Häufige Muster in zeitbezogenen Daten

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Diplominformatiker Steffen Kempe,
geboren am 29. Mai 1979 in Magdeburg

Gutachter:
Prof. Dr. Rudolf Kruse
Prof. Dr. Gholamreza Nakhaeizadeh
PD Dr. Christian Borgelt

Promotionskolloquium:
Magdeburg, den 19. Dezember 2008

Danksagung

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit als Doktorand am Forschungszentrum der Daimler AG in Ulm. Während dieser Zeit erhielt ich von vielen Personen Anleitung, Rat und Motivation.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr. Rudolf Kruse, für das stete Interesse an meiner Arbeit, die wertvollen Rücksprachen und die initiierten Kontakte zu anderen Forschern. Des Weiteren möchte ich mich bei Prof. Dr. Gholamreza Nakhaeizadeh und PD Dr. Christian Borgelt bedanken. Prof. Nakhaeizadeh gab mir als damaliger Leiter der Abteilung Data Mining Solutions die Möglichkeit zur Dissertation und stand mir mit wichtigen Ratschlägen beiseite. Dr. Borgelt bestärkte mich durch seine umsichtige und motivierende Betreuung meiner Diplomarbeit darin, eine Promotion anzustreben.

Ich bin ebenso dankbar für die anhaltende Unterstützung meiner Kollegen am Forschungszentrum Ulm. Ohne die vielen Diskussionen und die gute Zusammenarbeit, insbesondere mit Dr. Jochen Hipp, Dr. Carsten Lanquillon und den Doktoranden Axel Blumenstock und Markus Müller, wäre die vorliegende Arbeit so nicht zu Stande gekommen. Eine große Hilfe waren auch Daniel Nolte, Anne Kröske, Christian Möwes und vor allem Doreen Pittner, die frühe Versionen der Arbeit korrigiert haben.

Zuletzt möchte ich meinen Eltern und Silvana Runow für ihren Beistand und ihre Geduld danken.

Kurzfassung

Temporales Data Mining ist ein Teilgebiet der *Wissensentdeckung in Datenbanken*, das sich mit der Suche nach Mustern in zeitbezogenen Daten beschäftigt. Zeitbezogene Daten entstehen auf natürliche Weise in den verschiedensten Anwendungsgebieten. Die vorliegende Arbeit ist durch drei Anwendungen in der Automobilindustrie motiviert, in denen zeitbezogene Daten in Form von Intervallsequenzen anfallen. Jedes Intervall einer Intervallsequenz beschreibt hierbei das Auftreten eines bestimmten Ereignisses über einen definierten Zeitraum. Für eine gegebene Menge von Intervallsequenzen werden in dieser Arbeit Algorithmen vorgestellt, die es ermöglichen, alle häufigen temporalen Muster zu identifizieren. Im Gegensatz zu existierenden Verfahren zeichnen sich die neu entwickelten Algorithmen dadurch aus, dass alle Vorkommen eines Musters innerhalb einer Intervallsequenz berücksichtigt werden und die Häufigkeit eines Musters auf der Anzahl seiner Vorkommen in den Daten basiert. Diese Eigenschaften sind sowohl für die Anwendungen dieser Arbeit als auch für viele weitere Anwendungsgebiete eine unerlässliche Voraussetzung, um die zugrunde liegende Data Mining-Aufgabe sachgerecht zu adressieren. Des Weiteren werden ergänzende Verfahren zu häufigen temporalen Mustern beschrieben, die den Praxiseinsatz der neuen Algorithmen unterstützen. Zu den ergänzenden Verfahren gehören beispielsweise eine geeignete Visualisierung temporaler Muster oder die Generierung von temporalen Regeln. Zum Abschluss wird die Wirksamkeit der neuen Algorithmen und Verfahren anhand der drei Anwendungen aus der Automobilindustrie demonstriert.

Abstract

Temporal Data Mining is the study of pattern searches in time-dependent data. Time-dependent data is generated in a natural way in many different application scenarios. This thesis is motivated by three applications from the automotive industry which produce time-dependent data in the form of interval sequences. Each interval of an interval sequence describes the occurrence of a certain event over a defined period of time. In this thesis, algorithms are developed which allow identification of all frequent temporal patterns from a given set of interval sequences. In contrast to existing work, the new algorithms are able to consider all occurrences of a temporal pattern within one interval sequence and to use a definition of frequency that is based on the number of pattern occurrences in the data. These qualities are essential for the applications described in this thesis as well as many other application scenarios in order to solve the underlying data mining task. Furthermore, supplementary methods to frequent temporal patterns which support the deployment of the new algorithms are discussed. These methods include e.g. the appropriate visualization of temporal patterns and the generation of temporal rules. Finally, the effectiveness of the new algorithms and methods are demonstrated using the three applications from the automotive industry.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahrzehnten hat die Informationstechnologie immer größere Fortschritte erzielt. Jedes Jahr können Speichermedien und Prozessoren bei sinkenden Kosten mehr Daten verwalten bzw. Daten noch schneller verarbeiten. Diese Entwicklung führte dazu, dass heute nahezu beliebige Datenmengen gespeichert und in kürzester Zeit wieder abgerufen werden können. Inzwischen benutzen Unternehmen aus den verschiedensten Branchen täglich Datenbanken, um anfallende Daten zu speichern. Die Daimler AG z.B. überwacht die Qualität aller ihrer produzierten Fahrzeuge in einer speziell angelegten Datenbank, die derzeit über ein Terabyte Speicherplatz belegt. Die amerikanische Supermarktkette Wal-Mart betreibt ein 100 Terabyte Datawarehouse, in dem alle Informationen über Zulieferer und Verkäufe aus den 2 900 Filialen gesammelt werden [Hegland, 2000]. Eine der größten Ansammlungen von Daten in einer einzigen Datenbank wird es ab 2008 geben, wenn der Large Hadron Collider, ein Teilchenbeschleuniger der europäischen Organisation für nukleare Forschung (CERN), seinen Betrieb aufnimmt und jährlich 10 Petabyte Daten über Teilchenkollisionen produziert [Gillies, 2001; Cass, 2007].

Das Speichern von Daten allein ist aber nicht ausreichend. Erst eine Analyse der Datenbestände ermöglicht es, Wissen zu gewinnen, mit dem Daimler die Qualität der Fahrzeuge verbessern, Wal-Mart einen treuen Kunden identifizieren und ein Physiker die Kräfte zwischen den Elementarteilchen erklären kann. Der hohe Umfang an Daten verhindert eine Analyse allein durch den Menschen. Durch rechnergestützte Verfahren zur Wissensentdeckung in Datenbanken wird Abhilfe für dieses Problem geschaffen. Sie nutzen Berechnungen und Algorithmen aus der Statistik, dem maschinellen Lernen, dem Softcomputing und der künstlichen Intelligenz, um einen Menschen bei der Analyse zu unterstützen. Das Forschungsgebiet der Wissensentdeckung in Datenbanken, im Englischen als „Knowledge Discovery in Databases“ (KDD) bezeichnet, wird in [Fayyad u. a., 1996a] wie folgt definiert:

> *„Knowledge Discovery in Databases is the non-trivial process of identifying valid, novel, useful and ultimately understandable patterns in data.“*

Ein relativ junges Teilgebiet innerhalb der Wissensentdeckung in Datenbanken ist das temporale Data Mining. Im Mittelpunkt des temporalen Data Minings stehen Daten, die zeitbezogene Ereignisse und Zusammenhänge beschreiben. Zeitbezogene Daten entstehen auf natürliche Weise in vielen Anwendungsfeldern, wie z.B. Aktienkursen, Datenverkehr in Netzwerken, Wetterdaten, medizinischen Daten, Nutzungsprotokollen von Webservern, Sensordaten oder Kundendaten. Die Analyseaufgaben für zeitbezogene Daten sind vergleichbar mit denen für statische (nicht-zeitbezogene) Daten. Dennoch können Analyseverfahren für statische Daten oft nicht direkt auf zeitbezogene Daten übertragen werden, da sie den Anforderungen und der

Komplexität zeitbezogener Daten nicht gerecht werden. Aus diesem Spannungsfeld leiten sich sowohl die Herausforderungen der vorliegenden Arbeit als auch viele gegenwärtige Problemstellungen des temporalen Data Minings ab. Wie eine Umfrage in [Yang u. Wu, 2006] belegt, zählen führende Wissenschaftler die Analyse zeitbezogener Daten zu den zehn herausforderndsten Problemen der KDD-Forschung.

1.1 Gegenstand dieser Arbeit

Diese Arbeit ist durch drei Anwendungen zur Analyse zeitbezogener Daten in der Automobilindustrie motiviert. In allen drei Anwendungen ist der Zeitbezug durch eine intervallbasierte Datengrundlage gegeben. Ein Intervall beschreibt hierbei das Eintreten eines Ereignisses über einen definierten Zeitraum. Die Anwendungen sind im Einzelnen:

1. *Qualitätsüberwachung von Fahrzeugflotten*: Eine Hauptaufgabe für jeden Automobilhersteller ist die Überwachung der Produktqualität beim Kunden. Temporales Data Mining kann diese Aufgabe unterstützen, indem Abfolgen von Schäden oder Kombinationen von Schäden und Fahrzeugkonfigurationen identifiziert werden, die besonders häufig auftreten und somit einen Qualitätsmangel kennzeichnen. Sowohl die Konfiguration als auch die aufgetretenen Schäden jedes Fahrzeugs lassen sich durch eine Menge von Intervallen darstellen. Die häufigen Muster der intervallbasierten Datengrundlage können von einem Ingenieur z.B. dazu verwendet werden, notwendige Produktänderungen einzuleiten.

2. *Diagnoseunterstützung*: Die Popularität elektronischer Systeme (Navigation, Mobilfunk, etc.) führte zu einer stetigen Zunahme der Anzahl elektronischer Steuergeräte im Fahrzeug. Die meisten Steuergeräte kommunizieren miteinander über ein Bussystem. Während der Entwicklung neuer Steuergeräte wird der Datenverkehr über die Bussysteme gezielt analysiert, da er den Status und alle Statuswechsel der verbundenen Steuergeräte enthält. Darüber hinaus liefern an die Bussysteme angeschlossene Sensoren Informationen über die einzelnen Fahrsituationen (Geschwindigkeit, Gang, Temperaturen, etc.). Zu jeder Fahrt lassen sich die vorliegenden Informationen zu einer Menge von Intervallen verdichten. Zum einen können die häufigen Muster aus dieser Datengrundlage dazu verwendet werden, um typische Fahrsituationen aufzuzeigen. Zum anderen unterstützen sie die Diagnose im Falle eines Steuergerätfehlers, indem sie die zeitlichen Zusammenhänge zwischen Fahrsituationen, Statuswechsel und Steuergerätfehler erklären.

3. *Customer Relationship Management*: Viele Kunden sind Wiederkäufer, das heißt sie kaufen ein Fahrzeug, verwenden es für eine bestimmte Zeit, verkaufen es und kaufen schließlich ein neues Fahrzeug derselben Marke. Diese Kunden sind für ein Unternehmen besonders wertvoll, da sie eine hohe Markenloyalität aufweisen. Die Analyse des Wiederkaufverhaltens loyaler Kunden kann Potentiale zum Cross- und Upselling bei anderen Kunden aufdecken. Vorhandene Informationen über Wiederkäufer stammen sowohl aus dem Verkauf, mit Angaben über den Fahrzeugtyp und die Verkaufsart (direkter Kauf oder Leasing), sowie aus unterschiedlichen Kundenzufriedenheitsumfragen. Wiederum lassen sich die Informationen zu jedem Kunden mit Hilfe intervallbezogener Daten repräsentieren. Muster die in dieser Datenbasis häufig auftreten, können lukrative Kundensegmente kennzeichnen und eigenen sich daher zur Steuerung von Werbeaktionen.

Obwohl die Anwendungen aus unterschiedlichen Geschäftsbereichen der Automobilindustrie stammen, teilen sie einen gemeinsamen Problemkern. Es gibt eine Menge von Objekten (Fahrzeuge, Fahrten, Kunden), zu denen zeitbezogene Informationen in Form von Intervallen vorliegen. Die Abfolge von Intervallen zu einem Objekt wird in dieser Arbeit als *Intervallsequenz* bezeichnet. Die Analyseaufgabe besteht daher darin, alle häufigen zeitbezogenen Muster in der gegebenen Menge von Intervallsequenzen zu finden.

Aus der Integration der Analyse in die Anwendungsprozesse ergeben sich zusätzliche Randbedingungen. Erst wenn ein Experte die identifizierten Muster im Kontext der Anwendung bewertet hat, können nachfolgende Maßnahmen aus ihnen abgeleitet werden. Dazu ist es notwendig, dass sowohl die Muster selbst als auch die Häufigkeit der Muster für den Experten *verständlich* sind. Insbesondere die Häufigkeit eines Musters muss die intuitive Frage beantworten können, wie oft ein Muster in den Daten vorkommt. Des Weiteren muss die Häufigkeit eines Musters *vollständig* bzgl. seiner Vorkommen in den Daten sein. Das heißt, alle Vorkommen eines zeitbezogenen Musters aus jeder gegebenen Intervallsequenz werden bei der Ermittlung der Häufigkeit berücksichtigt. Existierende Verfahren zur Suche nach häufigen Mustern in Intervallsequenzen können die Anforderungen an Vollständigkeit und Verständlichkeit der Häufigkeit zeitbezogener Muster nicht erfüllen. Sie vernachlässigen entweder multiple Vorkommen eines Musters innerhalb derselben Intervallsequenz oder sie verwenden eine Häufigkeitsdefinition, die auf der zeitlichen Ausdehnung eines Musters beruht und im Kontext der Anwendungen für Experten schwer zu interpretieren ist.

Die Entdeckung häufiger zeitbezogener Muster ist nur der erste Schritt, um die gegebenen Anwendungsprobleme zu lösen. Im zweiten Schritt müssen aus der Menge der häufigen Muster die Muster herausgefiltert werden, die neuartiges und verwendbares Wissen für die Anwendung darstellen. Dieser zweite Schritt wird von einem Anwendungsexperten durchgeführt und muss mit Hilfe ergänzender Verfahren geeignet unterstützt werden. Zu den ergänzenden Verfahren zählen bspw. die Bewertung und die Visualisierung zeitbezogener Muster.

Aus den beschriebenen industriellen Anwendungen ergeben sich daher drei Herausforderungen für diese Arbeit:

1. Es müssen Algorithmen entwickelt werden, die alle häufigen zeitbezogenen Muster in einer gegebenen intervallbasierten Datengrundlage identifizieren können. Bei der Entwicklung der Algorithmen ist zu beachten, dass
   - die Datengrundlage aus einer Vielzahl von Intervallsequenzen besteht,
   - die Häufigkeit eines Musters auf der Anzahl seiner Vorkommen in den Daten beruht
   - und alle Vorkommen eines Musters innerhalb einer Intervallsequenz berücksichtigt werden.
2. Der Praxiseinsatz der neuen Algorithmen muss durch die Entwicklung ergänzender Verfahren geeignet unterstützt werden.
3. Zuletzt sollen die neu entwickelten Algorithmen und Verfahren zur Lösung der drei Anwendungsprobleme eingesetzt werden.

Die nachfolgende Gliederung beschreibt, wie die Herausforderungen durch die einzelnen Kapitel dieser Arbeit adressiert werden.

1.2 Gliederung

Die vorliegende Arbeit setzt sich aus drei Teilen zusammen, die insgesamt acht Kapitel umfassen. Der erste Teil besteht aus den Kapiteln 1 bis 3 und vermittelt die notwendigen Grundlagen dieser Arbeit. Der zweite Teil beschäftigt sich in den Kapiteln 4 und 5 mit der Entwicklung und Evaluation neuer Algorithmen zur Entdeckung häufiger Muster in zeitbezogenen Daten. Zuletzt deckt der dritte Teil praxisbezogene Aspekte der neuen Algorithmen in den Kapiteln 6 und 7 ab. Kapitel 8 fasst die Ergebnisse der Arbeit zusammen.

In Kapitel 2 wird zunächst eine Einführung in das Gebiet der Wissensentdeckung in Datenbanken gegeben. Im Mittelpunkt der Einführung stehen dabei die Darstellung des Wissensentdeckungsprozesses anhand eines Prozessmodells sowie die Erläuterung typischer Data Mining-Aufgaben und -Methoden. Als grundlegende Verfahren zur Entdeckung häufiger Muster aus statischen Daten werden die Warenkorbanalyse und die mit ihr verbundenen Assoziationsregeln ausführlich beschrieben.

Kapitel 3 untersucht und systematisiert existierende Verfahren zur Suche nach häufigen zeitbezogenen Mustern. Insbesondere werden Unterschiede zwischen den verwendeten Mustersprachen und Häufigkeitsdefinitionen hervorgehoben.

Nachdem existierende Verfahren den Anforderungen der beschriebenen Anwendungen nicht gerecht werden, folgt in Kapitel 4 die Entwicklung neuer Algorithmen. Dazu wird zunächst die Problemstellung formal definiert. Die Vor- und Nachteile verschiedener Mustersprachen und Häufigkeitsdefinitionen werden dabei eingehend beleuchtet. Anschließend werden die Eigenschaften der Problemstellung ausführlich untersucht und Beziehungen zu verwandten Problemstellungen offengelegt. Zuletzt erfolgt die Entwicklung von drei Algorithmen zur Suche nach allen häufigen Mustern in einer gegebenen intervallbasierten Datengrundlage.[1]

In Kapitel 5 erfolgt die Evaluation der neuen Algorithmen. Es werden verschiedene Experimente durchgeführt, um die Effizienz der Verfahren bzgl. Laufzeit und Speicherbedarf zu untersuchen. Weitere Experimente analysieren das Skalierungsverhalten der Algorithmen und die Effizienz bei Einführung zeitlicher Randbedingungen. Für die Experimente werden sowohl künstliche Daten als auch reale Anwendungsdaten herangezogen.

Kapitel 6 beschreibt drei ergänzende Verfahren zu den neuen Algorithmen, die einen Benutzer bei der Bearbeitung eines Anwendungsproblems unterstützen sollen. Das erste Verfahren betrachtet die Ableitung und die Bewertung zeitbezogener Regeln aus den identifizierten häufigen Mustern. Das zweite Verfahren erlaubt eine geeignete Visualisierung von zeitbezogenen Mustern und Regeln gegenüber einem Anwender. Das dritte Verfahren ermöglicht die Integration von zeitbezogenen und nicht-zeitbezogenen Daten in einer gemeinsamen Analyse.[2]

In Kapitel 7 werden die drei Anwendungen mit Hilfe der neu entwickelten Algorithmen und Verfahren bearbeitet. Die Darstellung der Anwendungen erfolgt anhand des im Kapitel 2 erläuterten Prozessmodells zur Wissensentdeckung in Datenbanken.[3]

Zum Abschluss werden in Kapitel 8 die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf weiterführende Fragestellungen gegeben.

