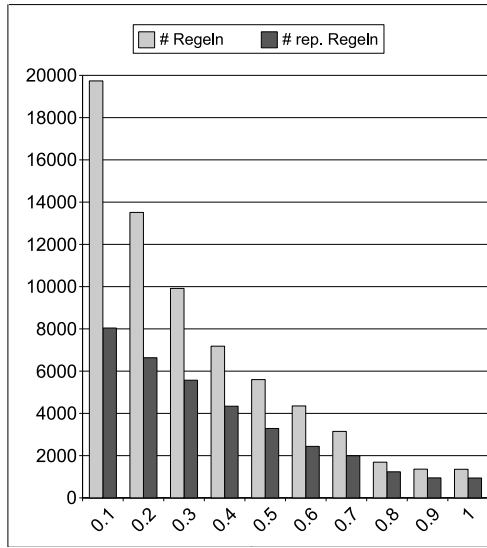


Die Regeln sind hier aus den seriellen Episoden generiert, die in der Ereignis-Sequenz „msnSeq1000“ für den relativen minimalen Support 0.0005 und für die Zeitfensterbreite 10 häufig sind.

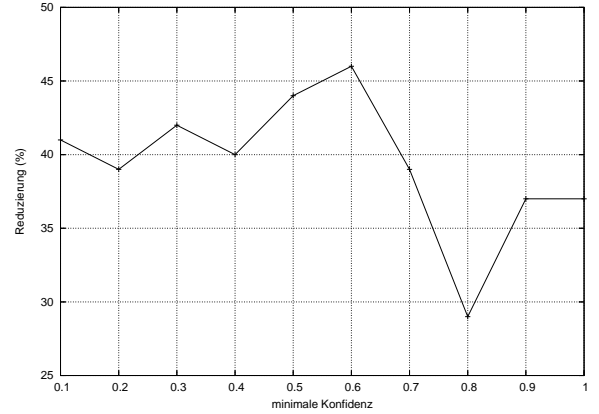
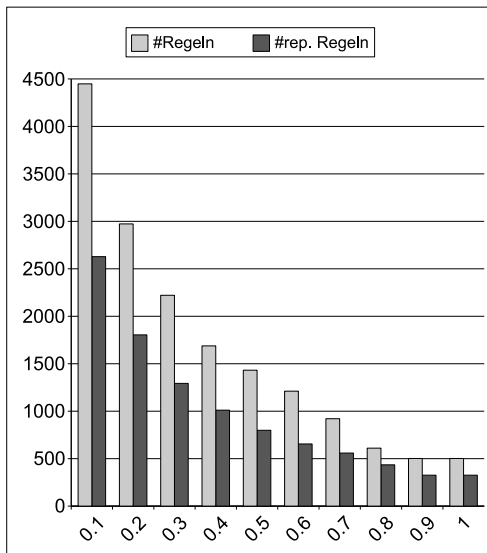


Die Regeln sind hier aus den injektiven seriellen Episoden generiert, die in der Ereignis-Sequenz „msnSeq1000“ für den relativen minimalen Support 0.0005 und für die Zeitfensterbreite 10 häufig sind.

Abbildung 7.5: In den Histogrammen sind die Anzahl der Episode-Regeln mit der Anzahl der entsprechenden repräsentativen Episode-Regeln für verschiedene Werte der minimalen Konfidenz verglichen. Die anderen Diagramme zeigen die prozentuale Reduzierung der Episode-Regeln durch die repräsentativen Episode-Regeln.

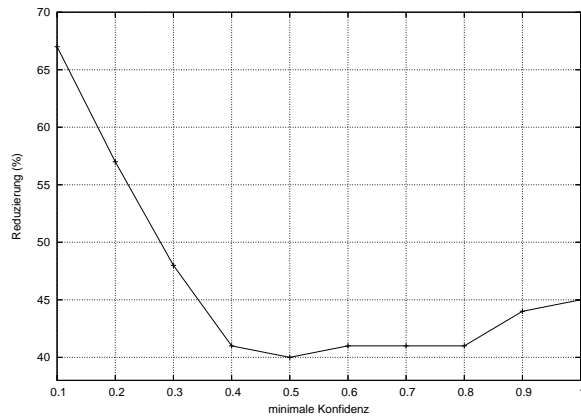
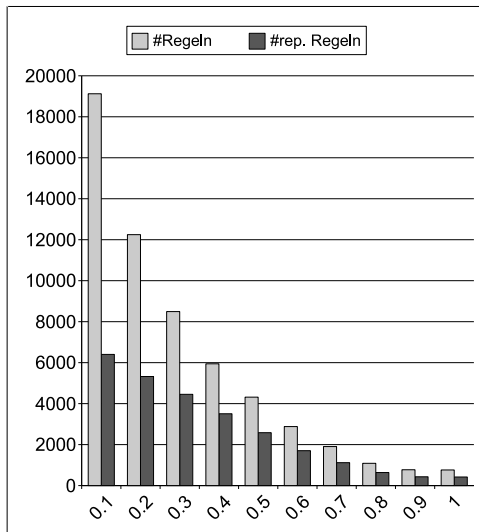


Die Regeln sind hier aus den seriellen Episoden generiert, die in der Ereignis-Sequenz „msnSeq2000“ für den relativen minimalen Support 0.0001 und für die Zeitfensterbreite 7 häufig sind.

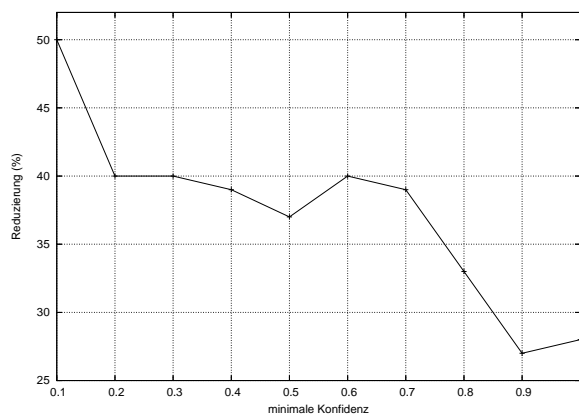
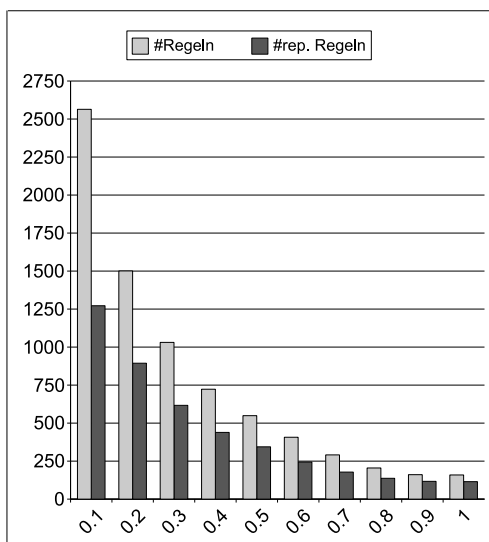


Die Regeln sind hier aus den injektiven seriellen Episoden generiert, die in der Ereignis-Sequenz „msnSeq2000“ für den relativen minimalen Support 0.0001 und für die Zeitfensterbreite 7 häufig sind.

Abbildung 7.6: Die Histogramme vergleichen jeweils für verschiedene Werte der minimalen Konfidenz die Anzahl der Episode-Regeln mit der Anzahl der entsprechenden repräsentativen Episode-Regeln, während die andere Diagramme die jeweilige prozentuale Reduzierung der Episode-Regeln durch die repräsentativen Episode-Regeln darstellen.



Die Regeln sind hier aus den parallelen Episoden generiert, die in der Ereignis-Sequenz „msnSeq2000“ für den relativen minimalen Support 0.0001 und für die Zeitfensterbreite 7 häufig sind.



Die Regeln sind hier aus den injektiven parallelen Episoden generiert, die in der Ereignis-Sequenz „msnSeq2000“ für den relativen minimalen Support 0.0001 und für die Zeitfensterbreite 7 häufig sind.

Abbildung 7.7: Die Histogramme vergleichen jeweils für verschiedene Werte der minimalen Konfidenz die Anzahl der Episode-Regeln mit der Anzahl der entsprechenden repräsentativen Episode-Regeln, während die andere Diagramme die jeweilige prozentuale Reduzierung der Episode-Regeln durch die repräsentativen Episode-Regeln darstellen.

Die repräsentative Episode-Regel:

$$r : [\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}]$$

$$\text{sup}(r) = 0.0002 \text{ und } \text{conf}(r) = 0.7$$

repräsentiert die folgenden Episode-Regeln:

$$\begin{aligned}
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{travel}] \\
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{news}, \text{travel}] \\
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{travel}] \\
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}] \\
 \\
 &[\text{msn-news}, \text{local}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{travel}] \\
 &[\text{msn-news}, \text{local}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{travel}] \\
 &[\text{msn-news}, \text{news}, \text{travel}] \rightarrow [\text{msn-news}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{living}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{living}, \text{travel}] \rightarrow [\text{msn-news}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{local}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{living}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}] \\
 \\
 &[\text{msn-news}, \text{local}, \text{news}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{local}, \text{living}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}] \\
 &[\text{msn-news}, \text{news}, \text{living}, \text{travel}] \rightarrow [\text{msn-news}, \text{local}, \text{news}, \text{living}, \text{travel}]
 \end{aligned}$$

Abbildung 7.8: Diese Abbildung stellt beispielsweise eine repräsentative Episode-Regeln mit allen von ihr repräsentierten Episode-Regeln dar. Dabei ist diese repräsentative Episode-Regel aus den injektiven seriellen Episoden generiert, die für den relativen minimalen Support 0.0001 und die Zeitfensterbreite $win = 8$ häufige in „msnSeq1000“ sind. Für jede Episode-Regel gilt nach dem Lemma (6.3), dass ihre Häufigkeit mindestens 0.0002 und ihre Konfidenz mindestens 0.7 ist.