[1] Auszüge aus Kapitel 4 sind in [Kempe u. Hipp, 2006] und [Kempe u. a., 2008a] veröffentlicht worden.
[2] Auszüge aus Kapitel 6 sind in [Kempe u. Kruse, 2008] veröffentlicht worden.
[3] Auszüge aus Kapitel 4 und Kapitel 7 sind in [Kempe u. a., 2008b] veröffentlicht worden.

Kapitel 2

Wissensentdeckung mit Mustern und Regeln

In diesem Kapitel werden zunächst die Grundlagen der Wissensentdeckung in Datenbanken eingeführt. Dazu folgt in Abschnitt 2.1 eine Darstellung des Wissensentdeckungsprozesses mit Hilfe des etablierten CRISP-DM Prozessmodells. Darüber hinaus werden typische Data Mining-Aufgaben und die dazu benötigten Data Mining-Methoden besprochen. Anschließend werden in Abschnitt 2.2 Verfahren beschrieben, die Muster in Form von häufigen Warenkörben, Assoziationsregeln und Klassifikationsregeln aus einer gegeben Datenmenge extrahieren. Insbesondere die erörterten Methoden zur Warenkorbanalyse und zu Assoziationsregeln sind von zentraler Bedeutung für die vorliegende Arbeit. Sowohl existierende Verfahren zur Entdeckung häufiger zeitbezogener Muster (Kapitel 3) als auch die in Kapitel 4 beschriebenen Verfahren basieren in ihren zugrunde liegenden Ideen auf der Warenkorbanalyse und den damit verbundenen Assoziationsregeln.

2.1 Wissensentdeckung in Datenbanken

Die Aufgabe, ein Anwendungsproblem durch eine geeignete Analyse der vorhandenen Daten zu lösen, ist durch die sich stetig verbessernden technischen Möglichkeiten zur Datenspeicherung und zum Datenmanagement von wachsender Bedeutung. In der Literatur sind die Methoden der Datenanalyse oft mit den Schlagworten *Wissensentdeckung* (z.B. [Fayyad u. a., 1996d]), *Data Mining* (z.B. [Nakhaeizadeh, 1998]) oder *intelligente Datenanalyse* (z.B. [Berthold u. Hand, 2003]) verknüpft. Die Anwendung der Methoden ist aber nur einer von mehreren Schritten, um aus den Daten einer Anwendung einen Nutzen ziehen zu können. Nachfolgend wird der gesamte Prozess zur Wissensentdeckung in Datenbanken (kurz: KDD-Prozess) beschrieben.

2.1.1 KDD-Prozess

Der KDD-Prozess ist ein iterativer Prozess, der viele Entscheidungen von Seiten des Benutzers erfordert. Häufig muss der Analyst zwischen den einzelnen Prozessschritten wechseln, um ein gewünschtes Resultat zu erzielen. Auf Grund dieser Komplexität ist der KDD-Prozess auch in der Literatur nicht einheitlich beschrieben (vgl. [Fayyad u. a., 1996a,c; Brachman u. Anand, 1996; Williams u. Huang, 1996; Krahl u. a., 1998; Nakhaeizadeh u. a., 1998; Chapman u. a., 1999; Klösgen u. Zytkow, 2002b]). Es werden unterschiedliche Einteilungen des KDD-Prozesses in einzelne Prozessschritte vorgenommen und unterschiedliche Bezeichnungen verwendet.

Im Folgenden wird der KDD-Prozess, stellvertretend für die verschiedenen Prozessmodelle, anhand des CRSIP-DM (*CR*oss *I*ndustry *S*tandard *P*rocess for *D*ata *M*ining, [Chapman u. a., 1999]) Referenzmodells vorgestellt. Der CRISP-DM Prozess entstand aus einem Projekt, in

dem sich vier Firmen zu einem Konsortium (DaimlerChrysler, SPSS, NCR und OHRA) zusammengeschlossen hatten. Im Unterschied zu anderen Prozessmodellen weist CRISP-DM die größte Detaillierung auf. Nach Umfragen in [Piatetsky-Shapiro, 2002, 2004, 2007] ist CRISP-DM das in der Praxis am häufigsten verwendete Prozessmodell für Data Mining Projekte.

Das CRISP-DM Modell zergliedert den KDD-Prozess in die sechs Prozessschritte Anwendungsanalyse, Datenbeschaffung und -verständnis, Datenaufbereitung, Data Mining, Bewertung und Umsetzung.

Die folgende Auflistung gibt die Hauptaufgaben für jeden Prozessschritt wieder.

1. Anwendungsanalyse (Business understanding)

   Der erste Schritt beginnt mit der Bildung eines allgemeinen Verständnisses der Problemstellung aus Sicht der Anwendung. Hierzu müssen die Ziele der Anwendung formuliert und ggf. darin auftretende Widersprüche aufgelöst werden. Nachdem die Erfolgskriterien der Anwendung erkannt sind, muss das Anwendungsproblem adäquat auf ein Datenanalyseproblem abgebildet werden. Weitere Schritte betreffen die Erstellung einer Kosten-Nutzen-Analyse, die Einplanung notwendiger Ressourcen (Personal und Technik) sowie die Aufstellung eines vorläufigen Projektplans zur Durchführung des KDD-Prozesses.

2. Datenbeschaffung und -verständnis (Data understanding)

   Der zweite Schritt beginnt mit der Sammlung der notwendigen Rohdaten. Alle Daten die zur Erfüllung der identifizierten Anwendungskriterien erforderlich sind, müssen für spätere Analysen zugänglich gemacht werden. Neben der Datensammlung ist die Bildung eines Datenverständnisses das zweite wichtige Ziel in dieser Phase. Zur Bildung des Datenverständnisses kommen häufig direkte Datenbankanfragen, deskriptive Statistiken oder einfache Datenvisualisierungen zum Einsatz. Des Weiteren wird die Qualität der Daten im Hinblick auf die Anwendungsziele untersucht.

3. Datenaufbereitung (Data preparation)

   Die Datenaufbereitung umfasst alle notwendigen Transformationen, um aus den Rohdaten einen endgültigen Datensatz zu erstellen. Hierzu gehören die Auswahl von Tabellen, Attributen und Einträgen aus den Rohdaten sowie deren Datenbereinigung (z.B. Auffüllen fehlender Werte). Weitere Schritte können das Ableiten neuer Attribute oder die Diskretisierung numerischer Attribute betreffen. Häufig müssen darüber hinaus syntaktische Änderungen am Datensatz vorgenommen werden, um den Einsatz spezieller Data Mining-Algorithmen zu ermöglichen.

4. Data Mining (Modeling)

   In diesem Schritt findet die Datenanalyse statt. Dazu müssen zunächst ein oder mehrere geeignete Data Mining-Algorithmen ausgewählt und auf den erstellten Datensatz angewendet werden. Anschließend erfolgt eine erste Bewertung der berechneten Modelle unter Berücksichtigung der Anwendungskriterien (z.B. ausreichende Genauigkeit).

5. Bewertung (Evaluation)

   Während im vorherigen Schritt Modelleigenschaften, wie Genauigkeit oder Generalität, bewertet wurden, folgt in diesem Schritt eine Bewertung der Modelle auf Anwendungsebene. Es gilt die Plausibilität, Verlässlichkeit und die Anwendbarkeit der berechneten

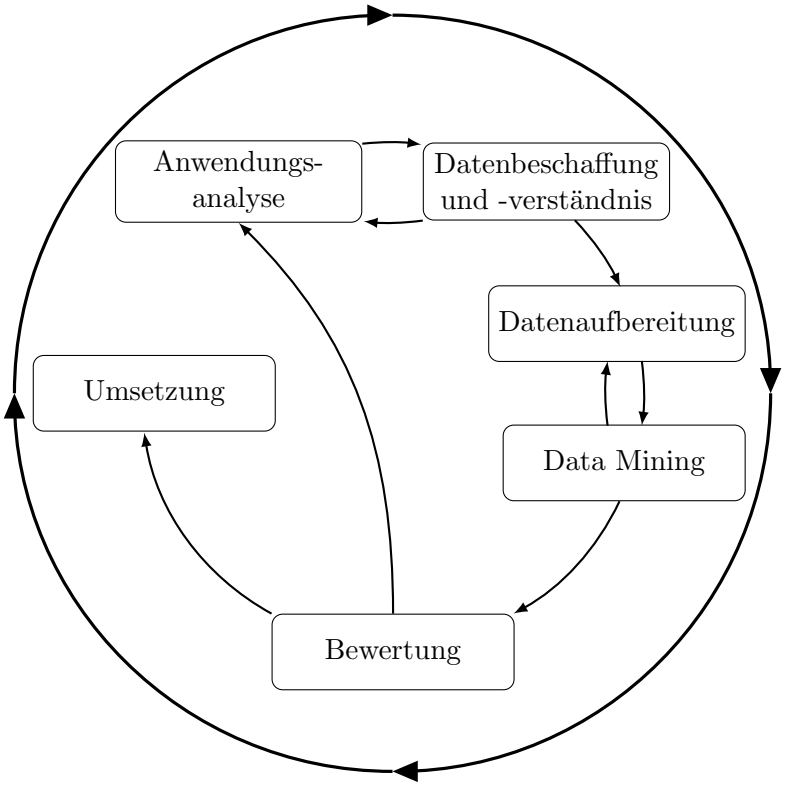


Abbildung 2.1: KDD-Prozess nach CRISP-DM

Modelle kritisch zu hinterfragen. Insbesondere die Erfüllung der Anwendungsziele aus dem ersten Schritt muss überprüft werden. Eine Revision der bisherigen Prozessschritte kann helfen, weiteres Verbesserungspotential zu identifizieren.

6. Umsetzung (Deployment)

   Der letzte Schritt beinhaltet den Transfer der erzielten Ergebnisse in die Anwendungsdomäne. Im einfachsten Fall kann die Umsetzung in Form eines Berichts oder einer Präsentation, welche die Ergebnisse für Anwendungsexperten verständlich beschreibt, erfolgen. Komplexere Fälle können jedoch die Integration einer wiederholbaren Analyse in bestehende Geschäftsprozesse des Unternehmens erfordern. In diesem Fall müssen auch Überwachungs- und Wartungsstrategien für die Analyseprozesse abgeleitet werden.

Abbildung 2.1 verdeutlicht neben der Abfolge der einzelnen Prozessschritte noch einmal den iterativen Charakter des KDD-Prozesses. Oftmals muss ein Analyst zu bereits durchlaufenen Prozessschritten zurückkehren, wenn die erzielten Resultate nicht mit den Erwartungen übereinstimmen. So kann ein unzureichendes Modell als Ergebnis des vierten Schrittes eine Rückkehr zu allen vorherigen Schritten erfordern. Es ist möglich, dass bereits eine erneute Anwendung der Data Mining-Algorithmen (Schritt 4, z.B. mit neuen Parametern) oder eine andere Datenaufbereitung (Schritt 3) zur Modellverbesserung beiträgt. Erweisen sich die Unterschiede zwischen den Ansprüchen der Anwendung und dem Potential der vorhandenen

Daten als unüberwindbar, müssen entweder weitere Datenquellen in Schritt 2 erschlossen oder sogar die Anwendungsziele in Schritt 1 selbst neu überdacht werden.

Auf Grund der Komplexität des KDD-Prozesses sind die Fähigkeiten des Analysten entscheidend für den Erfolg oder Misserfolg eines Datenanalyseprojektes. Die bestmögliche Unterstützung des Analysten muss daher ein ständiges Ziel bei der Durchführung eines KDD-Prozesses sein (vgl. [Brachman u. Anand, 1996; Hipp u. a., 2002; Blumenstock u. a., 2006]).

2.1.2 Data Mining-Aufgaben und Methoden

Es gibt eine Vielzahl von Data Mining-Methoden mit denen die Analyseziele einer Anwendung verfolgt werden können. Häufig stehen sogar mehrere Methoden zur Verfügung, so dass die Vorzüge der verschiedenen methodischen Ansätze im Sinne der Anwendung gegeneinander abzuwiegen sind. Unabhängig von den konkreten Methoden haben sich in der Vergangenheit mehrere charakteristische Data Mining-Aufgaben etabliert (vgl. z.B. [Fayyad u. a., 1996b], [Nakhaeizadeh u. a., 1998], [Weiss u. Indurkhya, 1998, Seite 7ff.] und [Reinartz, 1999, Seite 21ff.]). Im Folgenden werden die wichtigsten Data Mining-Aufgaben mit einem beschreibenden Beispiel näher erläutert.

Data Mining-Aufgaben

- Klassifikation

  Die Klassifikation ist eine der wichtigsten Data Mining-Aufgaben. Sie setzt voraus, dass jedem Objekt eine Klasse aus einer vorgegebenen Klasseneinteilung zugeordnet werden kann. Die Aufgabe besteht darin, ein Klassifikationsmodell zu erlernen, welches aus den Eigenschaften des Objektes, d.h. aus dem beschreibenden Datensatz, die korrekte Klasse ableitet. Ein Beispiel für eine Klassifikationsaufgabe sind Banken, die darüber entscheiden müssen, ob ein Kreditantrag bewilligt wird. Das Objekt ist in diesem Fall der potentielle Kreditnehmer und die Klasseneinteilung besteht aus den Klassen kreditwürdig und nicht-kreditwürdig.

- Segmentierung

  Die Segmentierung dient der Aufteilung der Daten in disjunkte Teilmengen. Die Aufteilung soll dabei so erfolgen, dass Objekte in einer Teilmenge möglichst homogen sind. Homogen bedeutet, dass sich die Objekte bzgl. eines vorgegebenen Maßes ähnlich sind. Gleichzeitig sollen die verschiedenen Teilmengen untereinander eine hohe Heterogenität aufweisen. Die Segmentierung hilft z.B. lukrative Kundensegmente im Einzelhandel zu identifizieren.

- Konzeptbeschreibung

  Das Ziel der Konzeptbeschreibung ist es, eine verständliche Beschreibung von Klassen oder Konzepten von Objekten zu finden. Im Gegensatz zur Klassifikation dient die Konzeptbeschreibung weniger der Klassifikation bisher unbekannter Objekte (obwohl sie dafür verwendet werden kann), als der anschaulichen Charakterisierung von Teilmengen der Daten. Ein Beispiel aus der Automobilindustrie betrifft die Aufgabe, eine charakterisierende Beschreibung von Fahrzeugen mit einem bestimmten Schaden zu finden. Häufig ist die

Segmentierung eine notwendige Voraussetzung der Konzeptbeschreibung, um zunächst die Teilmenge der Daten zu identifizieren, für die anschließend ein Konzept gesucht wird.

- Prognose

  Die Prognose ist verwandt mit der Klassifikation. Anstelle einer (symbolischen) Klasse wird jedoch einem Objekt ein numerischer Wert zugeordnet. Zumeist sollen Prognosemodelle eine Aussage über zukünftige Zeitpunkte machen. Ein typisches Beispiel ist die Prognose von Aktienkursen an der Börse.

- Abhängigkeitsanalyse

  Das Ziel der Abhängigkeitsanalyse ist es, signifikante Abhängigkeiten zwischen Eigenschaften von Objekten oder Ereignissen zu finden. Im Einzelhandel kann z.B. eine Abhängigkeit zwischen bestimmten Artikeln bestehen, wenn diese häufig zusammen gekauft werden.

- Abweichungserkennung

  Die Abweichungserkennung identifiziert Datensätze, die nicht den Erwartungen bzw. Normen entsprechen. Solche Ausreißer können zum einen das Resultat von zugrunde liegenden Datenqualitätsproblemen sein. Zum anderen können sie aber auch durch noch unbekannte Phänomene der Anwendung entstehen. In der Finanzdienstleistung wird die Abweichungserkennung bspw. dazu genutzt, um mögliche Betrugsversuche zu erkennen.

Data Mining-Methoden

Eine Data Mining-Methode analysiert gegebene Daten mit einer gezielten Strategie. Die meisten Data Mining-Methoden lassen sich für mehrere der oben genannten Data Mining-Aufgaben einsetzen. Ein Beispiel hierfür sind Entscheidungsbäume. Entscheidungsbäume können sowohl der Klassifikation [Quinlan, 1993], der Konzeptbeschreibung [Blumenstock u. a., 2006] als auch der Prognose (als Regressionsbaum) [Breiman u. a., 1984] dienen.

Die nachfolgende Aufzählung präsentiert einige ausgewählte Data Mining-Methoden zusammen mit den unterstützten Data Mining-Aufgaben und weiterführender Literatur (in Anlehnung an [Borgelt, 2000, Seite 8ff.]). Die Auswahl der Methoden besitzt weder Anspruch auf Vollständigkeit noch soll sie als Indikator für die Wichtigkeit der Methoden dienen. Vielmehr zeigt sie einen Ausschnitt der Methoden, die der Autor als Mitarbeiter eines Datenanalyseteams mit Anwendungen in der Automobilindustrie eingesetzt hat.

- Entscheidungsbäume
  Klassifikation, Konzeptbeschreibung, Prognose
  [Kass, 1980; Breiman u. a., 1984; Quinlan, 1993]

- Assoziationsregeln
  Klassifikation, Konzeptbeschreibung, Abhängigkeitsanalyse, Abweichungserkennung
  [Agrawal u. a., 1993b; Mannila u. a., 1994; Hipp u. a., 2001]

- Probabilistische Netze
  Klassifikation, Abhängigkeitsanalyse
  [Pearl, 1988; Heckerman u. a., 1995; Borgelt, 2002]

- Künstliche Neuronale Netze
  Klassifikation, Segmentierung, Prognose
  [Zell, 1994; Nauck u. a., 1996]

- Clusterverfahren
  Segmentierung, Abweichungserkennung
  [MacQueen, 1967; Everitt, 1993]

- Regelinduktion
  Klassifikation, Konzeptbeschreibung
  [Clark u. Niblett, 1989; Quinlan u. Cameron-Jones, 1995; Joshi u. a., 2001]

- traditionelle statistische Methoden (z.B. Regression, Hauptkomponentenanalyse, etc.)
  Klassifikation, Konzeptbeschreibung, Prognose
  [Agresti, 1990; Everitt, 1992; Bosch, 1996]

- und viele andere mehr

2.2 Muster und Regeln

In diesem Abschnitt werden Data Mining-Methoden vorgestellt, die Muster und Regeln aus einer gegebenen Datengrundlage extrahieren. Von besonderer Bedeutung für die vorliegende Arbeit sind hierbei die Warenkorbanalyse und die mit ihr verbundene Entdeckung von Assoziationsregeln. Beide Methoden werden im folgenden Abschnitt formal definiert und anhand eines Beispiels erläutert. Abschnitt 2.2.2 beschreibt zum Abschluss eine weitere Klasse von Methoden, die mit Hilfe der Regelinduktion Regeln aus den Daten lernen.

2.2.1 Häufige Warenkörbe und Assoziationsregeln

Warenkörbe sind eine spezielle Variante von Mustern, die bei der Wissensentdeckung in Datenbanken eingesetzt werden. Die Warenkorbanalyse kommt zum Einsatz, wenn die Anwendung nach Mustern sucht, die besonders häufig in den Daten auftreten. Typische Fragen aus Sicht der Anwendung lauten:

- Welche Ereignisse geschehen häufig gleichzeitig?

- Welche Eigenschaften treten häufig gemeinsam auf?

- Welche Artikel[1] werden häufig zusammen gekauft?

Sowohl die Warenkorbanalyse als auch Assoziationsregeln werden vor allem für die Data Mining-Aufgabe Abhängigkeitsanalyse eingesetzt.

[1] Der erfolgreiche Einsatz der Warenkorbanalyse im Einzelhandel (z.B. [Agrawal u. a., 1993b]) begründet auch die verwendete Terminologie.

Warenkorbanalyse

Die Warenkorbanalyse wird nachfolgend formal definiert.

Definition 2.1 *(Warenkorb, Itemset) Sei $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ eine nichtleere Menge von Ereignissen (Items). Eine nichtleere Teilmenge $X \subseteq \mathcal{I}$ mit $|X| = k$ heißt k-Warenkorb (k-Itemset).*

Definition 2.2 *(Datengrundlage, Transaktionsdatenbank) Eine Multimenge von Warenkörben zur Ereignismenge $\mathcal{I}$ heißt Transaktionsdatenbank und ist durch D gekennzeichnet. Die Warenkörbe in D werden auch Transaktionen genannt.*

Definition 2.3 *(Support, Häufigkeit) Der Support (die Häufigkeit) eines Warenkorbs X ist definiert durch:*

$$Sup_D(X) = \frac{|\{T \in D : X \subseteq T\}|}{|D|}.$$

Mit Hilfe der Definitionen für die Datengrundlage, der Warenkörbe und des Supports kann die Aufgabe der Warenkorbanalyse wie folgt formuliert werden.

Definition 2.4 *(Warenkorbanalyse, Frequent Itemset Mining) Ein Warenkorb ist* häufig*, wenn sein Support dem minimalen Schwellwert MinSup genügt ($Sup_D(f \in \mathcal{F}) \geq MinSup$). Die Warenkorbanalyse beschreibt die Aufgabe, die Menge $\mathcal{F}$ aller häufigen Warenkörbe in einer gegebenen Transaktionsdatenbank D zu finden.*

Tabelle 2.1 zeigt ein Beispiel für eine Transaktionsdatenbank. Die Transaktionen sind Teil-

Enumeration	Transaktion
T_1	$\{a, c, d\}$
T_2	$\{b, c, e\}$
T_3	$\{a, b, c, e\}$
T_4	$\{b, e\}$
T_5	$\{a, b\}$

Tabelle 2.1: Beispiel für eine Transaktionsdatenbank

mengen der Ereignismenge $\mathcal{I} = \{a, b, c, d, e\}$. Die Menge der häufigen Warenkörbe besteht in diesem Beispiel aus 10 Warenkörben (für einen minimalen Schwellwert von $\frac{2}{5} = 40\%$) und ist in Tabelle 2.2 dargestellt. Die Warenkorbanalyse ist für eine Anwendung meist der erste Schritt, um eine Menge von anwendbaren Regeln zu finden. Der zweite Schritt besteht aus dem Ableiten von sogenannten Assoziationsregeln aus den häufigen Warenkörben.

Assoziationsregeln

Definition 2.5 *(Assoziationsregel) Sind A und B nichtleere disjunkte Warenkörbe (Ereigniskombinationen) aus der Ereignismenge $\mathcal{I}$, so wird die Implikation $A \Rightarrow B$ als Assoziationsregel mit der Prämisse A und der Konklusion B bezeichnet.*

Warenkorb	Support
$\{a\}$	$3/5 = 60\%$
$\{a, b\}$	$2/5 = 40\%$
$\{a, c\}$	$2/5 = 40\%$
$\{b\}$	$4/5 = 80\%$
$\{b, c\}$	$2/5 = 40\%$
$\{b, c, e\}$	$2/5 = 40\%$
$\{b, e\}$	$3/5 = 60\%$
$\{c\}$	$3/5 = 60\%$
$\{c, e\}$	$2/5 = 40\%$
$\{e\}$	$3/5 = 60\%$

Tabelle 2.2: Die häufigen Warenkörbe für MinSup $= \frac{2}{5}$

Assoziationsregel	Support	Konfidenz
$\{a\} \Rightarrow \{b\}$	$2/5 = 40\%$	$2/3 \approx 67\%$
$\{b\} \Rightarrow \{c\}$	$2/5 = 40\%$	$2/4 = 50\%$
$\{b, c\} \Rightarrow \{e\}$	$2/5 = 40\%$	$2/2 = 100\%$
$\{b\} \Rightarrow \{e\}$	$3/5 = 60\%$	$3/4 = 75\%$
$\vdots$	$\vdots$	$\vdots$

Tabelle 2.3: Assoziationsregeln zur Transaktionsdatenbank aus Tabelle 2.1 und MinSup $= \frac{2}{5}$

Die Semantik einer Assoziationsregel entspricht einer *Wenn-Dann* Aussage. *Wenn* eine Transaktion die Ereignisse aus A enthält, *dann* enthält sie auch die Ereignisse aus B. In Tabelle 2.3 sind einige Assoziationsregeln für die Transaktionsdatenbank aus Tabelle 2.1 dargestellt.

Für jede Assoziationsregel können bewertende Gütemaße berechnet werden. Der Support einer Assoziationsregel $A \Rightarrow B$ ist definiert als der Anteil der Transaktionen, für die $A \Rightarrow B$ gültig ist. Das heißt, sowohl Prämisse als auch Konklusion müssen in einer Transaktion vorkommen.

$$\mathrm{Sup}_D(A \Rightarrow B) = \mathrm{Sup}_D(A \cup B) = \frac{|\{T \in D : A \cup B \subseteq T\}|}{|D|}$$

Als Ergänzung zum Support einer Regel dient die Konfidenz. Sie bewertet die Korrektheit der Regel, indem sie den Support der Regel ins Verhältnis zum Support der Prämisse stellt.

$$\mathrm{Conf}_D(A \Rightarrow B) = \frac{\mathrm{Sup}_D(A \Rightarrow B)}{\mathrm{Sup}_D(A)} = \frac{|\{T \in D : A \cup B \subseteq T\}|}{|\{T \in D : A \subseteq T\}|}$$

Die Konfidenz ist identisch mit der bedingten Wahrscheinlichkeit $P(B|A)$. Sie kann als die Wahrscheinlichkeit interpretiert werden, dass eine zufällig gezogene Transaktion der Datenbank, welche die Prämisse A enthält, auch die Konklusion B enthält. Neben dem Support und der Konfidenz existieren eine Vielzahl weiterer Gütemaße zur Bewertung von Assoziationsregeln (siehe z.B. [Sahar, 1999; Shah u. a., 1999; Tan u. Kumar, 2000; Tan u. a., 2002; McGarry, 2005]).

Die Anzahl der möglichen Assoziationsregeln wächst exponentiell mit der Kardinalität von $\mathcal{I}$ [Hipp, 2003, Seite 28]. Für die Ereignismenge aus Tabelle 2.1 existieren bereits nahezu 180 verschiedene Assoziationsregeln. Um die Anzahl der Assoziationsregeln einzugrenzen, werden nur solche Assoziationsregeln betrachtet, die sich auf häufige Warenkörbe beziehen (d.h. $A \cup B$ ist ein häufiger Warenkorb). Eine weitere Möglichkeit die Regelmenge zu beschränken besteht darin, einen Schwellwert für die minimale Konfidenz einer Regel einzuführen (siehe z.B. [Agrawal u. a., 1996]).

Die Extraktion von Assoziationsregeln erfolgt direkt auf Basis der häufigen Warenkörbe. Für jeden häufigen Warenkorb W, werden alle Assoziationsregeln der Form $A \Rightarrow (W - A)$ gebildet. Hierbei kennzeichnet A eine echte Teilmenge von W ($A \subset W$). In der allgemeinen Form kann die Konklusion einer Assoziationsregel beliebig viele Ereignisse enthalten. Eine praxisrelevante Einschränkung besteht darin, dass nur solche Regeln abgeleitet werden, deren Konklusion nur ein Ereignis (Item) enthält ($|W - A| = 1$).

Algorithmen zur Warenkorbanalyse

Im Gegensatz zur Gewinnung der Assoziationsregeln aus den häufigen Warenkörben, ist die Suche nach häufigen Warenkörben in einer Transaktionsdatenbank ungleich schwieriger. Die Anzahl der potentiell häufigen Warenkörbe steigt exponentiell mit der Anzahl der Ereignisse in $\mathcal{I}$ (vgl. Satz 4.18 auf Seite 67). Ein Algorithmus der alle möglichen Warenkörbe auf ihre Häufigkeit hin untersucht, ist daher selbst für kleinste Transaktionsdatenbanken praktisch nicht anwendbar.

Von zentraler Bedeutung für alle effizienten Verfahren zur Warenkorbanalyse ist das Apriori-Kriterium. Mit seiner Hilfe kann die Anzahl der zu überprüfenden Warenkörbe erheblich eingeschränkt werden.

Satz 2.6 *(Apriori-Kriterium der Warenkorbanalyse, Abgeschlossenheit des Supports) Sei W ein häufiger Warenkorb in der Transaktionsdatenbank D bzgl. der minimalen Supportschwelle MinSup ($Sup_D(W) \geq MinSup$). Dann ist jede Teilmenge von W mindestens so häufig wie W:*

$$\forall A \subset W : Sup_D(A) \geq Sup_D(W).$$

Die Gültigkeit des Apriori-Kriteriums folgt direkt aus der Definition des Supports für Warenkörbe (vgl. auch [Agrawal u. Srikant, 1994a; Mannila u. a., 1994]).

Einer der ältesten Algorithmen zur Warenkorbanalyse, der die Abgeschlossenheit des Supports ausnutzt, ist Apriori [Agrawal u. Srikant, 1994a; Agrawal u. a., 1996]. Stellvertretend für alle anderen Verfahren soll der Apriori-Ansatz zur Entdeckung aller häufigen Warenkörbe näher erläutert werden. Algorithmus 2.1 gibt den Pseudocode von Apriori auf einer allgemeinen Ebene wieder. Apriori unterteilt die Suche nach den häufigen Warenkörben in zwei Schritte: der Kandidatengenerierung (Zeile 17 in Algorithmus 2.1) und der Supportevaluation (Zeile 11). Die Kandidatengenerierung erzeugt potentiell häufige Warenkörbe, deren Support die Supportevaluation ermittelt. Beide Schritte werden solange abwechselnd wiederholt bis keine weiteren Kandidaten generiert werden können (Schleife über die Zeilen 10 – 19).

Apriori startet mit den häufigen 1-Warenkörben und evaluiert ihren Support anhand der Transaktionsdatenbank. Damit sind bereits alle häufigen 1-Warenkörbe ermittelt. Anschließend generiert Apriori die potentiell häufigen 2-Warenkörbe (2-Kandidaten) ausschließlich auf

Algorithmus 2.1 Apriori – Algorithmus zum Auffinden aller häufigen Warenkörbe

```
1: D: Transaktionsdatenbank
2: 𝓘: Ereignismenge der Transaktionsdatenbank D
3: MinSup: minimaler Support
4: C_i: Menge aller Kandidatenmuster in der i-ten Iteration (i-Kandidaten)
5: F_i: Menge aller häufigen Warenkörbe in der i-ten Iteration (i-Warenkörbe)
6:
7: procedure APRIORI(D, 𝓘)
8:     k := 1
9:     C_k := 𝓘
10:    repeat
11:        evaluateSupport(C_k, D)
12:        for all c ∈ C_k do
13:            if getSupport(c) ≥ MinSup then
14:                F_k := F_k ∪ {c}
15:            end if
16:        end for
17:        C_{k+1} := generateCandidates(F_k)
18:        k := k+1
19:    until C_k = ∅
20:    return F_{1...k}
21: end procedure
```

Basis der häufigen 1-Warenkörbe. Wiederum erfolgt die Evaluation des Supports der Kandidaten und der Prozess wiederholt sich bis alle häufigen Warenkörbe gefunden sind. Die zugrunde liegende Idee besteht darin, dass die Kandidatengenerierung durch Verknüpfung der häufigen k-Warenkörbe eine Obermenge aller häufigen Warenkörbe mit $k+1$ Items erzeugt. Diese Vorgehensweise ist durch das Apriori-Kriterium gerechtfertigt, da jede k-elementige Teilmenge eines häufigen $(k+1)$-Warenkorbs selbst häufig sein muss. Die Menge der $(k+1)$-Kandidaten wiederum ist eine Teilmenge aller möglichen $(k+1)$-Warenkörbe. Insbesondere fehlen in der Kandidatenmenge $(k+1)$-Warenkörbe, die nicht-häufige k-Warenkörbe enthalten. Diese können auf Grund des Apriori-Kriteriums nicht häufig sein und brauchen während der Supportevaluation nicht überprüft werden. Die Differenzmenge zwischen der Menge aller $(k+1)$-Warenkörbe und der Menge $(k+1)$-Kandidaten begründet die Effizienz von Apriori. In Abhängigkeit von der gewählten minimalen Supportschwelle braucht ein erheblicher Teil der möglichen Warenkörbe nicht auf seinen Support untersucht werden.

Im Laufe der Zeit sind eine Vielzahl weiterer effizienter Algorithmen zur Warenkorbanalyse vorgeschlagen worden. Für einen allgemeinen Überblick und Vergleich der einzelnen Verfahren sei unter anderem auf [Hipp u. a., 2000], [Hipp, 2003, Kapitel 3] und [Han u. a., 2007] verwiesen.

2.2.2 Regelinduktion durch sequentielles Abdecken

Assoziationsregeln sind nicht der einzige Regeltyp, der aus einer gegebenen Datenmenge abgeleitet werden kann. Insbesondere für die Data Mining-Aufgabe Klassifikation gibt es viele

Verfahren, die *Wenn-Dann* Regeln aus den Daten ableiten. Die Unterschiede zwischen Assoziationsregeln und Klassifikationsregeln sind vor allem durch die verschiedenen Zielsetzungen der Data Mining-Aufgaben Abhängigkeitsanalyse und Klassifikation begründet. Klassifikationsverfahren fokussieren auf solche Regeln bzw. Regelmengen, welche die Klassenzugehörigkeit neuer Objekte gut bestimmen können. Zum einen enthält dadurch die Konklusion der Klassifikationsregeln ausschließlich die gesuchte Zielklasse (im Gegensatz zu beliebigen Ereignissen) und zum anderen sind Klassifikationsregeln nicht zwingend auf Regeln beschränkt, die einer minimalen Supportschwelle genügen.

Unter den Verfahren zur Induktion von Klassifikationsregeln ist die algorithmische Strategie der *sequentiellen Abdeckung* (sequential covering) weit verbreitet. Verfahren die sich dieser Strategie zuordnen lassen sind unter anderem CN2 [Clark u. Niblett, 1989], GoldDigger [Riddle u. a., 1994], RIPPER [Cohen, 1995], FOIL [Quinlan u. Cameron-Jones, 1995], PRIM [Friedman u. Fisher, 1999] oder CN2-SD [Lavrac u. a., 2004].

Die Datengrundlage für die Verfahren der sequentiellen Abdeckung besteht aus Objekten (bzw. Instanzen) mit bekannter Klassenzuordnung, deren Eigenschaften durch eine Reihe von Attributen näher beschrieben sind. Die Prämisse einer Klassifikationsregel setzt sich aus Konjunktionen von Attribut-Wert Paaren zusammen. Ein Beispiel mit zwei Konjunktionsgliedern ist:

$$\text{Form} = \text{Ball} \wedge \text{Farbe} = \text{Gelb} \Rightarrow \text{Spiel} = \text{Tennis}.$$

Die grundlegende Idee der sequentiellen Abdeckung besteht darin, aus einer gegebenen Datenmenge eine einzige Regel zu extrahieren, die mit möglichst hoher Korrektheit die gewünschte Zielklasse vorhersagt. Alle von der Regel korrekt klassifizierten Objekte werden anschließend aus der Datenmenge entfernt. Die so entstandene neue Datengrundlage ist Ausgangspunkt für weitere Iterationen desselben Vorgangs. Daher wird erneut eine Regel in der nun modifizierten Datengrundlage gesucht. Auf diese Weise erweitert sich schrittweise die Menge der gefundenen Klassifikationsregeln bis entweder alle Objekte der Zielklasse entfernt wurden oder die letzte gefundene Regel den minimalen Güteanforderungen nicht mehr genügt. Der Pseudocode der sequentiellen Abdeckung ist in Algorithmus 2.2 dargestellt. Algorithmus 2.2 sucht zunächst eine erste Regel auf der gesamten Datengrundlage (Hilfsfunktion *findSingleRule* Zeile 7). Anschließend beginnt die Schleife der sequentiellen Abdeckung, in der die zuvor gefundene Regel der Regelmenge hinzugefügt wird (Zeile 9) und Objekte aus der Datengrundlage entfernt werden, welche durch die Regel korrekt klassifiziert werden (Zeile 10). Vor dem nächsten Schleifendurchlauf wird nach einer neuen Regel auf den modifizierten Daten gesucht (Zeile 11). Die Hilfsfunktion *performance* evaluiert die Güte der zuletzt gefundenen Regel (Zeile 8) und dient als Abbruchkriterium für die Regelsuche.

Die genannten Verfahren zur Regelinduktion variieren das Schema der sequentiellen Abdeckung an mehreren Stellen. Die folgende Auflistung zeigt einige der Ansatzpunkte für Abweichungen der konkreten Verfahren.

- Regelsprache: Einige Verfahren erlauben in der Prämisse auch die Verwendung von Konjunktionsgliedern, wie z.B. Farbe $\in$ {Rot, Blau, Grün} oder Farbe $\neq$ Blau.
- Numerische Attribute: Einige Verfahren können direkt mit numerischen Attributen im Datensatz umgehen und bilden in der Prämisse Konjunktionsglieder mit Schwellwerten, z.B. Alter > 40.