Die repräsentative Episode-Regel:

$$r : [\text{living, msn-sports}] \rightarrow [\text{msn-news, msn-news, msn-news, living, msn-sports}]$$

$$\text{sup}(r) = 0.00025 \text{ und } \text{conf}(r) = 0.5$$

repräsentiert die folgenden Regeln:

[living, msn-sports] \rightarrow [msn-news, living, msn-sports]

[living, msn-sports] \rightarrow [msn-news, msn-news, living, msn-sports]

[living, msn-sports] \rightarrow [msn-news, msn-news, msn-news, living, msn-sports]

[msn-news, living, msn-sports] \rightarrow [msn-news, msn-news, living, msn-sports]

[msn-news, living, msn-sports] \rightarrow [msn-news, msn-news, msn-news, living, msn-sports]

[msn-news, msn-news, living, msn-sports] \rightarrow [msn-news, msn-news, msn-news, living, msn-sports]

Abbildung 7.9: Diese Abbildung stellt beispielsweise eine repräsentative Episode-Regeln mit allen von ihr repräsentierten Episode-Regeln dar. Dabei ist diese repräsentative Episode-Regel aus den seriellen Episoden generiert, die für den relativen minimalen Support 0.0002 und die Zeitfensterbreite $\text{win} = 7$ häufige in „msn-Seq1000“ sind. Für jede Episode-Regel gilt nach dem Lemma (6.3), dass ihre Häufigkeit mindestens 0.00025 und ihre Konfidenz mindestens 0.5 ist.

7.4 Das YALE-Plugin „PatternDiscovery“

Die Software YALE ([50]) ist eine Lernumgebung, die am Lehrstuhl für künstliche Intelligenz des Fachbereiches Informatik der Universität Dortmund entwickelt wurde und gepflegt bzw. angeboten wird. YALE dient einerseits zur schnellen Durchführung der typischen Anwendungen des maschinellen Lernens und der Datenanalyse, andererseits zur leichten Integration neu entwickelter bzw. implementierter Algorithmen. Das Konzept von YALE basiert auf Bäumen von Operatoren, wobei jeder Operator eine einzige Aufgabe zu erledigen hat und durch seine Eingabe, seine Ausgabe und seine Funktion definiert ist. Dieses Konzept ist eine Verallgemeinerung des Pipe-Konzeptes des Betriebssystems Unix.

Im Rahmen dieser Diplomarbeit wurde ein YALE-Plugin namens „PatternDiscovery“ geschrieben, das aus folgenden Operatoren besteht:

FPgrowth: Dieser Operator stellt die im Abschnitt (7.1) vorgestellten Java-Implementierung des FP-growth-Algorithmus dar.

AssociationRulesDiscovery: Dieser Operator ist die Implementierung des im Abschnitt (5.2.2) vorgestellten Algorithmus zur Generierung der Assoziationsregeln in einer Transaktionsdatenbank. Die zugrundeliegenden häufigen Mengen werden mithilfe des Operators „FPgrowth“ entdeckt.

FrequentEpisodesDiscovery: Dieser Operator ist die Implementierung des im Kapitel (4) entwickelten Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen.

EpisodeRulesDiscovery: Dieser Operator ist die Implementierung des im Abschnitt (6.1) vorgestellten Algorithmus zur Generierung der Episode-Regeln in einer Ereignis-Sequenz.

RepresentativeEpisodeRulesDiscovery: Dieser Operator ist die Implementierung des im Abschnitt (6.3) entwickelten Algorithmus zur Generierung aller repräsentativen Episode-Regeln aus den häufigen Episoden. Er verfügt über eine Option, deren Aktivierung dazu führt, dass er für jede repräsentative Episode-Regel alle von ihr repräsentierten Episode-Regeln generiert.

Das YALE-Plugin „PatternDiscovery“ sollte mit seiner Dokumentation auf der Internetseite des Programms YALE und unter der Kategorie Plugins zur Verfügung gestellt sein.

8 Zusammenfassung und Ausblick

Der FP-growth-Algorithmus zur Entdeckung häufiger Mengen in Datenbanken wurde im Rahmen dieser Diplomarbeit ausführlich theoretisch studiert und mit zwei anderen Algorithmen zur Lösung des selben Problems, nämlich dem Apriori-Algorithmus und dem Eclat-Algorithmus, verglichen. Die Basis des Vergleiches war die Datenstrukturen, die diese Algorithmen zur Darstellung der Datenbank bzw. zur Lösung des Problems entworfen hatten. Zur Implementierung des FP-growth-Algorithmus wurden zwei Techniken vorgestellt und miteinander verglichen. Eine dieser Techniken wurde im Rahmen der Diplomarbeit neu entwickelt und nach ihr wurde der FP-growth-Algorithmus implementiert. Die andere Technik wurde durch die Analyse einer in [51] zur Verfügung stehenden C++-Implementierung des FP-growth-Algorithmus entdeckt. Die C++-Implementierung wurde zum Testen der neuen FP-growth-Implementierung ausgenutzt.

Ein neuer Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen wurde im Rahmen der Diplomarbeit entwickelt. Er unterscheidet sich in seiner Arbeitsweise vollständig vom WINEPI-Algorithmus, der die selbe Aufgabe aber nach dem Prinzip der Kandidaten-Generierung löst und ausführlich untersucht und vorgestellt wurde.

Eine der zentralen Ideen der Entwicklung dieses neuen Algorithmus war, die symbolischen Informationen (Ereignis-Typen) von der zeitlichen Informationen (Zeitpunkte der Ereignisse) durch den Entwurf geeigneter Datenstrukturen zu trennen. Dies hat ermöglicht, eine Beziehung zwischen den häufigen Mengen und den häufigen Episoden zu entdecken bzw. zu studieren. Diese hergestellte Beziehung hat zur Erkenntnis geführt, dass die Lösung des Problems der Entdeckung häufiger Mengen zu der Lösung des Problems der Entdeckung häufiger Episoden beitragen kann. Also verwendet der neue Algorithmus zur Entdeckung häufiger Episoden in einem seiner Schritte einen Algorithmus zur Entdeckung häufiger Mengen. In seine Implementierung wurde der FP-growth-Algorithmus eingebaut, da FP-growth im Rahmen dieser Diplomarbeit implementiert wurde. Dementsprechend besitzt der neue Algorithmus eine interessante und wichtige Eigenschaft, nämlich die Eigenschaft: Wird es in der Zukunft einen besseren Algorithmus als FP-growth zur Entdeckung häufiger Mengen geben, dann führt seine Verwendung zur automatischen Verbesserung der Laufzeit der neuen Algorithmus zur Entdeckung häufiger Episoden. Die entdeckten häufigen Mengen stellen hier im Kontext der Entdeckung häufiger Episoden diejenigen Mengen dar, die häufig in der als Datenbank zu betrachtenden Menge aller Zeitfenster der Ereignis-Sequenz und deren Elemente dementsprechend Ereignis-Typen sind.

Die andere zentrale Idee bei der Entwicklung des neuen Algorithmus war die Technik, durch die alle häufigen Episoden aus den Mengen, die in der als Datenbank zu betrachtenden Menge aller Zeitfenster der Ereignis-Sequenz häufig sind, und aus den zeitlichen

Informationen über jeden Ereignis-Typ, die in geeigneten Datenstrukturen gespeichert wurden, generiert wurden. Diese Technik basiert im Wesentlichen auf der Generierung aller möglichen neuen häufigen Episoden aus jeder der bisher generierten häufigen Episoden und aus einem neu hinzugefügten Ereignis-Typ. Mit anderen Worten wurde jede der bisher generierten häufigen Episoden durch die Betrachtung aller Ereignis-Typen, die nicht in ihrer Knotenmenge enthalten sind, um alle möglichen häufigen Episoden erweitert, die Oberepisoden von ihr und länger um 1 als sie sind. Zu einer effizienten Implementierung dieser Technik wurde eine spezielle Datenstruktur namens „Episoden-Manager“ entworfen. Diese Datenstruktur dient zur gemeinsamen Verwaltung bzw. Speicherung der häufigen Mengen und der häufigen Episoden und nutzt durch ihren Entwurf die hergestellte Beziehung zwischen den häufigen Mengen und den entsprechenden häufigen Episoden aus, um diese aus den häufigen Mengen und den separat gespeicherten Zeitpunkten der in diesen häufigen Mengen vorkommenden Ereignis-Typen zu generieren.

Die dritte Aufgabe, die diese Diplomarbeit zu lösen hatte, war die Entwicklung eines neuen Konzeptes zur kondensierten Repräsentation der Episode-Regeln in Ereignis-Sequenzen. Um dies zu erreichen, hat die Diplomarbeit zuerst das Konzept der kondensierten Repräsentation der interessanten Muster in einer Datenbank formal vorgestellt und anhand der kondensierten Repräsentationen der häufigen Mengen und der Assoziationsregeln ausführlich studiert. Danach wurde das Konzept der repräsentativen Episode-Regeln entwickelt. Die Entwicklung basierte auf den folgenden Ideen:

- Die Unterepisoden-Beziehung wurde durch die Teilmengen-Beziehung formuliert. Dies war möglich, weil wir die zeitliche Relation zwischen den Elementen der Knotenmenge einer Episode mathematisch exakt als binäre Relation definiert haben (Dies ist in der Literatur und besonders in [9] nicht explizit erwähnt bzw. definiert).
- Eine Vereinigung von zwei Episoden wurde durch die Definition eines Operators, nämlich des Operators \sqcup , formuliert.
- Die Formulierung der Unterepisoden-Beziehung durch die Teilmengen-Beziehung und die Definition der Vereinigung von zwei Episoden haben die Möglichkeit geschaffen, einen Operator über der Menge der Episode-Regeln, nämlich den Operator RC , zu definieren. Die Anwendung des RC -Operators auf eine Episode-Regel $\beta \rightarrow \alpha$ liefert eine Menge von Episode-Regeln, deren Prämissen Oberepisoden von β und deren Konklusionen Unterepisoden von α sind.
- Mithilfe des RC -Operators konnte die Menge aller Episode-Regeln in zwei disjunkte Untermengen eingeteilt werden. Jede Episode-Regel der ersten Untermenge hat die Eigenschaft, dass sie durch die Anwendung des RC -Operators auf eine andere Episode-Regel nicht abgeleitet werden kann, während jede Episode-Regel der zweiten Untermenge die umgekehrte Eigenschaft hat, nämlich die Eigenschaft, dass sie durch die Anwendung des RC -Operators auf mindestens eine andere Episode-Regel generiert werden kann. Die Idee der kondensierten Repräsentation bestand darin, dass jede Episode-Regel $\gamma \rightarrow \delta$ der ersten Untermenge als Stellvertreter für diejenigen Episode-Regeln der zweiten Untermenge eingesetzt werden kann, die aus ihr

(aus $\gamma \rightarrow \delta$) durch die Anwendung des *RC*-Operators generiert werden können. In diesem Sinn wurden die Episode-Regeln der ersten Untermenge als repräsentative Episode-Regeln bezeichnet.

- Es wurde gezeigt, dass die Menge aller interessanten repräsentativen Episode-Regeln mit dem *RC*-Operator als Inferenz-Methode eine korrekte und vollständige Repräsentation der Menge aller interessanten Episode-Regeln darstellt. Die Repräsentation ist korrekt in dem Sinn, dass jede aus der Menge der interessanten repräsentativen Episode-Regeln abgeleitete Episode-Regel eine interessante Episode-Regel ist. Sie ist vollständig in dem Sinn, dass alle interessanten Episode-Regeln aus der Menge der interessanten repräsentativen Episode-Regeln durch die Anwendung des *RC*-Operators generiert werden können.

Also besteht die kondensierten Repräsentation der Episode-Regeln aus der Menge der repräsentativen Episode-Regeln und dem *RC*-Operators als Inferenz-Methode. Zur Entdeckung der interessanten repräsentativen Episode-Regeln in einer Ereignis-Sequenz wurde ein Algorithmus formuliert, der sie aus der Menge aller häufigen Episoden generiert. Die Experimente haben in Bezug auf die verwendeten Datensätze (Ereignis-Sequenzen) gezeigt, dass die Anzahl der Episode-Regeln durch die Repräsentation durch die repräsentativen Episode-Regeln um bis zu 65% reduziert werden kann. Diese prozentuale Reduzierung kann aber weniger sein und ist von dem Wert der minimalen Konfidenz und der Ereignis-Sequenz abhängig.

Durch die Generierung der interessanten repräsentativen Episode-Regeln aus der Menge der häufigen Episoden ist der Zusammenhang der drei Aufgaben, die diese Diplomarbeit zu lösen hatte, deutlicher wie folgt zu erkennen: Die Implementierung des *FP-growth*-Algorithmus wurde in die Implementierung des neuen Algorithmus zur Entdeckung häufiger Episoden eingebaut. Aus den häufigen Episoden wurde die Menge aller interessanten repräsentativen Episode-Regeln generiert.

Im Rahmen der Diplomarbeit wurde ein Yale-Plugin geschrieben, das die Implementierung der folgenden Algorithmen darstellt:

1. *FP-growth*-Algorithmus (s. die Abschnitte 2.2.3 und 7.1).
2. Des Algorithmus zur Generierung der Assoziationsregeln aus häufigen Mengen (s. den Abschnitt 5.2).
3. Des neuen Algorithmus zur Entdeckung häufiger Episoden in Ereignis-Sequenzen (s. das Kapitel 4 und den Abschnitt 7.2).
4. Des Algorithmus zur Generierung der Episode-Regeln aus häufigen Episoden (s. den Abschnitt 6.1).
5. Des Algorithmus zur Generierung der interessanten repräsentativen Episode-Regeln aus häufigen Episoden (s. die Abschnitte 6.3 und 7.3).

Es ist am Ende dieser Zusammenfassung zu erwähnen, dass das Ziel aller durchgeführten Experimente das Testen der Implementierungen auf Korrektheit und Stabilität und nicht die Analyse irgendeines Datensatzes war.

Mit Blick in die Zukunft werden im Rest dieses Kapitels weitere Probleme und Fragen gestellt, deren Untersuchung auf dem Inhalt dieser Diplomarbeit basieren kann.

Der in [14] vorgestellte Algorithmus zur Entdeckung häufiger Intervall-Muster in Intervall-Sequenzen folgt wie Apriori-Algorithmus und WINEPI-Algorithmus auch der Technik der Kandidaten-Generierung. Es stellt sich die Frage, ob man von den Ideen unseres Algorithmus zur Entdeckung häufiger Episoden profitieren kann, um einen Algorithmus zur Entdeckung häufiger Intervall-Muster zu entwickeln, der auf das Prinzip der Kandidaten-Generierung verzichtet.