Algorithmus 2.2 Sequentielles Abdecken – Erlernen von Klassifikationsregeln

```
1: D: Datengrundlage
2: C: Zielklasse der Klassifikationsaufgabe
3: P_T: Schwellwert für die minmale Güte einer gefundenen Regel (Abbruchkriterium)
4:
5: procedure SequentialCovering(C, D, P_T)
6:     ruleset := ∅
7:     rule := findSingleRule(C, D)
8:     while performance(rule, D) ≥ P_T do
9:         ruleset := ruleset ∪ {rule}
10:        D := D − {objects correctly classified by rule}
11:        rule := findSingleRule(C, D)
12:    end while
13:    return ruleset
14: end procedure
```

- Regelauswahl: Die Hilfsfunktion *findSingleRule* wird von jedem Verfahren unterschiedlich implementiert. Es kommen unterschiedliche Suchverfahren für die Regel (z.B. Strahlensuche, Gradientenabstieg) und verschiedene heuristische Bewertungsmaße zum Einsatz.
- Backtracking: Sequentielles Abdecken ist eine heuristische (gierige) Suche nach einer Regelmenge zur Klassifikation. Es ist nicht garantiert, dass die gefundene Regelmenge die beste oder kleinste ist. Um diesen Nachteil abzumildern, implementieren einige Verfahren eine Backtracking-Technik.
- Gütemaß: Die durch die Hilfsfunktion *performance* durchgeführte Bewertung einer Regel, wird von den Verfahren unterschiedlich realisiert.

Für eine detaillierte Einführung in die sequentielle Abdeckung sei auf [Mitchell, 1997, Kapitel 10] verwiesen. Des Weiteren vergleichen [Klösgen, 2002] und [Fürnkranz, 1999] mehrere Verfahren.

Kapitel 3

Entdeckung temporaler Muster

Dieses Kapitel beleuchtet existierende Ansätze zur Entdeckung von Mustern und Regeln in zeitbezogenen Daten. Dem Ziel dieser Arbeit folgend, liegt der Schwerpunkt im ersten Teil des Kapitels auf Verfahren, die häufige Muster entdecken. Anschließend (Abschnitt 3.3) werden Verfahren besprochen, die aus einem Datensatz Regeln lernen oder mit numerischen Zeitreihen arbeiten.

Die Abschnitte 3.1 und 3.2 geben einen Überblick über etablierte Verfahren zur Entdeckung häufiger temporaler Muster. In der Literatur gibt es viele Verfahren, die trotz einer ähnlichen Problemstellung entscheidende Unterschiede aufweisen. Die Unterschiede zeigen sich vor allem in der Art der Ausgangsdaten, der verwendeten Mustersprache (Hypothesensprache), der Supportdefinition und der algorithmischen Lösungsansätze. Um dieser Vielfalt gerecht zu werden, wird im vorliegenden Kapitel eine Systematisierung der vorgestellten Verfahren vorgenommen. Als Basis der Systematisierung dient die Art der Ausgangsdaten mit zwei relevanten Eigenschaften. Zum einen können die Ausgangsdaten ereignis- oder intervallbasiert sein und zum anderen können sie ein oder mehrere Objekte der realen Welt beschreiben. Ereignisbasierte Daten kennzeichnen punktuelle Ereignisse anhand eines einzelnen Zeitstempels. Im Gegensatz dazu können intervallbasierte Daten auch Ereignisse ausdrücken, die sich über einen längeren Zeitraum erstrecken. Daten, die ein einzelnes reales Objekt charakterisieren, sind z.B. der Verlauf des DAX an der Frankfurter Börse oder der Temperaturverlauf auf der Zugspitze während eines Monats. Dem gegenüber stehen Daten die mehrere reale Objekte beschreiben, wie z.B. der Temperaturverlauf aller Wetterstationen in Deutschland. Ereignisse und Intervalle die Aussagen über dasselbe reale Objekt machen, werden in den Ausgangsdaten für gewöhnlich in einer Eingabesequenz zusammengefasst.

Für jede Kombination aus ereignis- oder intervallbasierten und einer oder mehreren Ein-

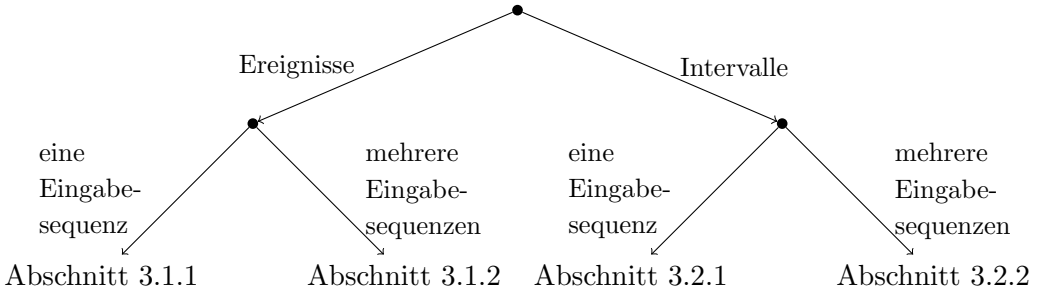


Abbildung 3.1: Systematisierung der Verfahren anhand der Ausgangsdaten und dazugehörige Abschnitte in diesem Kapitel

gabesequenzen existieren Verfahren, welche die häufigen temporalen Muster extrahieren. Die Grundannahme dieser Systematisierung besteht darin, dass Verfahren mit gleicher Datenbasis auch ein vergleichbares Problem lösen. Diese intuitive Kategorisierung der Verfahren ist in bisherigen Übersichtsarbeiten und Bibliographien zum Thema nicht verwendet worden (vgl. [Roddick u. Spiliopoulou, 1999, 2002; Roddick u. a., 2000; Antunes u. Oliveira, 2001; Zhao u. Bhowmick, 2003; Mörchen, 2007]).

Im Folgenden werden die vier Kategorien der Systematisierung (siehe Abbildung 3.1) mit ihren wichtigsten Verfahren vorgestellt.

3.1 Häufige Muster in ereignisbasierten Daten

Zunächst werden Verfahren beschrieben, die häufige Muster in Abfolgen von punktuellen Ereignissen suchen.

3.1.1 Eine Eingabesequenz

Die erste Gruppe von Verfahren untersucht Ausgangsdaten, die aus einer einzelnen Eingabesequenz von Ereignissen (S) besteht. Abbildung 3.2 zeigt ein Beispiel für diese einfachste Art der Datenbasis.

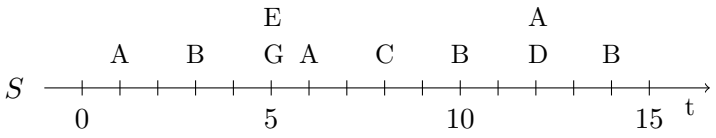


Abbildung 3.2: Eine einzelne Eingabesequenz aus Ereignissen

WINEPI und MINEPI [Mannila u. a., 1995, 1997; Mannila u. Toivonen, 1996]

Mannila u. a. beschreiben die Algorithmen WINEPI und MINEPI. Beide Algorithmen entdecken häufige Episoden, jedoch mit Hilfe verschiedener Supportdefinitionen. Eine Episode ist ein Muster in der Eingabesequenz und ist durch eine Menge von Ereignissen sowie einer partiellen Ordnung über dieser Menge spezifiziert. In Abbildung 3.3 sind zwei Beispiele für Episoden dargestellt. Die Episode in Abbildung 3.3a besteht aus den Ereignissen A und B. Die partielle Ordnung der Episode bestimmt, dass A vor B auftreten muss. Im Gegensatz dazu ist die Reihenfolge der Ereignisse A und B in Abbildung 3.3b unbestimmt. A kann zeitlich sowohl vor als auch nach B liegen. Beide Ereignisse müssen aber vor dem Ereignis C liegen.

WINEPI benutzt ein gleitendes Zeitfenster, um den Support einer Episode zu bestimmen. Das Zeitfenster, mit der benutzerdefinierten Breite Δt, wird schrittweise über die Eingabesequenz geschoben. Jede Fensterposition, in der eine Episode beobachtbar ist, trägt zum Support der Episode bei. Aus dem Verhältnis der Zeitfenster mit einer Episode E zu der Anzahl aller möglichen Fensterpositionen ergibt sich der Support (hier *frequency* genannt) der Episode E:

$$\text{frequency}(E) = \frac{\text{Anzahl der Fenster mit } E}{\text{Anzahl aller Fenster}}.$$

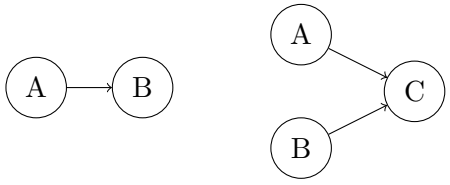


Abbildung 3.3: Beispiele für Episoden

Der Algorithmus MINEPI verwendet die Anzahl der minimalen Vorkommen (minimal occurrences) einer Episode als Supportdefinition. Ein minimales Vorkommen einer Episode E ist ein Zeitintervall $[b, e]$ der Eingabesequenz, falls

1. der durch $[b, e]$ spezifizierte Abschnitt der Eingabesequenz ein Vorkommen von E enthält und

2. es kein echtes Teilintervall $[b', e']$ von $[b, e]$ gibt, welches ebenfalls ein Vorkommen von E enthält.

Als Beispiel diene die Episode $A \rightarrow B$ (Abbildung 3.3a) in der Eingabesequenz aus Abbildung 3.2. Das Zeitintervall $[4, 12]$ ist kein minimales Vorkommen, da die Episode z.B. auch im Teilintervall $[6, 11]$ enthalten ist. Ebenso ist $[3, 7]$ kein minimales Vorkommen, da in diesem Abschnitt der Intervallsequenz $A \rightarrow B$ nicht vorkommt. Der Support von $A \rightarrow B$ beträgt 3 mit den minimalen Vorkommen $[1, 3]$, $[6, 10]$, $[12, 14]$.

Ungeachtet der unterschiedlichen Supportdefinitionen verfolgen beide Algorithmen den Apriori-Ansatz, um alle häufigen Episoden zu entdecken. Daher bestehen beide Verfahren aus den sich abwechselnden Phasen der Kandidatengenerierung und Supportevaluation. Zuerst werden die häufigen Episoden bestimmt, die nur aus einem Ereignis bestehen. Anschließend ermittelt die Kandidatengenerierung eine Obermenge der häufigen Episoden mit zwei Ereignissen. Der Support der Kandidaten wird in der nachfolgenden Supportevaluation bestimmt. Die gefundenen häufigen Episoden mit zwei Ereignissen bilden wiederum den Ausgangspunkt für die nächste Kandidatengenerierung.

Häufige Zeichenketten [Vilo, 1998]

Eine Zeichenkette (String) stellt einen Spezialfall der Eingabedaten dar. Hier sind die Ereignisse Zeichen eines definierten Alphabets und ihr Zeitstempel entspricht der Position eines Zeichens in der Zeichenkette. Des Weiteren können in einer Zeichenkette zwei Ereignisse (Zeichen) nicht denselben Zeitstempel (dieselbe Position) besitzen.

Vilo adaptiert für die Suche aller häufigen Teilzeichenketten (Substrings) die Datenstruktur des Suffixbaumes. Suffixbäume sind etablierte Datenstrukturen zum Auffinden der Vorkommen gegebener Zeichenketten in einer langen Zeichenkette (vgl. [McCreight, 1976; Ukkonen, 1995]). Vilo geht in zwei Schritten vor, um alle häufigen Zeichenketten zu finden. Im ersten Schritt

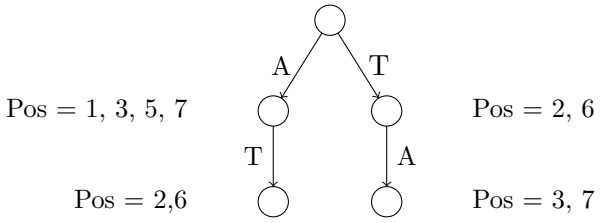


Abbildung 3.4: Suffixbaum für häufige Zeichenketten im String ATACATA (MinSup= 2) [Vilo, 1998]

erstellt er den Suffixbaum unter Berücksichtigung des minimalen Supportschwellwertes. Anschließend wird der Suffixbaum zur Ausgabe der häufigen Zeichenketten durchlaufen. In Abbildung 3.4 ist der Suffixbaum für die Zeichenkette ATACATA und einen minimalen Schwellwert MinSup= 2 dargestellt. Jeder Pfad im Suffixbaum von der Wurzel zu einem Knoten repräsentiert eine häufige Zeichenkette. Während jede Kante dem Pfad ein weiteres Zeichen hinzufügt, speichern die Knoten die Positionen der Teilzeichenkette. Das Zeichen A kommt viermal in ATACATA vor. Daher sind die Positionen 1, 3, 5 und 7 in den entsprechenden Konten des Suffixbaumes gespeichert. In zwei Fällen kann A zu AT erweitert werden. Somit besitzt A einen Kindknoten mit den Positionen 2 und 6 und der Kante T. Durch Durchlaufen des Suffixbaumes können alle anderen häufigen Zeichenketten ermittelt werden (A, AT, T und TA).

Vilo verallgemeinert den Ansatz der Suffixbäume, um auch häufige Zeichenketten mit Platzhaltern (Wildcards) zu identifizieren. Ein solches Muster ist z.B. A*BT*C.

Gen-FCE und MOWCATL [Harms u. a., 2001, 2002; Harms u. Deogun, 2004]

Harms u. a. [2001] entwickelten den Algorithmus Gen-FCE (*Gen*erating *F*requent *C*losed *E*pisodes). Gen-FCE erweitert den Ansatz von WINEPI für benutzerdefinierte Randbedingungen und geschlossene Episoden. Analog zu geschlossenen Warenkörben ist eine geschlossene Episode E nicht Teil einer längeren Episode E' mit gleichem Support ($E \subset E' \rightarrow \text{Sup}(E) \neq \text{Sup}(E')$). Die benutzerdefinierten Randbedingungen werden genutzt, um die Menge der resultierenden Episoden einzuschränken.

Analog zu Gen-FCE erweitert MOWCATL (*M*inimal *O*ccurrences *W*ith *C*ostraints *A*nd *T*ime *L*ags) MINEPI um benutzerdefinierte Randbedingungen. Auch hier werden die Randbedingungen in den Algorithmus integriert, um möglichst nur einen kleinen Teil des Hypothesenraums durchsuchen zu müssen. Im Gegensatz zu MINEPI ist die Generierung von Regeln aus den gefundenen Episoden ein integrierter Bestandteil von MOWCATL.

Anwendungen

Zwei Anwendungen zu häufigen Episoden bilden den Abschluss dieses Abschnittes.

- Hätönen u. a. [1996] analysieren auftretende Alarme in einem Telekommunikationsnetzwerk. Ein Alarm ist hierbei eine Ausnahmesituation, die von einer Komponente des Netzwerks (z.B. Vermittlungsstellen, Softwaremodule, etc.) angezeigt wird. Hätönen u. a. finden mit Hilfe von WINEPI die häufigen Episoden und leiten Regeln aus ihnen ab. In

einem zweiten Schritt werden uninteressante Regeln durch einen Experten entfernt. Die übrigen Regeln werden nach ihrer semantischen Aussage gruppiert und priorisiert. Das so gebildete Regelwerk dient dem Experten als Entscheidungshilfe bei neu auftretenden Alarmen im Netzwerk.

- Harms u. Deogun [2004] beschreiben die Anwendung ihres Verfahrens MOWCATL auf Wetterdaten. Ziel der Analyse ist die Identifikation und Vorhersage von Dürreperioden für die Landwirtschaft. Die gefundenen Episoden, sowie die aus ihnen abgeleiteten Regeln, werden von einem Experten überprüft und zur Risikobewertung herangezogen.

3.1.2 Mehrere Eingabesequenzen

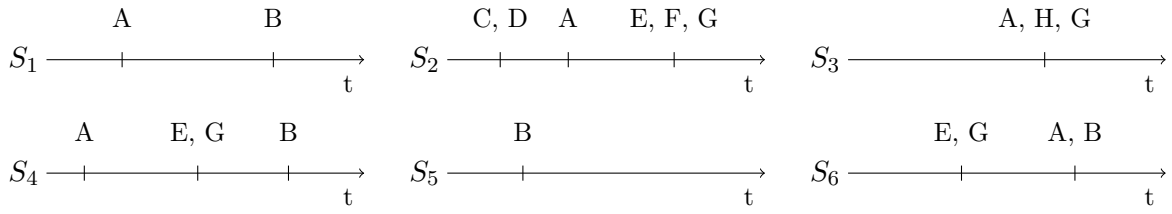


Abbildung 3.5: Ein Beispieldatensatz mit sechs Eingabesequenzen. Jede Eingabesequenz wird durch eine Reihe von Ereignissen näher beschrieben.

Das in der Literatur am stärksten untersuchte Problem ist die Entdeckung häufiger Muster aus ereignisbasierten Daten, die mehrere Objekte der realen Welt beschreiben (siehe Abbildung 3.5). Gemäß der allgemeinen Terminologie, wird das Problem nachfolgend als Sequenzanalyse bezeichnet. Es ist interessant, dass alle relevanten Veröffentlichungen zu diesem Thema die ursprüngliche Problemdefinition aus [Agrawal u. Srikant, 1995] bzw. die erweiterte Problemdefinition aus [Srikant u. Agrawal, 1996] nahezu unverändert übernommen haben. Im Gegensatz dazu sind bei den in den Abschnitten 3.1.1, 3.2.1 und 3.2.2 beschriebenen Problemen auch verschiedene Hypothesensprachen und Supportdefinitionen vorgeschlagen worden. Nach über einem Jahrzehnt gibt es eine Vielzahl von Verfahren zur Sequenzanalyse, die sich nur auf Grund ihres algorithmischen Ansatzes unterscheiden.

Im Folgenden wird zunächst die Sequenzanalyse in Analogie zu [Agrawal u. Srikant, 1995] formal definiert.

Definition 3.1 *(Sequenz) Eine Sequenz S ist eine Abfolge von Warenkörben (w_i): $S = w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_n$. Eine Sequenz mit k Warenkörben wird auch k-Sequenz oder Sequenz der Länge k genannt. Der Begriff „n-elementige Sequenz“ kennzeichnet eine Sequenz, deren Warenkörbe in Summe n Items enthalten.*

Definition 3.2 *(Teilsequenz) Eine Sequenz $A = w_1^A \rightarrow w_2^A \rightarrow \cdots \rightarrow w_n^A$ ist Teilsequenz der Sequenz $B = w_1^B \rightarrow w_2^B \rightarrow \cdots \rightarrow w_m^B$ ($A \subseteq B$), wenn es eine sortierte Liste von ganzen Zahlen $i_1 < i_2 < \cdots < i_n$ gibt, so dass $w_1^A \subseteq w_{i_1}^B, w_2^A \subseteq w_{i_2}^B, \ldots, w_n^A \subseteq w_{i_n}^B$ erfüllt ist.*

Als Beispiel diene die Sequenz $\{A\} \rightarrow \{A, B\} \rightarrow \{C, D\}$. Sie ist eine Teilsequenz von $\{E\} \rightarrow \{A, C\} \rightarrow \{A, B, E, F\} \rightarrow \{F\} \rightarrow \{C, D\}$, da $\{A\} \subseteq \{A, C\}, \{A, B\} \subseteq \{A, B, E, F\}$ und $\{C, D\} \subseteq \{C, D\}$ gilt.