Der im Rahmen der Diplomarbeit formulierte Algorithmus zur Generierung der repräsentativen Episode-Regeln in einer Ereignis-Sequenz benötigt als Eingabe die häufigen Episoden in der Ereignis-Sequenz. Aber wenn man den Beweis des Lemmas (6.6), das zeigt, dass die Menge der repräsentativen Episode-Regeln eine vollständige Repräsentation der Menge aller Episode-Regeln ist, genau analysiert, erkennt man eine Möglichkeit zur Formulierung eines anderen Algorithmus zur Generierung der repräsentativen Episode-Regeln, der als Eingabe die Menge aller Episode-Regeln hat. Mit anderen Worten man kann die Menge der repräsentativen Episode-Regeln nicht nur aus der Menge der häufigen Episoden sondern auch aus der Menge der Episode-Regeln generieren.

Analog zum Konzept der häufigen abgeschlossenen Mengen, das im Abschnitt (5.1.3) definiert wurde und zu einer kondensierten Repräsentation der häufigen Menge dient, kann man eine abgeschlossene häufige Episode wie folgt definieren:

$\alpha \in \mathcal{F}(\mathcal{S}, win, s)$ ist eine abgeschlossene häufige Episode in \mathcal{S} genau dann, wenn für jede echte Oberepisode $\delta \in \mathcal{F}(\mathcal{S}, win, s)$ von α gilt: $sup(\alpha, \mathcal{S}, win) > sup(\delta, \mathcal{S}, win)$.

Basierend auf dieser Definition stellen sich folgende Fragen:

1. Wie kann man algorithmisch alle häufigen abgeschlossenen Episoden in einer Ereignis-Sequenz entdecken?
2. Wie lassen sich die häufigen Episoden, die nicht abgeschlossen sind, aus der Menge aller häufigen abgeschlossenen Episoden ableiten?
3. Können die interessanten repräsentativen Episode-Regeln nur aus der Menge aller häufigen abgeschlossenen Episoden generiert werden?

Als Motivation zur zukünftigen Untersuchung der oben gestellten Fragen wird im **Algorithm (17)** eine Prozedur formuliert, die die Menge der abgeschlossenen häufigen Episoden $\mathcal{CF}(\mathcal{S}, win, s)$ aus der Menge der Menge der häufigen Episoden generiert und eine mögliche Antwort auf die erste Frage darstellt. Dabei bezeichnet die Menge U_δ in der Zeile (8) die Menge aller Unterepisoden von δ . Außerdem wird mit „computeMaxFreqEpisodes“ in der Zeile (5) eine Prozedur gemeint, die die maximalen häufigen Episoden

aus einer Menge von häufigen Episoden liefert.

Algorithm 17 Eine Prozedur zur Generierung der Menge der abgeschlossenen häufigen Episoden aus der Menge der häufigen Episoden

```

1: function COMPUTECLOSEDFREQEPISODES( $\mathcal{F}(\mathcal{S}, win, s)$ )
2:    $\mathcal{CF}(\mathcal{S}, win, s) \leftarrow \emptyset$ 
3:    $\mathcal{F}_1 \leftarrow \mathcal{F}(\mathcal{S}, win, s)$ 
4:   while  $\mathcal{F}_1 \neq \emptyset$  do
5:      $\mathcal{F}_2 \leftarrow \text{computeMaxFerqEpisodes}(\mathcal{F}_1)$ 
6:      $\mathcal{CF}(\mathcal{S}, win, s) \leftarrow \mathcal{CF}(\mathcal{S}, win, s) \cup \mathcal{F}_2$ 
7:      $\mathcal{F}_1 \leftarrow \mathcal{F}_1 \setminus \mathcal{F}_2$ 
8:      $\mathcal{F}_1 \leftarrow \{\alpha \in \mathcal{F}_1 \mid \neg \exists \delta \in \mathcal{F}_2 : \alpha \in U_\delta \wedge \text{sup}(\alpha, \mathcal{S}, win) = \text{sup}(\delta, \mathcal{S}, win)\}$ 
9:   end while
10:  return  $\mathcal{CF}(\mathcal{S}, win, s)$ 
11: end function

```

Analog zu 0-freien Mengen in einer Datenbank (s. den Abschnitt 5.1.4) kann man eine häufige Episode in einer Ereignis-Sequenz als freie Episode wie folgt definieren:

Eine Episode $\alpha \in \mathcal{F}(\mathcal{S}, win, s)$ ist eine freie Episode in \mathcal{S} genau dann, wenn für jede echte Unterepisode β von α gilt: $\text{sup}(\beta, \mathcal{S}, win) - \text{sup}(\alpha, \mathcal{S}, win) > 0$.

Daher sind folgende Fragen zu stellen:

1. Wie lässt sich die Menge aller häufigen freien Episoden in einer Ereignis-Sequenz generieren?
2. Wie lassen sich die häufigen Episoden, die nicht frei sind, aus der Menge der häufigen freien Episoden berechnen?
3. Kann man eine Beziehung zwischen den häufigen abgeschlossenen Episoden und den häufigen freien Episoden finden?
4. Können die interessanten repräsentativen Episode-Regeln nur aus der Menge aller häufigen freien Episoden generiert werden?