Definition 3.3 *(Sequenzdatenbank) Eine Menge von Tupeln (sid, S), bestehend aus einer Sequenz S und einer Sequenzidentifikation sid, wird als Sequenzdatenbank bezeichnet. Für eine Sequenzdatenbank D ist die Anzahl der enthaltenen Sequenzen durch $|D| = N$ gegeben. Eine Sequenz aus der Sequenzdatenbank wird auch Eingabesequenz genannt.*

Abbildung 3.5 zeigt eine Sequenzdatenbank bestehend aus sechs Sequenzen, $sid = S_1, \ldots, S_6$.

Definition 3.4 *(Support, Häufigkeit) Der Support (die Häufigkeit) einer Sequenz A in einer Sequenzdatenbank D ist die Anzahl der Tupel in D, die A als Teilsequenz enthalten.*

$$Sup_D(A) = |\{(sid, S) : (sid, S) \in D \wedge A \subseteq S\}|$$

Häufig wird auch der relative Support verwendet. Dieser gibt den Anteil der Sequenzen in der Sequenzdatenbank an, welche die gegebene Sequenz als Teilsequenz enthalten ($\mathrm{Sup}_D(A) = \frac{|\{(\mathrm{sid},S):(\mathrm{sid},S)\in D\wedge A\subseteq S\}|}{N}$). Auf Basis der vorangegangenen Definitionen kann die Sequenzanalyse definiert werden.

Definition 3.5 *(Sequenzanalyse) Finde alle Sequenzen in der Sequenzdatenbank D, deren Support größer ist als der vorgegebene Schwellwert MinSup ($Sup_D(S) \geq MinSup$).*

AprioriAll [Agrawal u. Srikant, 1994b, 1995]

AprioriAll ist der erste Algorithmus, der zur Sequenzanalyse vorgestellt worden ist. Wie der Name bereits andeutet, nutzt AprioriAll den Apriori-Ansatz um alle häufigen Sequenzen zu entdecken (siehe Abschnitt 2.2.1 oder [Agrawal u. a., 1993b]). Die Durchführung des Apriori-Ansatzes ist jedoch erst der letzte Schritt in einem 3-schrittigen Prozess, bestehend aus:

1. Bestimmung der häufigen 1-Sequenzen (Warenkörbe),
2. Transformation der Sequenzdatenbank,
3. Bestimmung aller häufigen Sequenzen mit AprioriAll.

Im ersten Schritt werden alle häufigen Sequenzen der Länge 1 bestimmt. Alle 1-Sequenzen bestehen lediglich aus einem einzigen Warenkorb (siehe Definition 3.1). Daher kann dieser Schritt mit Hilfe von Algorithmen zur Warenkorbanalyse durchgeführt werden (z.B. wie in [Agrawal u. a., 1993b]). Der einzige Unterschied besteht in einer Anpassung der Supportdefinition. Bei der Sequenzanalyse darf der Supportzähler eines Warenkorbes nur einmal für jede Sequenz der Sequenzdatenbank inkrementiert werden, auch wenn die Sequenz den Warenkorb mehrmals enthält (vgl. Definitionen 3.4 und 2.3). Für die Sequenzdatenbank aus Abbildung 3.5 liefert dieser erste Schritt bei einem Minimumsupport von 2 fünf häufige 1-Sequenzen (siehe Tabelle 3.1).

Im zweiten Schritt wird die Sequenzdatenbank in eine alternative Darstellung transformiert. Das Ziel der Transformation ist es, sehr schnell testen zu können, ob eine gegebene Sequenz in der Sequenzdatenbank enthalten ist. Dazu werden zunächst alle häufigen 1-Sequenzen enumeriert (siehe letzte Spalte in Tabelle 3.1). Anschließend werden alle Warenkörbe der Sequenzdatenbank durch die Menge der enthaltenen häufigen 1-Sequenzen ersetzt. Zuletzt wird

1-Sequenz	Support	Enumeration
{A}	5	1
{B}	4	2
{E}	3	3
{G}	4	4
{E, G}	3	5

Tabelle 3.1: Häufige 1-Sequenzen zur Sequenzdatenbank aus Abbildung 3.5 (MinSup= 2)

sid	ursprüngliche Sequenz	transformierte Sequenz	nach Abbildung
S_1	$\{A\} \rightarrow \{B\}$	$\{A\} \rightarrow \{B\}$	$\{1\} \rightarrow \{2\}$
S_2	$\{C, D\} \rightarrow \{A\} \rightarrow \{E, F, G\}$	$\{A\} \rightarrow \langle\{E\}, \{G\}, \{E, G\}\rangle$	$\{1\} \rightarrow \{3, 4, 5\}$
S_3	$\{A, H, G\}$	$\langle\{A\}, \{G\}\rangle$	$\{1, 4\}$
S_4	$\{A\} \rightarrow \{E, G\} \rightarrow \{B\}$	$\{A\} \rightarrow \langle\{E\}, \{G\}, \{E, G\}\rangle \rightarrow \{B\}$	$\{1\} \rightarrow \{3, 4, 5\} \rightarrow \{2\}$
S_5	$\{B\}$	$\{B\}$	$\{2\}$
S_6	$\{E, G\} \rightarrow \{A, B\}$	$\langle\{E\}, \{G\}\{E, G\}\rangle \rightarrow \langle\{A\}, \{B\}\rangle$	$\{3, 4, 5\} \rightarrow \{1, 2\}$

Tabelle 3.2: Transformation der Sequenzdatenbank

die Enumeration der häufigen 1-Sequenzen genutzt, um eine komprimierte Abbildung der ursprünglichen Sequenzdatenbank zu erhalten. Tabelle 3.2 zeigt die Abfolge der einzelnen Transformationsschritte anhand der Sequenzdatenbank aus Abbildung 3.5.

Der dritte Schritt ist der eigentliche Startpunkt von AprioriAll. Auf Basis der häufigen 1-Sequenzen und der transformierten Sequenzdatenbank beginnt die Suche nach allen häufigen Sequenzen mit Hilfe der Kandidatengenerierung und Supportevaluation. Die Lösungsidee ist hierbei identisch zum Apriori Algorithmus der Warenkorbanalyse. Zunächst ermittelt AprioriAll die häufigen Sequenzen der Länge k. Aus ihnen werden die Kandidatensequenzen der Länge $k+1$ abgeleitet. Schließlich bestimmt AprioriAll den Support der Kandidatensequenzen anhand der Sequenzdatenbank. Diese beiden Schritte werden so lange wiederholt, bis keine weiteren Kandidatensequenzen mehr erzeugt werden können.

Für die Kandidatengenerierung benutzt AprioriAll alle Paare von häufigen k-Sequenzen, deren erste $k-1$ Warenkörbe identisch sind. Aus jedem dieser Paare entsteht ein neuer Kandidat der Länge $k+1$. Tabelle 3.3 verdeutlicht diese Vorgehensweise an einem Beispiel. Unter den

häufige 3-Sequenzen	Kandidatensequenzen der Länge 4	Kandidaten nach Stutzen
$1 \rightarrow 4 \rightarrow 3$	$3 \rightarrow 2 \rightarrow 1 \rightarrow 2$	$2 \rightarrow 1 \rightarrow 4 \rightarrow 3$
$3 \rightarrow 2 \rightarrow 1$	$3 \rightarrow 2 \rightarrow 2 \rightarrow 1$	
$3 \rightarrow 2 \rightarrow 2$	$2 \rightarrow 1 \rightarrow 4 \rightarrow 3$	
$2 \rightarrow 1 \rightarrow 4$	$2 \rightarrow 1 \rightarrow 3 \rightarrow 4$	
$2 \rightarrow 1 \rightarrow 3$		
$2 \rightarrow 4 \rightarrow 3$		

Tabelle 3.3: Kandidatengenerierung von AprioriAll

sechs 3-Sequenzen gibt es zwei Paare mit einer gemeinsamen 2-Sequenz als Anfang. Aus diesen beiden Paaren entstehen durch Vereinigung jeweils zwei Kandidatensequenzen der Länge 4. An dieser Stelle stutzt AprioriAll die Menge der erzeugten Kandidaten mit Hilfe des Apriori-Kriteriums. Analog zu Warenkorbanalyse muss auch jede Teilsequenz einer häufigen Sequenz selbst häufig sein. Im obigen Beispiel sind jedoch nur bei der Kandidatensequenz $2 \rightarrow 1 \rightarrow 4 \rightarrow 3$ alle Teilsequenzen der Länge 3 ($1 \rightarrow 4 \rightarrow 3$, $2 \rightarrow 4 \rightarrow 3$, $2 \rightarrow 1 \rightarrow 3$ und $2 \rightarrow 1 \rightarrow 4$) in der Menge der häufigen 3-Sequenzen enthalten. AprioriAll verwirft alle übrigen Kandidaten, da diese nicht häufig sein können.

Für die Supportevaluation organisiert AprioriAll alle Kandidatensequenzen in einem Hashbaum. Die Knoten des Hashbaumes bestehen entweder aus Blättern oder inneren Knoten. Jeder innere Knoten der Tiefe d besitzt eine Hashtabelle, deren Einträge auf weitere Knoten des Baumes in der Tiefe $d + 1$ verweisen. Als Hashfunktion verwendet ein innerer Knoten der Tiefe d den d-ten Warenkorb einer gegebenen Sequenz. Blätter sind die Endpunkte des Hashbaumes und enthalten Kandidatensequenzen. Um das Blatt einer Sequenz in einem Hashbaum zu finden, muss beginnend beim Wurzelknoten ($d = 1$) so lange rekursiv die Hashfunktion über dem d-ten Warenkorb ausgeführt werden, bis der Eintrag in der Hashtabelle auf ein Blatt verweist. Für jede Sequenz der Sequenzdatenbank traversiert AprioriAll den Hashbaum, um die in der Sequenz enthaltenen Kandidatensequenzen zu identifizieren. Findet AprioriAll eine entsprechende Kandidatensequenz, so wird gleichzeitig der Supportzähler dieser Kandidatensequenz inkrementiert. Nachdem alle Sequenzen der Sequenzdatenbank abgearbeitet sind, können anhand der Supportzähler die häufigen Sequenzen bestimmt werden. Diese häufigen Sequenzen bilden den Ausgangspunkt für die nächste Kandidatengenerierung von AprioriAll.

GSP [Srikant u. Agrawal, 1995b, 1996]

Das Akronym GSP steht für *G*eneralized *S*equential *P*atterns. Srikant u. Agrawal verstehen unter der Generalisierung der Sequenzanalyse die Fähigkeit von GSP mit zeitlichen Bedingungen, Zeitfenstern und Taxonomien umgehen zu können.

- (Zeitliche Bedingungen) Die erste Erweiterung von GSP gegenüber AprioriAll betrifft den zeitlichen Abstand zwischen benachbarten Warenkörben einer Eingabesequenz. Hier erlaubt GSP die Angabe von minimalen und maximalen Schwellwerten durch den Benutzer. Nur wenn eine Eingabesequenz diese zeitlichen Bedingungen erfüllt, kann es während der Supportevaluation zum Support einer Sequenz beitragen.

- (Zeitfenster) Die zweite Erweiterung erlaubt es, Warenkörbe einer Eingabesequenz zusammenzufassen, wenn ihr zeitlicher Abstand geringer ist als die Breite eines benutzerdefinierten Zeitfensters Δt. Das folgende Beispiel verdeutlicht diesen Ansatz. Sei $\{A\} \rightarrow \{B\} \rightarrow \{C\}$ eine Eingabesequenz. Während der Supportevaluation kann diese Eingabesequenz zum Support der Sequenz $\{A, B\} \rightarrow \{C\}$ beitragen, wenn der zeitliche Abstand zwischen den Warenkörben $\{A\}$ und $\{B\}$ Δt nicht überschreitet. Srikant u. Agrawal argumentieren, dass Zeitfenster eine praxisorientierte Erweiterung der Hypothesensprache darstellen. In vielen Anwendungen käme es nicht darauf an, ob zwei Items in einem oder verschiedenen Warenkörben auftreten, solange die Zeitdifferenz zwischen den Warenkörben klein genug ist.

- (Taxonomien) Die letzte Erweiterung erlaubt den Umgang mit Taxonomien. Taxonomien sind Hierarchien von *is-a* bzw. *ist-ein* Relationen (vgl. [Srikant u. Agrawal, 1995a]). Ein einfaches Beispiel für eine Taxonomie ist: „Der Herr der Ringe“ *ist-ein* Fantasy-Buch *ist-ein* Buch. GSP ist in der Lage Sequenzen auf unterschiedlichen Ebenen der Taxonomie zu finden.

Wie bereits bei AprioriAll besteht auch bei GSP die algorithmische Lösungsidee zum Auffinden aller häufigen Muster in der Anwendung des Apriori-Ansatzes. Alle drei Erweiterungen von GSP sind durch leichte Modifikationen der Kandidatengenerierung bzw. Supportevaluation zu erreichen.

Im Gegensatz zu AprioriAll verzichtet GSP auf den 3-schrittigen Prozess und findet die häufigen Sequenzen ohne die aufwendige Transformation der Sequenzdatenbank. Dafür führt GSP die Kandidatengenerierung und Supportevaluation auf der Ebene der Items und nicht der Warenkörbe durch. Während der Kandidatengenerierung werden die häufigen k-elementigen Sequenzen verwendet, um $(k+1)$-elementige Kandidatensequenzen zu generieren. GSP sucht dazu alle Paare von Sequenzen S_1 und S_2, so dass nach dem Eliminieren des ersten Items aus dem ersten Warenkorb von S_1 und dem letzten Item aus dem letzten Warenkorb von S_2 identische Sequenzen entstehen. Durch Vereinigung von S_1 und S_2 über ihren identischen Teil entsteht eine $(k+1)$-elementige Kandidatensequenz. Tabelle 3.4 verdeutlicht diesen Ansatz an einem Beispiel. Die erste Spalte zeigt die häufigen 3-elementigen Sequenzen. Hierbei lässt

häufige 3-Sequenzen	Kandidatensequenzen der Länge 4	Kandidaten nach Stutzen
$\{A\} \rightarrow \{C, D\}$	$\{A, B\} \rightarrow \{C, D\}$	$\{A, B\} \rightarrow \{C, D\}$
$\{A, B\} \rightarrow \{C\}$	$\{A, B\} \rightarrow \{C\} \rightarrow \{D\}$	
$\{A, B\} \rightarrow \{D\}$		
$\{B\} \rightarrow \{C, D\}$		
$\{B\} \rightarrow \{C\} \rightarrow \{D\}$		

Tabelle 3.4: Kandidatengenerierung von GSP

sich die Sequenz $\{A, B\} \rightarrow \{C\}$ sowohl mit $\{B\} \rightarrow \{C, D\}$ als auch mit $\{B\} \rightarrow \{C\} \rightarrow \{D\}$ kombinieren (die gemeinsame Teilsequenz nach dem Eliminieren des ersten und letzten Items ist $\{B\} \rightarrow \{C\}$). Es entstehen die beiden Kandidaten $\{A, B\} \rightarrow \{C, D\}$ und $\{A, B\} \rightarrow \{C\} \rightarrow \{D\}$. Wie bei AprioriAll lässt sich die Menge der erzeugten Kandidaten reduzieren, indem geprüft wird, ob alle Teilsequenzen häufig sind. Der Kandidat $\{A, B\} \rightarrow \{C\} \rightarrow \{D\}$ enthält die nicht häufige Teilsequenz $\{A\} \rightarrow \{C\} \rightarrow \{D\}$ und kann daher aus der Menge der Kandidaten entfernt werden (siehe dritte Spalte von Tabelle 3.4).

Die Supportevaluation von GSP verwendet wie bereits bei AprioriAll einen Hashbaum, um alle Kandidaten zusammenzufassen. Der Unterschied zwischen den Hashbäumen besteht lediglich darin, dass GSP die Hashfunktion nicht über den Warenkörben, sondern über den Items der Sequenz anwendet. Des Weiteren nutzt GSP intensiv die Zeitstempel der einzelnen Warenkörbe jeder Eingabesequenz, um zeitliche Bedingungen und Zeitfenster berücksichtigen zu können.

Auf mehreren synthetischen und realen Datensätzen konnten Srikant u. Agrawal zeigen, dass GSP kürzere Laufzeiten zum Auffinden aller häufigen Sequenzen benötigt als AprioriAll. Die

beiden Hauptgründe für die besseren Laufzeiten sind die Generierung kleinerer Kandidatenmengen von GSP (auf Grund der Behandlung der Sequenzen auf dem Level der Items) und die Vermeidung der aufwendigen Transformation der Sequenzdatenbank.

PSP [Masseglia u. a., 1998]

Das Verfahren PSP (*P*refix-tree *S*equential *P*atterns) von Masseglia u. a. ist eine direkte Verbesserung von GSP. Anstelle des Hashbaumes verwendet PSP einen Präfixbaum, um alle Kandidaten während der Supportevaluation in einer Datenstruktur zusammenzufassen. Die Idee des Präfixbaumes ist es, gemeinsame Teilsequenzen von verschiedenen Kandidaten lediglich einmal im Speicher abzulegen. Der Präfixbaum von PSP nutzt dazu gemeinsame Teilsequenzen die am Anfang der Kandidatensequenzen stehen (Präfixe). Als Beispiel dienen die vier Sequenzen $\{A\} \rightarrow \{B\} \rightarrow \{C\}$, $\{A, B, C\}$, $\{B\} \rightarrow \{C\} \rightarrow \{D\}$ und $\{B\} \rightarrow \{C, E\}$. Die Transformation der vier Sequenzen in einen Präfixbaum ist in Abbildung 3.6 zu sehen.

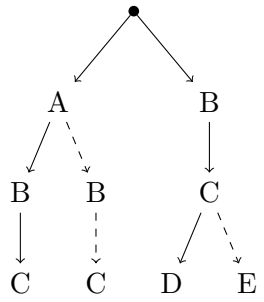


Abbildung 3.6: Beispiel eines Präfixbaumes aus vier Kandidatensequenzen

Jeder Pfad von der Wurzel zu einem Blatt entspricht einer Kandidatensequenz. Ein Knoten der Tiefe d enthält das d-te Item der Sequenz. Im Gegensatz zum Hashbaum steht jedes Blatt des Präfixbaumes für genau einen Kandidaten. Bei den Hashbäumen von AprioriAll und GSP können die Blätter mehrere Kandidaten enthalten. Vor allem aber wird jeder Kandidat vollständig im Blatt abgelegt. Mehrfach auftretende Präfixe werden somit mehrfach gespeichert. Der Präfixbaum von PSP nutzt den Speicherplatz effizienter, da gemeinsame Präfixe nur einmal im Baum abgelegt sind. Im obigen Beispiel ist $\{A\}$ das Präfix der Sequenzen $\{A\} \rightarrow \{B\} \rightarrow \{C\}$ und $\{A, B, C\}$ und $\{B\} \rightarrow \{C\}$ von $\{B\} \rightarrow \{C\} \rightarrow \{D\}$ und $\{B\} \rightarrow \{C, E\}$. Weiterhin ist im Präfixbaum die Information abgelegt, ob zwei Items zum gleichen Warenkorb gehören. In Abbildung 3.6 ist dies mit Hilfe durchgezogener (bzw. gestrichelter) Linien gekennzeichnet.

Masseglia u. a. wiesen in Experimenten mit synthetischen Datensätzen nach, dass durch den Einsatz eines Präfixbaumes, anstelle des Hashbaumes von GSP, sowohl kürzere Laufzeiten als auch eine effizientere Speichernutzung ermöglicht werden.

SPIRIT [Garofalakis u. a., 1999, 2002]

SPIRIT ist eine Familie von vier Algorithmen, die es durch Angabe von regulären Ausdrücken erlauben, die Menge der gesuchten häufigen Muster näher zu spezifizieren. Diese Erweiterung ist insbesondere dann hilfreich, wenn die Menge der häufigen Sequenzen sehr groß ist. Die regulären Ausdrücke werden bereits während der Laufzeit verwendet, um die Ergebnismenge zu

reduzieren. Eine Bedingung in Form eines regulären Ausdrucks ist z.B. $\{A\} \rightarrow (\{B, C\}|\{D\})$. Dieser reguläre Ausdruck verlangt, dass die Ergebnismenge nur aus Sequenzen der Länge 2 besteht. Des Weiteren muss der erste Warenkorb das Item A und der zweite Warenkorb entweder die Items B und C oder das Item D enthalten. Einige Sequenzen, die dem regulären Ausdruck genügen sind: $\{A, B\} \rightarrow \{D, E\}$, $\{A\} \rightarrow \{B, C, E\}$ oder $\{A\} \rightarrow \{D\}$.

Die SPIRIT Algorithmen (*S*equential *P*attern M*I*ning with *R*egular Express*I*on Cons*T*raints) nutzen ebenfalls den Ansatz der Kandidatengenerierung und Supportevaluierung zur Suche nach den häufigen Sequenzen. Die Unterschiede zwischen den einzelnen Algorithmen bestehen in verschiedenen Strategien, wie die regulären Ausdrücke in die Kandidatengenerierung bzw. Supportevaluation integriert werden. Prinzipiell nutzen die SPIRIT Algorithmen die aus der theoretischen Informatik bekannte Äquivalenz zwischen regulären Ausdrücken und endlichen Automaten (siehe z.B. [Wagner, 1994, Seite 178ff.]). Für die Menge der angegebenen regulären Ausdrücke wird ein entsprechender endlicher Automat konstruiert. Dieser Automat dient dazu nicht benötigte Kandidatensequenzen bereits während der Kandidatengenerierung zu entfernen.

SPADE [Zaki, 1998, 2001a,b]

SPADE ist ein weiterer Algorithmus zur Entdeckung häufiger Sequenzen. Im Gegensatz zu den bisher betrachteten Verfahren, verwendet SPADE aber einen anderen algorithmischen Ansatz. Im Abschnitt zu PSP wurde bereits der Präfixbaum für eine gegebene Menge von Kandidatenmustern vorgestellt. SPADE verwendet den Präfixbaum nicht nur als effiziente Datenstruktur, sondern versteht ihn als eine Möglichkeit, den kompletten Hypothesenraum abzubilden. Offensichtlich kann jede beliebige Sequenz im Präfixbaum dargestellt werden. Das Problem, dass der Hypothesenraum nicht beschränkt ist (prinzipiell gibt es unendlich viele verschiedene Sequenzen), spielt keine Rolle. Zum einen hat jeder reale Datensatz eine endliche Größe, d.h. auch die Anzahl der enthaltenen Sequenzen ist begrenzt. Und zum anderen interessieren lediglich die häufigen Sequenzen aus dem Hypothesenraum.

Die zweite Einschränkung führt dazu, dass komplette Zweige des Präfixbaumes nicht betrachtet werden müssen. Besitzt z.B. die Sequenz $\{A\} \rightarrow \{B\}$ einen Support kleiner dem Schwellwert MinSup, so muss jede Sequenz, die diese als Präfix enthält, (z.B. $\{A\} \rightarrow \{B\} \rightarrow \{C\}$) ebenso nicht häufig sein (Apriori-Kriterium). Durch die Verwendung gemeinsamer Präfixe im Präfixbaum sind alle diese Sequenzen Kindknoten von $\{A\} \rightarrow \{B\}$. SPADE sucht alle häufigen Muster, indem er den Hypothesenraum mit Hilfe des Präfixbaumes durchläuft. Stößt SPADE dabei auf ein nicht häufiges Muster, so wird die Suche im entsprechenden Zweig abgebrochen und in einem anderen Zweig fortgeführt. Laut [Zaki, 1998] stammen Sequenzen mit einem gemeinsamen Präfix aus derselben Äquivalenzklasse, die durch das Präfix über die Menge aller Sequenzen induziert wird. Daher erklärt sich auch der Name des Algorithmus: *S*equential *PA*ttern *D*iscovery using *E*quivalence classes.

SPADE erstellt den Präfixbaum während der Suche nach häufigen Sequenzen. Dabei werden ausgehend von einer häufigen Sequenz (d.h. einem Knoten im Präfixbaum) die 1-elementigen Erweiterungen dieser Sequenz (Kindknoten) untersucht. Um den Support eines Kindknotens zu ermitteln, verwendet SPADE die *temporale Verknüpfung* von Ereignislisten. SPADE speichert zu jeder Sequenz eine Liste bestehend aus der Sequenzidentifikation und dem Zeitstempel des letzten Warenkorbs, indem die Sequenz in einer Eingabesequenz beobachtbar war. Abbil-

$\{A\}$	
sid	Zeit
S_1	1
S_2	2
S_3	1
S_4	1
S_6	2

$\cup$

$\{E\}$	
sid	Zeit
S_2	3
S_4	2
S_6	1

$\Longrightarrow$

$\{A\} \rightarrow \{E\}$	
sid	Zeit
S_2	3
S_4	2

Abbildung 3.7: Verknüpfung der Ereignislisten für die Sequenzen $\{A\}$ und $\{E\}$ zur Sequenz $\{A\} \rightarrow \{E\}$ (zur Sequenzdatenbank aus Abbildung 3.5).

dung 3.7 zeigt die Ereignislisten der Sequenzen $\{A\}$ und $\{E\}$ für die Sequenzdatenbank aus Abbildung 3.5. Als Zeitstempel wurde die Nummerierung der Warenkörbe herangezogen. Die Anzahl der verschiedenen Sequenzidentifikationen in der Ereignisliste liefert gerade den Support der Sequenz. Bei der temporalen Verknüpfung vereint SPADE die Ereignislisten zweier k-elementiger Sequenzen, die über ein gemeinsames $(k-1)$-elementiges Präfix verfügen. Das Ergebnis der Verknüpfung ist die Ereignisliste (und damit der Support) einer $(k+1)$-elementigen Sequenz. Die Verknüpfung von Sequenzen über ein gemeinsames Präfix ist nicht eindeutig. Abhängig davon, ob die jeweils letzten Items der Sequenzen einen eigenen Warenkorb bilden oder zusammen mit dem vorletzten Item in einem Warenkorb sind, können bis zu drei Fälle entstehen. Das folgende Beispiel zeigt alle Fälle. Die Sequenzen $\{A\} \rightarrow \{B\}$ und $\{A\} \rightarrow \{C\}$ sollen über ihr gemeinsames Präfix $\{A\}$ verknüpft werden. Die drei Verknüpfungsergebnisse sind $\{A\} \rightarrow \{B\} \rightarrow \{C\}$, $\{A\} \rightarrow \{C\} \rightarrow \{B\}$ und $\{A\} \rightarrow \{B, C\}$. Bei der Vereinigung der Ereignislisten muss entsprechend für jeden der drei Fälle überprüft werden, ob zu identischen Sequenzidentifikationen die Zeitstempel identisch (gleicher Warenkorb) oder unterschiedlich sind. Abbildung 3.7 zeigt den Spezialfall der Vereinigung von $\{A\}$ und $\{E\}$ über das leere Präfix zu $\{A\} \rightarrow \{E\}$. In die Ereignisliste für $\{A\} \rightarrow \{E\}$ können nur die Einträge von $\{A\}$ und $\{E\}$ übernommen werden, bei denen $\{E\}$ bei gleicher Sequenzidentifikation einen größeren Zeitstempel aufweist. Wie Abbildung 3.7 zeigt, enthält die resultierende Ereignisliste von $\{A\} \rightarrow \{E\}$ zwei Einträge.

Ein Ausschnitt des Präfixbaumes für die Sequenzdatenbank aus Abbildung 3.5 und einen minimalen Support von zwei ist in Abbildung 3.8 zu sehen. SPADE ermittelt zunächst die Ereignislisten der 1-elementigen Sequenzen. Dabei erweisen sich die Sequenzen $\{C\}$, $\{D\}$, $\{F\}$ und $\{H\}$ als nicht häufig. Die restlichen Sequenzen können für temporale Verknüpfungen herangezogen werden. Durch die Verknüpfung der Sequenzen $\{A\}$ mit $\{E\}$ (Abbildung 3.7) und $\{A\}$ mit $\{G\}$ entstehen die Sequenzen $\{A\} \rightarrow \{E\}$ und $\{A\} \rightarrow \{G\}$. Beide Sequenzen sind häufig und besitzen das gemeinsame Präfix $\{A\}$. Das Ergebnis ihrer Verknüpfung liefert die häufige 3-elementige Sequenz $\{A\} \rightarrow \{E, G\}$ (siehe Abbildung 3.9). Andere Verknüpfungsergebnisse, wie z.B. $\{A\} \rightarrow \{E\} \rightarrow \{G\}$, erweisen sich als nicht häufig.

SPADE ist in der Lage, den Präfixbaum wahlweise mit einer Tiefensuche oder einer Breitensuche zu durchlaufen. Bei der Breitensuche werden zunächst alle Knoten auf einer Ebene abgearbeitet, bevor die Knoten der nächsten Ebene verarbeitet werden. Im Gegensatz dazu untersucht die Tiefensuche zuerst alle Knoten entlang eines Zweiges, bevor zum nächsten Zweig übergegangen wird. Nach [Zaki, 2001a] ist die Tiefensuche zu bevorzugen, da hier nicht mehr

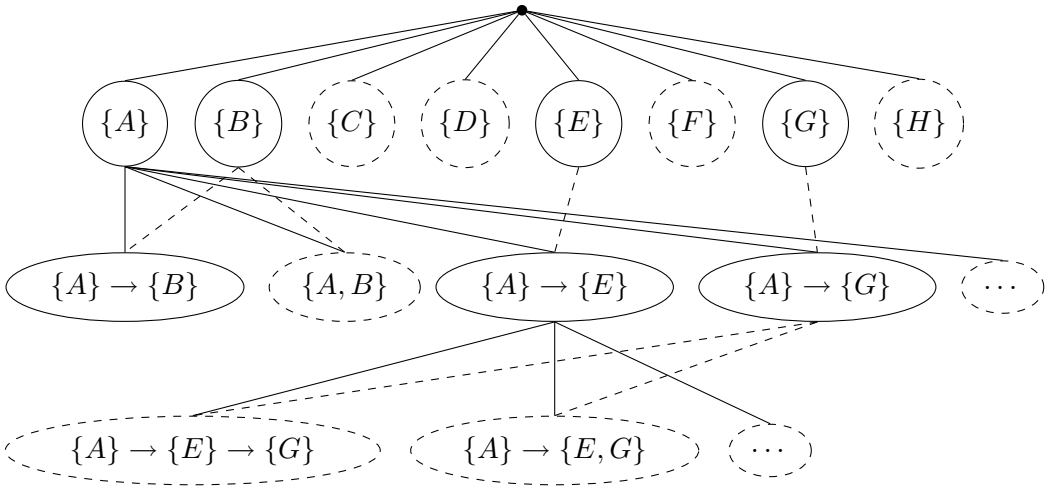


Abbildung 3.8: Ausschnitt des Präfixbaumes von SPADE für die Sequenzdatenbank aus Abbildung 3.5

$\{A\} \rightarrow \{E\}$	
sid	Zeit
S_2	3
S_4	2

$\cup$

$\{A\} \rightarrow \{G\}$	
sid	Zeit
S_2	3
S_4	2

$\Longrightarrow$

$\{A\} \rightarrow \{E, G\}$	
sid	Zeit
S_2	3
S_4	2

Abbildung 3.9: Verknüpfung der Ereignislisten für die Sequenzen $\{A\} \rightarrow \{E\}$ und $\{A\} \rightarrow \{G\}$ zur Sequenz $\{A\} \rightarrow \{E, G\}$ (zur Sequenzdatenbank aus Abbildung 3.5).

benötigte Ereignislisten früher freigegeben werden können und dadurch weniger Arbeitsspeicher benötigt wird. An dieser Stelle sei erwähnt, dass Algorithmen auf Basis des Apriori-Ansatzes eine Breitensuche durchführen. Erst wenn alle häufigen k-elementigen Sequenzen gefunden sind, wird nach den häufigen $(k+1)$-elementigen Sequenzen gesucht.

In [Zaki, 2000] wurde SPADE so erweitert, dass zeitliche Bedingungen für die Warenkörbe einer Sequenz (wie bei GSP) berücksichtigt werden können. In Experimenten mit synthetischen und realen Daten konnte Zaki nachweisen, dass SPADE die Menge der häufigen Sequenzen schneller findet als GSP [Zaki, 2001a].

SPAM [Ayres u. a., 2002]

Der Algorithmus SPAM (*S*equential *PA*ttern *M*ining) greift die Ideen der Tiefensuche und temporalen Verknüpfungen auf und verbindet sie mit einer neuen Datenstruktur für Ereignislisten. SPAM speichert Ereignislisten nicht in Form von Tupeln aus Sequenzidentifikationen und Zeitstempeln, sondern als Bitvektoren. Jedes Bit des Vektors steht für den Warenkorb einer Eingabesequenz. Im einfachsten Fall einer 1-elementigen Sequenz steht eine „1“ im Bitvektor dafür, dass der zugehörige Warenkorb in der Sequenzdatenbank das Item der Sequenz enthält. Abbildung 3.10 zeigt die Bitvektoren einiger 1-elementiger Sequenzen anhand der Sequenzdatenbank aus Abbildung 3.5. Um den Support einer Sequenz zu bestimmen, muss SPAM lediglich

sid	Zeit	$\{A\}$	$\{B\}$	$\{E\}$	$\{G\}$
S_1	1	1	0	0	0
S_1	2	0	1	0	0
S_2	1	0	0	0	0
S_2	2	1	0	0	0
S_2	3	0	0	1	1
S_3	1	1	0	0	1
S_4	1	1	0	0	0
S_4	2	0	0	1	1
S_4	3	0	1	0	0
S_5	1	0	1	0	0
S_6	1	0	0	1	1
S_6	2	1	1	0	0

Abbildung 3.10: Bitvektoren der häufigen 1-elementigen Sequenzen für die Sequenzdatenbank aus Abbildung 3.5 (MinSup ≥ 2)

ermitteln, für wie viele verschiedene Sequenzidentifikationen der zugehörige Bitvektor eine „1“ enthält. Analog zur Tiefensuche von SPADE sucht SPAM alle häufigen Sequenzen, indem eine häufige k-elementige Sequenz zu einer $(k+1)$-elementigen Sequenz erweitert wird. SPAM unterscheidet zwei verschiedene Arten eine gegebene Sequenz zu erweitern: die S-Erweiterung und die I-Erweiterung. Bei der S-Erweiterung (Sequenz-Erweiterung) fügt SPAM der Sequenz einen weiteren Warenkorb hinzu, der aus einem einzelnen Item besteht. Bei der I-Erweiterung (Itemset-Erweiterung) hingegen fügt SPAM dem letzten Warenkorb der Sequenz ein Item hinzu. Der Vorteil der Bitvektoren zur Darstellung der Ereignislisten besteht darin, dass durch das

sid	Zeit	$\{A\}$		$\{A\}_S$		$\{E\}$		$\{A\} \rightarrow \{E\}$
S_1	1	1		0		0		0
S_1	2	0		1		0		0
S_2	1	0		0		0		0
S_2	2	1		0		0		0
S_2	3	0		1		1		1
S_3	1	1	$\xrightarrow{S}$	0	&	0	$\Longrightarrow$	0
S_4	1	1		0		0		0
S_4	2	0		1		1		1
S_4	3	0		1		0		0
S_5	1	0		0		0		0
S_6	1	0		0		1		0
S_6	2	1		0		0		0

Abbildung 3.11: S-Erweiterung von $\{A\}$ zu $\{A\} \rightarrow \{E\}$

logische „Und“ die Erweiterung von Sequenzen um ein einzelnes Item sehr schnell durchgeführt werden kann.

Abbildung 3.11 zeigt die S-Erweiterung der häufigen Sequenz $\{A\}$ zu $\{A\} \rightarrow \{E\}$ am Beispiel der Sequenzdatenbank aus Abbildung 3.5. Die beiden Bitvektoren von $\{A\}$ und $\{E\}$ kennzeichnen die Warenkörbe, welche die Items A und E enthalten. Damit der resultierende Bitvektor, der aus der Verknüpfung beider Bitvektoren durch ein logisches Und entsteht, gerade die Eingabesequenzen markiert, die $\{A\} \rightarrow \{E\}$ enthalten, muss der Bitvektor von $\{A\}$ einer Vorverarbeitung unterzogen werden. Die Vorverarbeitung (in Abbildung 3.11 durch $\xrightarrow{S}$ gekennzeichnet) setzt die Bits aller Warenkörbe, die A enthalten, auf 0. Alle nachfolgenden Warenkörbe der gleichen Sequenz werden auf 1 gesetzt. Die zugrunde liegende Idee besteht darin, dass nach Ausführung des logischen Unds alle Items E markiert sind, denen ein A in derselben Eingabesequenz vorausgeht.

Die I-Erweiterung einer Sequenz benötigt keinen Vorverarbeitungsschritt. Wie Abbildung 3.12 zeigt, kann der Bitvektor der Sequenz direkt mit dem Bitvektor einer 1-Sequenz über ein logisches Und verknüpft werden.