Es ist auch interessant die folgende Frage zu beantworten: Wie kann man für eine bestimmte Zeitfensterbreite die Menge aller maximalen häufigen Episoden bzw. die Menge aller minimalen nicht häufigen Episoden berechnen, ohne die Menge aller häufigen Episoden explizit zu generieren?

Literaturverzeichnis

- [1] R. Agrawal, T. Imielinski, and A. Swami (1993). Mining association rules between sets of items in large databases. In Buneman, P. and Jajodia, S., editors, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, volume 22 (2) of SIGMOD Record, pages 207-216. ACM press.
- [2] R. Agrawal. and R. Srikant. (1994). Fast algorithms for mining association rules. In Bocca, J., Jarke, M., and Zaniolo, C., editors, Proceedings 20th International Conference on Very Large Data Bases, pages 487-499. Morgan Kaufmann.
- [3] G. Görz, C.-R. Rollinger, J. Schneeberger (Hrsg.). Handbuch der Künstlichen Intelligenz, 3. Auflage, Kapitel 14. Oldenbourg Verlag.
- [4] O. Maimon, L. Rokach. The Data Mining and Knowledge Discovery Handbook, chapter 16 and chapter 17. Springer Verlag.
- [5] M. Zaki. (2000). Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 12(3):372-390.
- [6] J. Han, J. Pei and Y. Yin. Mining frequent patterns without candidate generation. In: Proc. Conf. on the Management of Data(SIGMOD'00, Dalls, TX). ACM Press, New York, NY, USA 2000.
- [7] J. Han, J. Pei, Y. Yin and R. Mao (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 8(1):53-87.
- [8] Rainer Schlittgen und Bernd H. J. Streitberg. Zeitreihenanalyse. Oldenbourg, 9. Auflage, 2001.
- [9] H. Mannila, H. Toivonen and A.I. Verkamo. Discovery of frequent episodes in event sequences, Department of Computer Science, Series of Publications C, Report C-1997-15.
- [10] R. Agrawal. and R. Srikant. Mining Sequential Patterns. In Proceedings of the 11th International Conference on Data Engineering, pages 3-14. IEEE Computer Society, 1995.
- [11] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Advances in Database Technology- EDBT'96, Proceedings

- of the 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Band 1057 der Reihe *Lecture Notes in Computer Science*. Springer Verlag, 1996.
- [12] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26 (11): 832-843, 1983.
- [13] Po-shan Kam and Ada Wai-chee Fu. Discovering Temporal Patterns for Interval-based Events. In Yahiko Kambayashi, Mukesh Mohania and A Min Tjoa (Eds.): *DaWak 2000*, Band 1874 der Reihe *Lecture Notes in Computer Science*, pages 317-326. Springer Verlag, 2000.
- [14] F. Höppner. Learning Temporal Rules from State Sequences. In *IJCA Workshop on Learning from Temporal and Spatial Data*, pages 25-31, 2001.
- [15] Jr.Bayardo, J. Roberto. Efficiently mining long patterns from databases. In Laura M. Haas and Ashutosh Tiwary, editors, In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, June 2-4, 1998, Seattle, Washington, USA, pages 85-93. ACM Press 1998.
- [16] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In Alex Buchmann and Dimitrios Georgakopoulos, editors, *Proceedings of the 17th International Conference on Data Engineering*, April 2-6, 2001, Heidelberg, Germany, pages 443-452. IEEE Computer Society, 2001.
- [17] K. Gouda, M. Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, 29 November - 2 December 2001, San Jose, California, USA. IEEE Computer Society 2001, pages 163-170.
- [18] K. Satoh and T. Uno. Enumerating maximal frequent sets using irredundant dualization. *Discovery Science*, 6th International Conference, DS 2003, Sapporo, Japan, October 17-19, 2003, *Proceedings*, volume 2843 of *Lecture Notes in Computer Science* pages 256-268. Springer, 2003.
- [19] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science*, Antibes- Juan les Pins, France, March 14-16, 2002, *Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 133-141. Springer, 2002.
- [20] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proc. 2nd Int. Conf. on Knowledge Discovery, and Data Mining KDD'96*, pages 189-194, Portland, USA, 1996. AAAI Press.
- [21] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3): 241-258, 1997.