In den Abbildungen 3.11 und 3.12 ist zu sehen, dass die Bitvektoren der S- und I-Erweiterungen gerade die Warenkörbe der Eingabesequenzen mit 1 belegt, die auch SPADE in seinen Ereignislisten speichert (vgl. Abbildungen 3.7 und 3.9). Die S- und I-Erweiterungen lassen sich leicht „hardware-nah“ implementieren. Daher können mit einem einzelnen Rechenschritt des Prozessors (CPU) 32 bzw. 64 Warenkörbe (Bits) auf einmal ausgewertet werden. In [Ayres u. a., 2002] wird die Laufzeit von SPAM mit SPADE und dem nachfolgend vorgestellten PrefixSpan in verschiedenen Experimenten verglichen. Hierbei zeigte sich, dass SPAM in den meisten Experimenten alle häufigen Sequenzen am schnellsten fand. Im Vergleich zu SPADE nennen die Autoren einen Geschwindigkeitsanstieg um Faktor 2.5.

sid	Zeit	$\{A\} \rightarrow \{E\}$		$\{G\}$		$\{A\} \rightarrow \{E, G\}$
S_1	1	0		0		0
S_1	2	0		0		0
S_2	1	0		0		0
S_2	2	0		0		0
S_2	3	1		1		1
S_3	1	0	&	1	$\Longrightarrow$	0
S_4	1	0		0		0
S_4	2	1		1		1
S_4	3	0		0		0
S_5	1	0		0		0
S_6	1	0		1		0
S_6	2	0		0		0

Abbildung 3.12: I-Erweiterung von $\{A\} \rightarrow \{E\}$ zu $\{A\} \rightarrow \{E, G\}$

PrefixSpan [Pei u. a., 2001, 2004]

PrefixSpan (*Prefix*-projected *S*equential *Pa*tter*n* mining) ist ein Algorithmus, der ebenfalls den Hypothesenraum der häufigen Sequenzen auf Basis gemeinsamer Präfixe durchsucht. Im Gegensatz zu SPADE und SPAM verzichtet PrefixSpan auf Ereignislisten und deren Verknüpfung. Er zählt die Häufigkeit von Sequenzen direkt auf der Sequenzdatenbank aus.

Die Vorgehensweise von PrefixSpan soll am Beispiel der Sequenzdatenbank aus Abbildung 3.5 geschildert werden. Zunächst durchläuft PrefixSpan einmal die Sequenzdatenbank und ermittelt dabei alle häufigen 1-elementigen Sequenzen. Für die gegebene Sequenzdatenbank sind dies (MinSup ≥ 2): $\{A\}$ mit Sup($\{A\}$)= 5, $\{B\}$ mit Sup($\{B\}$)= 4, $\{E\}$ mit Sup($\{E\}$)= 3 und $\{G\}$ mit Sup($\{G\}$)= 4. PrefixSpan sucht im nächsten Schritt nach längeren Sequenzen, indem es für jede häufige 1-elementige Sequenz eine neue „projizierte“ Sequenzdatenbank anlegt. Dabei umfasst die zu einem gegebenen Präfix projizierte Sequenzdatenbank nur die Eingabesequenzen, in denen das Präfix vorkommt. Des Weiteren enthalten die projizierten Eingabesequenzen nur die Items und Warenkörbe, die nach dem ersten Vorkommen des Präfixes auftreten. Die projizierte Sequenzdatenbank zur Sequenz $\{A\}$ ist in Abbildung 3.13a dargestellt. Gehört der letzte Warenkorb eines Präfixes noch zum ersten Warenkorb der projizierten Eingabesequenz, so wird dies durch einen Unterstrich (_) kenntlich gemacht (z.B. $\{\_HG\}$ in Abbildung 3.13a).

sid	Eingabesequenz
S_1	$\{B\}$
S_2	$\{EFG\}$
S_3	$\{\_HG\}$
S_4	$\{EG\} \rightarrow \{B\}$
S_6	$\{\_B\}$

(a)

sid	Eingabesequenz
S_2	$\{\_FG\}$
S_4	$\{\_G\} \rightarrow \{B\}$

(b)

Abbildung 3.13: projizierte Sequenzdatenbanken (a) über $\{A\}$ und (b) über $\{A\} \rightarrow \{E\}$

Die projizierten Sequenzdatenbanken sind der Ausgangspunkt eines rekursiven Prozesses. PrefixSpan ermittelt alle Items in der projizierten Sequenzdatenbank, so dass die Verknüpfung aus Präfix und Item häufig ist. Es gibt zwei mögliche Verknüpfungen. Entweder wird das Item dem letzten Warenkorb des Präfixes hinzugefügt, oder dem Präfix wird ein neuer Warenkorb angefügt, der nur das Item enthält. Die so gefundenen häufigen Sequenzen werden von PrefixSpan als neue Präfixe benutzt, um weitere projizierte Sequenzdatenbanken ableiten zu können. In der über $\{A\}$ projizierten Sequenzdatenbank aus Abbildung 3.13 ermittelt PrefixSpan die häufigen Items B, E und G. In Kombination mit dem Präfix $\{A\}$ sind somit die häufigen Sequenzen $\{A\} \rightarrow \{B\}$, $\{A\} \rightarrow \{E\}$ und $\{A\} \rightarrow \{G\}$ gefunden worden. PrefixSpan führt die Suche fort, indem es die gerade gefundenen Sequenzen als Präfixe zur erneuten Projektion der Sequenzdatenbank verwendet. Die über $\{A\} \rightarrow \{B\}$ projizierte Sequenzdatenbank ist leer. Dadurch ist die Suche nach häufigen Sequenzen in diesem Zweig beendet. Die $\{A\} \rightarrow \{E\}$ projizierte Sequenzdatenbank ist in Abbildung 3.13b dargestellt. Hier erweist sich das Item G für die Sequenz $\{A\} \rightarrow \{E, G\}$ als häufig. PrefixSpan führt die Suche nach diesem Schema fort, bis alle häufigen Sequenzen gefunden sind.

Pei u. a. [2001] konnten auf verschiedenen Datensätzen eine deutliche Steigerung der Laufzeiten von PrefixSpan gegenüber GSP nachweisen.

Weitere Verfahren und innere Systematik

Die oben beschriebenen Algorithmen zur Suche nach häufigen Sequenzen stellen nur einen Ausschnitt der zur Verfügung stehenden Verfahren dar. Es gibt eine Vielzahl von Veröffentlichungen, die weitere algorithmische Lösungsansätze, verbesserte Datenstrukturen oder weiterführende Verarbeitungsschritte vorschlagen. Zumeist orientieren sich diese Veröffentlichungen an einem der oben beschriebenen Algorithmen oder an etablierten Ideen aus der häufigen Warenkorbanalyse.

FreeSpan [Han u. a., 2000a] ist ein Vorgänger von PrefixSpan, der eine Tiefensuche mit Hilfe von projizierten Sequenzdatenbanken durchführt. Die Aufteilung des Hypothesenraumes orientierte sich nicht an den Präfixen und erwies sich im Vergleich zu PrefixSpan als weniger effizient [Pei u. a., 2001]. TSET [Guil u. a., 2005] speichert alle Sequenzen in einem Präfixbaum und besitzt einen Parameter für die maximale zeitliche Länge einer Sequenz. Lin u. Lee erweitern PrefixSpan um eine effiziente Indexstruktur für die Sequenzdatenbank. Diese Indexstruktur erleichtert die Projektion von Sequenzdatenbanken für ein gegebenes Präfix und führt zu besseren Laufzeitergebnissen [Lin u. Lee, 2005]. Ebenfalls eine Indexstruktur verwendet IBM [Savary u. Zeitouni, 2005], jedoch dürfen die Warenkörbe nur ein einziges Item enthalten. Der Algorithmus M^2SP aus [Plantevit u. a., 2005] sucht nach häufigen Sequenzen, die auch Platzhalter (Wildcards, z.B. $(A = a_1, B = *) \rightarrow (C = c_1)$) enthalten können. In [Li u. a., 2000] wird nach sich in den Daten wiederholenden Sequenzen gesucht. Die Zeitabstände zwischen den Wiederholungen können anhand eines Kalenderschemas in unterschiedlicher Granularität beschrieben werden. Des Weiteren gibt es Bestrebungen, Sequenzen in Datenströmen (Datastreams) zu finden (z.B. [Chen u. a., 2005]) oder mit räumlichen Daten zu verknüpfen (z.B. [Cao u. a., 2005]).

Aus der Warenkorbanalyse stammt die Idee der geschlossenen Sequenzen. Analog zu geschlossenen Warenkörben (siehe z.B. [Zaki u. Hsiao, 2002; Wang u. a., 2003]) beschreiben geschlossene Sequenzen (closed sequences) die Menge aller häufigen Sequenzen zu einer Sequenzdatenbank,

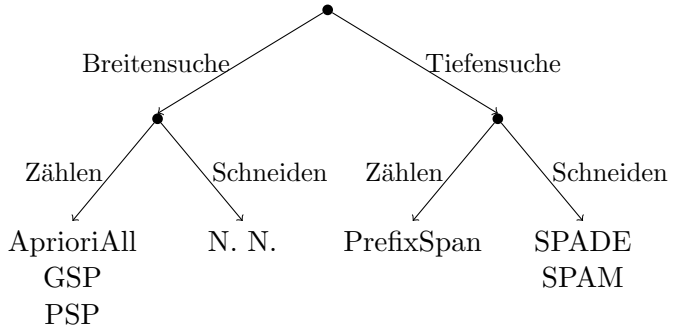


Abbildung 3.14: Innere Systematik der Verfahren zur Sequenzanalyse (vgl. [Hipp u. a., 2000])

die nicht Teilsequenz einer Sequenz mit gleichem Support sind. Der Vorteil geschlossener Sequenzen besteht darin, dass ihre Anzahl in der Regel deutlich geringer ist als die Anzahl aller häufigen Sequenzen. Die in [Yan u. a., 2003; Wang u. Han, 2004] und [Tzvetkov u. a., 2003, 2005] präsentierten Verfahren (CloSpan, BIDE und TSP) finden die Menge der geschlossenen Sequenzen direkt. Das heißt, es wird kein Zwischenschritt durchgeführt, in dem die geschlossenen Sequenzen aus der Menge der häufigen Sequenzen herausgefiltert werden. Die Verfahren nutzen dabei die Einschränkungen geschlossener Sequenzen, um den zu durchsuchenden Hypothesenraum weiter einzuschränken. In Folge dessen können auch die Laufzeiten von Verfahren der Sequenzanalyse unterboten werden.

Unterliegt eine Sequenzdatenbank einem kontinuierlichen Veränderungsprozess, d.h. der Datenbank werden neue Eingabesequenzen oder einigen Sequenzen neue Warenkörbe hinzugefügt, so können diese Veränderungen Auswirkungen auf die Menge der häufigen Sequenzen haben. Die Algorithmen ISM und MFS ([Parthasarathy u. a., 1999] und [Kao u. a., 2005]) sind in der Lage, anhand der Ergebnisse einer „alten“ Sequenzanalyse und der Datenbankänderungen, die neue Menge der häufigen Sequenzen zu berechnen, ohne eine Sequenzanalyse auf der aktuellen (kompletten) Sequenzdatenbank durchführen zu müssen.

Agrawal u. Srikant haben die Sequenzanalyse als eine Erweiterung der Warenkorbanalyse eingeführt [Agrawal u. Srikant, 1995]. Aber auch bei den verwendeten Algorithmen für beide Probleme lassen sich eindeutige Parallelen aufzeigen. Bei allen in diesem Abschnitt vorgestellten Verfahren für die Sequenzanalyse entsprang die algorithmische Lösungsstrategie zum Auffinden der häufigen Sequenzen einem Algorithmus zur Warenkorbanalyse. So ist z.B. GSP die konsequente Übertragung des Apriori-Ansatzes [Agrawal u. Srikant, 1994a]. Ebenfalls kennt die Tiefensuche mit Hilfe eines Präfixbaumes von SPADE und SPAM ein Vorbild aus der Warenkorbanalyse: Eclat [Zaki u. a., 1997]. Zuletzt setzt PrefixSpan den Ansatz von FP-Growth [Han u. a., 2000b] für die Sequenzanalyse um.

In [Hipp u. a., 2000] und [Hipp, 2003, Seite 65ff.] wurde eine Systematisierung der Verfahren zur Warenkorbanalyse auf Basis ihres algorithmischen Ansatzes vorgestellt. Aufgrund der beschriebenen Vorbildfunktion der Verfahren zur Warenkorbanalyse, lässt sich diese Systematisierung direkt auf die Verfahren zur Sequenzanalyse übertragen. Hipp u. a. berücksichtigen bei ihrer Systematisierung zwei Bereiche.

1. Die Bestimmung des Supports erfolgt entweder direkt durch das Zählen von Vorkommen oder alternativ indirekt durch das Schneiden (Verknüpfen) von Ereignislisten.

2. Als Suchstrategie findet entweder eine Tiefensuche oder eine Breitensuche Verwendung.

Abbildung 3.14 zeigt die Einordnung der vorgestellten Verfahren zur Sequenzanalyse, anhand der sich ergebenden zwei Hierarchiestufen.

Anwendungen

Die nachfolgende Liste von Anwendungen der Sequenzanalyse soll diesen Abschnitt abschließen. Es ist dabei jedoch festzuhalten, dass trotz der vielen Veröffentlichungen zur Entdeckung häufiger Sequenzen in der Literatur nur wenige Anwendungsprobleme detailliert geschildert werden.[1]

- Kundendaten waren die Motivation zur erstmaligen Formulierung des Problems der Sequenzanalyse in [Agrawal u. Srikant, 1995]. Durch das Aufkommen von Kundenkarten in Supermärkten konnten erstmals die zu verschiedenen Zeiten eingekauften Waren eindeutig einem Kunden zugeordnet werden. Agrawal u. Srikant interessierten sich dafür, ob es Abfolgen von Einkäufen gab, die von vielen verschiedenen Kunden getätigt wurden. Die so identifizierten Sequenzen sollten einen Supermarktbesitzer in die Lage versetzen sein Warenangebot zu verbessern.

- Garofalakis u. a. [1999, 2002] analysieren mit ihrem Algorithmus SPIRIT die Zugriffsprotokolle eines WWW-Servers. Die Protokolle enthalten Informationen darüber, wann ein Nutzer Seiten aus dem Angebot des Servers abgerufen hat. Die häufigen Sequenzen zeigen daher die Reihenfolgen von Seitenabrufen (Klick-Pfade), die von vielen Nutzern verwendet werden. Garofalakis u. a. wollen die häufigen Sequenzen dazu nutzen, das Angebot des WWW-Servers nutzungsfreundlicher zu gestalten. Das Weblog-Analyseprogramm FS-Miner in [El-Sayed u. a., 2004] verfolgt das gleiche Ziel.

- In der von Zaki u. a. beschriebenen Anwendung ist es notwendig, einen erfolgreichen Plan zur Evakuierung einer kleinen Insel zu generieren. Das von ihnen vorgeschlagene System PlanMine [Zaki u. a., 1998, 2000] dient der Analyse von Plänen auf Basis von Simulationsprotokollen. Dabei erstellt ein Anwender einen ersten Plan zur Evakuierung. Dieser Plan wird mehrfach mit Hilfe eines probabilistischen Modells simuliert. Aus den Simulationsprotokollen ist ersichtlich wie oft der Plan erfolgreich war oder an welchen Stellen er scheiterte. Die Simulationsprotokolle sind Ausgangspunkt für den Algorithmus SPADE, der alle häufigen Sequenzen extrahiert. PlanMine enthält zusätzlich einige Filterschritte, in denen wenig aussagekräftige Sequenzen (in Bezug auf die Domäne) entfernt werden. Die resultierenden Sequenzen geben Auskunft darüber, warum ein Plan scheitert. PlanMine kann auf zwei Arten eingesetzt werden: Zum einen generiert es Hinweise, die zur Verbesserung eines gegebenen Plans dienen, und zum anderen dient es der Überwachung der Ausführung eines Planes. Hier werden Fehler, die zum Scheitern eines Plans führen können, frühzeitig angezeigt.

- In [Ferreira u. Azevedo, 2005] ist die Sequenzanalyse Teil eines Klassifkationssystems, das einer gegebenen Proteinsequenz eine Proteinfamilie zuordnet. Als Datenbasis dienen Proteinsequenzen mit bekannter Proteinfamilie. Die Extraktion häufiger Sequenzen

[1] Dieses Missverhältnis wurde auch in [Antunes u. Oliveira, 2001] dargestellt.

aus dieser Datenbank ist hilfreich, da Proteinsequenzen der gleichen Familie häufig über gemeinsame Teilsequenzen verfügen. Häufige Sequenzen, die in einer Proteinsequenz vorkommen, geben somit einen Hinweis auf die zugehörige Proteinfamilie. Zur Klassifikation verknüpfen Ferreira u. Azevedo die Informationen aus den häufigen Sequenzen noch mit den Ergebnissen eines Naïve-Bayes Klassifikators.

Eine weitere Anwendung der Sequenzanalyse auf Biosequenzen wird in [Brazma u. a., 1998a] und [Brazma u. a., 1998b] beschrieben.

- In [Eichinger u. a., 2006] werden abwanderungswillige Kunden eines Telekommunikationsanbieters identifiziert. Ausgangspunkt ist eine Kundendatenbank, die Zeitpunkte von Beschwerden, Reparaturen, Vertragsabschlüssen und Kündigungen enthält. Auf Basis der häufigen Sequenzen in den Daten werden Regeln abgeleitet, die bewerten wie abwanderungswillig ein Kunde ist. Eichinger u. a. nutzen die Sequenzen jedoch nicht allein zur Klassifikation der Kunden, sondern kombinieren sie darüber hinaus mit den Ergebnissen eines Entscheidungsbaums.

Weitere Anwendungen sind z.B. im Umfeld des Web-Usage Minings [Spiliopoulou u. a., 1999], der Klassifikation von DNA Sequenzen [Han u. a., 2001] oder der Merkmalsextraktion [Lesh u. a., 2000] angesiedelt.

3.2 Häufige Muster in intervallbasierten Daten

Im Folgenden werden Verfahren für intervallbasierte Daten beschrieben. Eine intervallbasierte Datenbasis enthält nicht nur punktuelle Ereignisse, sondern umfasst auch Vorgänge, die sich über einen längeren Zeitraum erstrecken können. Im Gegensatz zu punktuellen Ereignissen können zwei Intervalle in mehreren verschiedenen Beziehungen zueinander stehen. So können sich zwei Intervalle z.B. überlappen oder eines kann das andere enthalten. Von besonderer Bedeutung sind die Intervallrelationen nach Allen [1983], da die meisten Verfahren sie (oder eine Teilmenge davon) zur Repräsentation eines intervallbasierten Musters verwenden. In Abbildung 3.15 sind die 13 Intervallrelationen nach Allen (kurz $\mathbb{I}$) dargestellt.