- [22] J. F. Boulicaut. Inductive databases and multiple uses of frequent itemsets: the cInQ approach. In Database Technologies for Data Mining- Discovering Knowledge with Inductive Queries, volume 2682 of LNCS, pages 1-23. Springer-Verlag 2004.
- [23] L. De Raedt. A perspective on inductive databases. SIGKDD Explorations, 4(2):69-77, 2003.
- [24] T. Mielikäinen. Summarization Techniques for Pattern Collections in Data Mining. PhD thesis, University of Helsinki, Department of Computer Science, Ph.D. Thesis Report A-2005-1, 2005.
- [25] T. Mielikäinen. Separating structure from interestingness. In Dai et al. Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia, may 26-28, 2004, Proceedings, volume 3056 of Lecture Notes in Artificial Intelligence. Springer, 2004, pages 476-485.
- [26] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In C. Beeri and P. Buneman editors, Proceedings of the 7th International Conference on Database Theory, volume 1540 of lecture notes in Computer Science, pages 398-416. Springer.
- [27] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Pruning closed itemset lattices for association rules. In Actes Bases de Données Avancées BDA'98, Hammamet, Tunisie, Oct. 1998.
- [28] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. Information Systems, 24(1): 25-46, Jan. 1999.
- [29] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. SIGKDD Explorations, 2(2): 66-75, 2000.
- [30] M. Zaki, C. J. Hsiao (2002). CHARM: An efficient algorithm for closed itemset mining. In R. Grossman J. Han, V. Kumar, H. Mannila and R. Motwani, editors, Proceeding of the second SIAM international conference on Data Mining.
- [31] M. Zaki and M. Ogihara. Theoretical foundations of association rules. In Proc. SIGMOD Workshop on Research Issues In Data Mining Knowledge Discovery DMKD'98, pages 1-8, June 1998.
- [32] J. Pei, J. Han, and R. Mao. COLSET an efficient algorithm for mining frequent closed itemsets. In Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery DMKD'00, Dallas, USA, May 2000.
- [33] J.F. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of frequency queries by mean of free-sets. In Proc. Principles and Practice of Knowledge Discovery in Databases PKDD'00, volume 1910 of LNAI, pages 75-85, Lyon, F, Sep. 2000. Springer-Verlag.

- [34] J.F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery journal*, 7(1): 5-22, 2003.
- [35] R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445-470. Reidel, Dordrecht-Boston, 1982.
- [36] B. Ganter und R. Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer-Verlag, 1996.
- [37] A. Bykowski and C. Rigotti. A condensed representation to find frequent patterns. In *Proc. ACM Symposium on Principles of Database Systems PODS'01*, pages 267-273, Santa Barbara, CA, USA, May 2001. ACM Press.
- [38] A. Bykowski and C. Rigotti. DBC: A condensed representation of frequent patterns for efficient mining. *Information Systems*, 28(8):949-977, 2003.
- [39] M. Kryszkiewicz und M. Gajek. Concise representation of frequent patterns based on generalized disjunction-free generators. In *Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining PAKDD'02*, volume 2336 of LNCS, pages 159-171, Taipei, Taiwan, 2002. Springer-Verlag.
- [40] M. Kryszkiewicz. Concise Representations of Association Rules. D.J. Hand et al. (Eds.): *Pattern Detection and Discovery*, LNAI 2447, pp. 92-109, 2002. Springer-Verlag Berlin Heidelberg 2002.
- [41] M. Kryszkiewicz. Representative Association Rules. In: *Proc. of PAKDD'98*. Melbourne, Australia. LNAI 1394, pages 198-209. Springer-Verlag 1998.
- [42] M. Kryszkiewicz. Fast Discovery of representative association rules. In: *Proc. of RSCTC'98*. Warsaw. LNAI 1424, pages 214-221. Springer-Verlag 1998.
- [43] M. Kryszkiewicz. Closed Set Based Discovery of Representative Association Rules. In: *Proc. of IDA'2001*. September 13-15, Lisbon, Portugal. LNCS. pages 350-359 Springer-Verlag 2001.
- [44] J. Saquer and J.S. Deogun. Using Closed itemsets for discovering representative association rules. In. *Proc. of ISMIS'00*. Charlotte. LNAI 1932. pages 495-504. Springer-Verlag 2000.
- [45] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme and L. Lakhal. Mining Minimal Non-redundant Association Rules Using Frequent Closed Itemsets. J. Lloyd et al. (Eds.): *CL 2000*, LNAI 1861, pp. 972-986. Springer-Verlag 2000.
- [46] M.J. Zaki. Generating Non-redundant Association Rules. In: *6th ACM SIGKDD 2000*.

- [47] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier and L. Lakhal. Intelligent Structuring and Reducing of Association Rules with Formal Concept Anlysis. KI/ÖGAI, pages 335-350, 2001.
- [48] U. Fayyad, G. Piatetsky-Shapiro, P.Smyth and R. Uthurusamy. Advances in knowledge discovery and data mining. MIT Press, Cambridge, Mass., 1996.
- [49] T. M. Mitchell. Machine learning. McGraw-Hill, New York, 1997.
- [50] Mierswa, Ingo and Wurst, Michael and Klinkenberg, Ralf and Scholz, Martin and Euler, Timm. YALE: Rapid Prototyping for Complex Data Mining Tasks. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), ACM Press, 2006.
- [51] <http://www.adrem.ua.ac.be/~goethals> (Die Adresse der Internetseite von Bart Goethals)
- [52] <http://fimi.cs.helsinki.fi> (Frequent Itemset Mining Implementations Repository)
- [53] <http://kdd.ics.uci.edu/databases/msnbc>