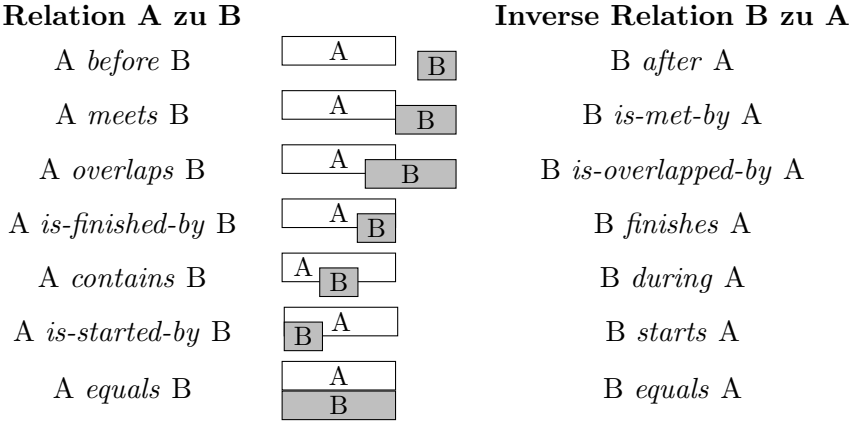


Abbildung 3.15: Allens Intervallrelationen $\mathbb{I}$ [Allen, 1983]

3.2.1 Eine Eingabesequenz

Zunächst stehen wiederum Verfahren im Fokus, deren Ausgangsdaten aus einer einzigen Eingabesequenz bestehen. Abbildung 3.16 gibt ein Beispiel für eine derartige Datenbasis. Ein Ereignis einer intervallbasierten Datenbasis wird auch als *Label* bezeichnet.

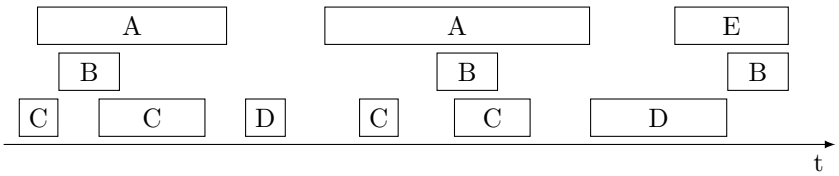


Abbildung 3.16: Beispiel für einen intervallbasierten Datensatz mit einer Eingabesequenz

Containment Graph [Villafane u. a., 1999, 2000]

Villafane u. a. untersuchen ausschließlich Muster, die auf der Intervallrelation *contains* basieren. Im einfachsten Fall besteht ein Muster daher aus zwei Labels z.B. *A contains B*. Längere Muster nutzen die transitive Eigenschaft der *contains* Relation und haben die Form: *A contains B* $\wedge$ *B contains C* $\wedge \ldots$. Für jedes Muster ist der Support durch die Anzahl der Vorkommen in den Daten gegeben. Wie Villafane u. a. feststellen, verletzt diese Supportdefinition das Apriori-Kriterium.

Ein Beispiel für die Verletzung des Apriori-Kriteriums ist in Abbildung 3.16 zu finden. Das Muster *A contains C* kommt dreimal in der Eingabesequenz vor. Es gibt jedoch nur zwei Intervalle mit dem Label *A*. Nach dem Apriori-Kriterium muss aber jedes Teilmuster mindestens den Support des gesamten Musters aufweisen.

Um dennoch alle häufigen Muster in einem Datensatz ermitteln zu können, verwenden Villafane u. a. einen „Containment Graph“. Der Containment Graph ist ein azyklisch gerichteter Graph, der direkt aus der Eingabesequenz erstellt wird. Für jedes Intervall der Eingabesequenz enthält der Graph einen Knoten. Prinzipiell wird zwischen zwei Knoten des Graphen eine Kante eingefügt, wenn die entsprechenden Intervalle zueinander in der Relation *contains* stehen. Nur Kanten, die sich aus der Transitivität der Relation *contains* aus anderen Kanten erschließen lassen, werden nicht in den Graphen eingefügt. Gilt z.B. *A contains B* und *B contains C*, so wird für *A contains C* keine Kante eingefügt. Für das Beispiel aus Abbildung 3.16 ergibt sich der Containment Graph aus Abbildung 3.17 (enthält nur Knoten mit Kanten).

Aus dem Containment Graphen lassen sich alle Muster in Bezug auf die Intervallrelation *contains* ablesen. Villafane u. a. traversieren den Graphen entlang der gerichteten Kanten, um die Menge der häufigen Muster zu bestimmen.

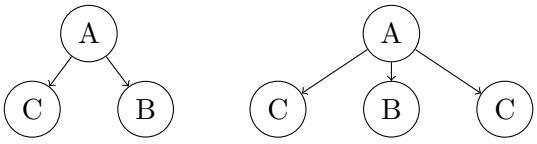


Abbildung 3.17: Containment Graph für das Beispiel aus Abbildung 3.16

Muster und temporaler Support [Höppner, 2001, 2002a; Höppner u. Klawonn, 2001, 2002]

Höppner stellte erstmals ein Verfahren zur Suche nach häufigen Mustern vor, das auf dem vollen Umfang von Allens Intervallrelationen basiert. Jedes Muster besteht aus der Liste aller Intervallrelationen, die zwischen den Labels des Musters existieren. Die Intervallrelationen werden in einer Tabelle organisiert. Abbildung 3.18 zeigt ein Muster mit den beteiligten Labels A, B und C. Für die Relationen zwischen den einzelnen Labels gilt: A *meets* B, A *before* C und B *overlaps* C.

	A	B	C
A	*equals*	*meets*	*before*
B	*is-met-by*	*equals*	*overlaps*
C	*after*	*is-overlapped-by*	*equals*

Abbildung 3.18: Muster mit drei Labels

Den Support eines Musters bestimmt Höppner nicht durch Zählen der Vorkommen, sondern mit Hilfe eines gleitenden Zeitfensters. Hierbei wird ein Zeitfenster mit einer definierten Breite von links nach rechts über die Eingabesequenz „geschoben“. Aus dem Verhältnis der Zeitdauer, wie lange ein Muster in dem Zeitfenster sichtbar war, zur gesamten Zeitdauer der Eingabesequenz ergibt sich der *temporale Support* des Musters. Der Unterschied zur *Frequency* aus [Mannila u. a., 1997] besteht darin, dass die Zeitachse der Eingabesequenz kontinuierlich sein kann. Dennoch kann der temporale Support angegeben werden durch:

$$\text{Temporaler Support}(P) = \frac{\text{Anzahl der Zeitfenster mit P}}{\text{Anzahl aller Zeitfenster}}.$$

Der temporale Support eines Musters P lässt sich als die Wahrscheinlichkeit interpretieren, ein Vorkommen von P im Zeitfenster finden zu können, wenn das Zeitfenster zufällig auf der Eingabesequenz platziert wird.

Höppners Algorithmus zum Auffinden aller häufigen Muster basiert auf dem Apriori-Ansatz. Das heißt, es wird zunächst in der Phase der Kandidatengenerierung die Menge der potentiell häufigen Muster mit k Labels bestimmt. Der temporale Support jedes Kandidatenmusters wird durch das gleitende Zeitfenster auf der Eingabesequenz berechnet (Supportevaluation). Anschließend bilden die häufigen Kandidaten die Grundlage der Kandidatengenerierung für Muster mit $k + 1$ Labels.

TSKR [Mörchen, 2006a,b,c]

Die TSKR (*T*ime *S*eries *K*nowledge *R*epresentation) ist eine hierarchisch organisierte Hypothesensprache zur Repräsentation von zeitlichen Mustern. Sie ist einer Erweiterung der UTG (*U*nification-based *T*emporal *G*rammar vgl. [Guimarães u. Ultsch, 1999] und [Ultsch, 2004]) für intervallbasierte Daten. Auf der untersten Hierarchiestufe stehen die Intervalle, welche innerhalb der TSKR als *Tones* bezeichnet werden. Basierend auf den Tones werden die *Chords* abgeleitet. Ein Chord spezifiziert ein Zeitintervall, in dem mehrere Tones zusammenfallen. Ein Chord beschreibt somit, dass etwas gleichzeitig geschieht. Auf der letzten Hierarchiestufe

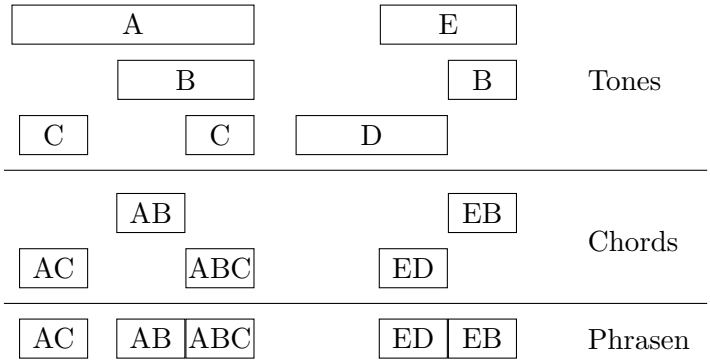


Abbildung 3.19: Ein Muster auf den verschiedenen Hierarchiestufen der TSKR

stehen die *Phrasen*. Sie bestehen aus einer geordneten Menge von Chords und drücken die zeitliche Abfolge von Ereignissen aus. Abbildung 3.19 zeigt ein Beispiel für die Muster auf den verschiedenen Ebenen der Hierarchie.

Die Suche nach häufigen Mustern in der TSKR erfolgt auf Grund der hierarchischen Struktur in zwei Schritten. Im ersten Schritt werden die häufigen Chords gesucht. Ein Chord gilt als häufig, wenn die Summe der Zeitdauern (Länge) aller Intervalle, in denen der Chord auftritt, einen gegebenen Schwellwert überschreitet. Dabei muss jedes berücksichtigte Intervall eine vorgegebene Mindestdauer aufweisen. Mörchen zeigt, dass häufige Chords mit Hilfe von Algorithmen der Warenkorbanalyse gefunden werden können und gibt eine Adaption des Algorithmus CHARM [Zaki u. Hsiao, 2002, 2005] zu diesem Zweck an [Mörchen, 2006c, Seite 80ff.].

Der zweite Schritt ermittelt die häufigen Phrasen aus den zuvor gefundenen häufigen Chords. Dazu wird zunächst die Menge der Chords in eine Sequenz von Warenkörben konvertiert. Für die Chords aus Abbildung 3.19 liefert diese Konvertierung die Sequenz $\{A, C\} \rightarrow \{A, B\} \rightarrow \{A, B, C\} \rightarrow \{E, D\} \rightarrow \{E, B\}$. Mörchen partitioniert die einzelne Sequenz in eine Menge kürzerer Sequenzen mit Hilfe von Anwendungswissen über die Daten oder einem Zeitfensterverfahren [Mörchen, 2006c, Seite 90]. Um alle Phrasen zu ermitteln, die einem gegebenen minimalen Support genügen, wird ein Verfahren der Sequenzanalyse auf die Menge der kurzen Sequenzen angewendet.

In einem abschließenden Schritt fasst Mörchen Phrasen in Form eines Graphen zusammen, die gemeinsame Chords enthalten. Die Algorithmen zur Erstellung der Graphen stammen aus [Casas-Garriga, 2005]. Abbildung 3.20 zeigt ein einfaches Beispiel für mehrere Phrasen in einem Graphen.

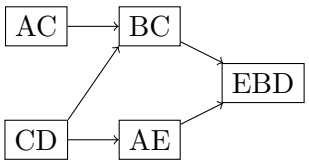


Abbildung 3.20: Mehrere Phrasen sind in einem Graphen zusammengefasst

Anwendungen

Die vorgestellten Verfahren zur Extraktion häufiger Muster aus intervallbasierten Daten mit einer Eingabesequenz wurden bereits für verschiedene Anwendungen eingesetzt. Die folgende Liste gibt eine Übersicht:

- Villafane u. a. untersuchen mit dem Containment Graph die Leistungscharakteristik eines Datenbanksystems. Aus den Protokolldateien der Datenbank werden z.B. Ereignisse extrahiert, die das Sperrverhalten der Ressourcen (Tabellen und Datensätze) beschreiben [Villafane u. a., 1999].

- In [Höppner u. Klawonn, 2002] und [Höppner, 2002a, Seite 131ff.] wird die Ableitung von Regeln mit Hilfe häufiger Muster aus Wetterdaten beschrieben. Grundlage der Analyse sind mehrjährige Messungen einer Wetterstation zum Luftdruck, zur Windgeschwindigkeit und zur Windrichtung. Höppner zeigt, dass ein Teil der gefundenen Regeln bekanntes Wissen von Experten modelliert. Darüber hinaus zeigten die Regeln neue Zusammenhänge zwischen den Eingabegrößen auf.

- Mörchen u. a. präsentieren in [Mörchen u. a., 2004] und [Mörchen u. Ultsch, 2007] eine sportmedizinische Anwendung. Dazu wurde der Bewegungsablauf eines Inlineskaters unter Laborbedingungen vermessen. Die gemessene Zeitreihe enthält sowohl das Aktivierungsniveau verschiedener Muskeln als auch die Beugungswinkel von Fuß-, Hüft- und Kniegelenken [Mörchen, 2006c, Seite 129]. Die mit Hilfe der TSKR gefundenen Muster beschreiben das Zusammenspiel der verschiedenen Muskelgruppen, während eines Bewegungszyklus des Inlineskaters.

- In [Papapetrou u. a., 2006a,b] werden Bereiche mit einer Häufung bestimmter Nukleinbasen in einer DNA-Sequenz identifiziert und ihre Intervallrelationen zueinander untersucht.[2] Papapetrou u. a. verwenden ein gleitendes Zeitfenster (analog zu [Höppner u. Klawonn, 2002]), um die häufigen Muster (bestehend aus den gefundenen Bereichen) zu bestimmen.

3.2.2 Mehrere Eingabesequenzen

Im vorherigen Abschnitt bestand ein Datensatz aus einer Eingabesequenz. Nachfolgend werden Verfahren vorgestellt, die einen anderen Typ von Ausgangsdaten adressieren. Wie Abbildung 3.21 veranschaulicht, enthält ein Datensatz in diesem Abschnitt die intervallbasierten Ereignisse zu beliebig vielen Eingabesequenzen.

A1 und A2 [Kam u. Fu, 2000]

Kam u. Fu definierten die Hypothesensprachen A1 und A2, um temporale Muster zu beschreiben. Wie bereits bei dem Verfahren von Höppner (vgl. Abschnitt 3.2.1) bilden Allens

[2] Der gleiche Ansatz findet sich auch in [Mooney u. Roddick, 2004]. Zunächst werden häufige ereignisbasierte Muster identifiziert, um anschließend die Intervallrelationen zwischen den Instanzen der Muster zu überprüfen.

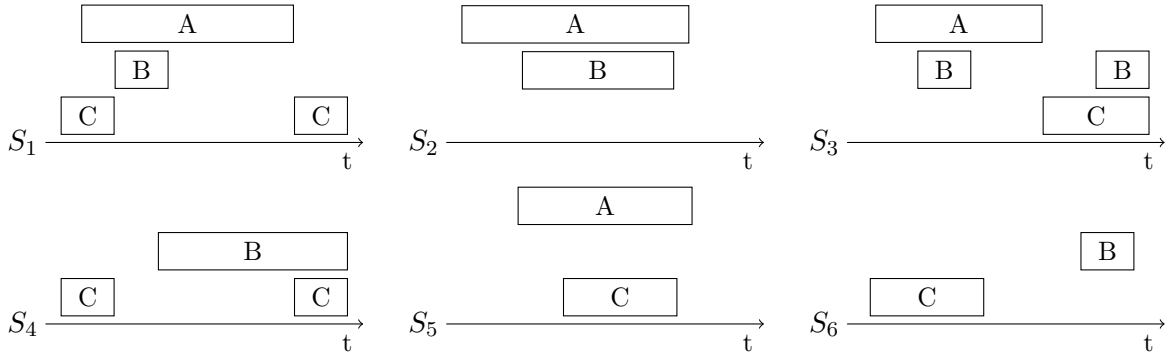


Abbildung 3.21: Ein Beispieldatensatz bestehend aus sechs Eingabesequenzen $S_1, \ldots, S_6$. Jede Eingabesequenz wird durch eine Reihe von Intervallen näher beschrieben.

Intervallrelationen die Grundlage für beide Hypothesensprachen. Im Gegensatz zu Höppner listen A1 und A2 nur einige Relationen zwischen den beteiligten Intervallen auf.

Die Muster in A1 sind wie folgt rekursiv definiert:

1. Sei l ein Label des Datensatzes, so ist l ein A1-Muster.
2. Sei A ein A1-Muster und l ein Label des Datensatzes, so ist auch $(A \text{ rel } l)$ mit rel $\in \mathbb{I}$ ein A1-Muster.

Aus der Definition der A1-Muster folgt, das jedes A1-Muster die folgende Form haben muss: $((\ldots(l_1 \text{ rel}_1 l_2) \text{ rel}_2 l_3) \ldots \text{ rel}_{k-1} l_k)$.

Analog zu A1 sind die Muster in A2 definiert.

1. Seien l_1 und l_2 zwei Labels des Datensatzes, so ist $(l_1 \text{ rel } l_2)$ mit rel $\in \mathbb{I}$ ein A2-Muster.
2. Seien A und B A2-Muster, so ist auch $(A \text{ rel } B)$ mit rel $\in \mathbb{I}$ ein A2-Muster.

Ein Beispiel für ein A2-Muster ist: *((A overlaps B) before (C meets D))*.

Zum Auffinden der häufigen Muster in einem gegebenen Datensatz verwenden Kam u. Fu für beide Hypothesensprachen den Apriori-Ansatz. Die Kandidatengenerierung für A1-Muster erfolgt durch Verbindung eines häufigen Musters mit $k-1$ Labels und einem häufigen Label des Datensatzes. Als Verknüpfung werden alle Intervallrelationen nach Allen benutzt. In gleicher Weise findet auch die Kandidatengenerierung für A2-Muster statt. Allerdings wird hier ein häufiges Muster der Größe $2k-2$ mit allen häufigen A2-Mustern der Größe 2 verknüpft.[3] Die Supportevaluation ist für beide Hypothesensprachen identisch und wird durch Auszählen der Kandidaten auf der Datenbasis realisiert. Der Supportzähler eines Kandidaten wird für jede Eingabesequenz, die ein Vorkommen des Musters enthält, inkrementiert. Somit gibt der Support eines Musters die Anzahl der Eingabesequenzen an, die das Muster enthalten. Damit adaptieren Kam u. Fu die Supportdefinition der Sequenzanalyse für intervallbasierte Datensätze.

[3] Die Definition der A2-Muster erzwingt, dass nur Muster mit einer geraden Anzahl von beteiligten Intervallen darstellbar ist.

Arrangement Enumeration Tree [Papapetrou u. a., 2005]

Der „*A*rrangement *E*numeration *T*ree“ (AET) ist die zentrale Datenstruktur der Algorithmen von Papapetrou u. a.. Ähnlich den Präfixbäumen bei der Sequenzanalyse organisiert der AET die enthaltenen Muster hierarchisch von den kurzen zu den langen Mustern. Wie bereits bei [Höppner, 2002a] ist ein Muster durch eine Menge von Labels und ihrer paarweisen Intervallrelationen nach Allen spezifiziert. Abbildung 3.22 gibt ein Beispiel für einen AET. Innere Knoten

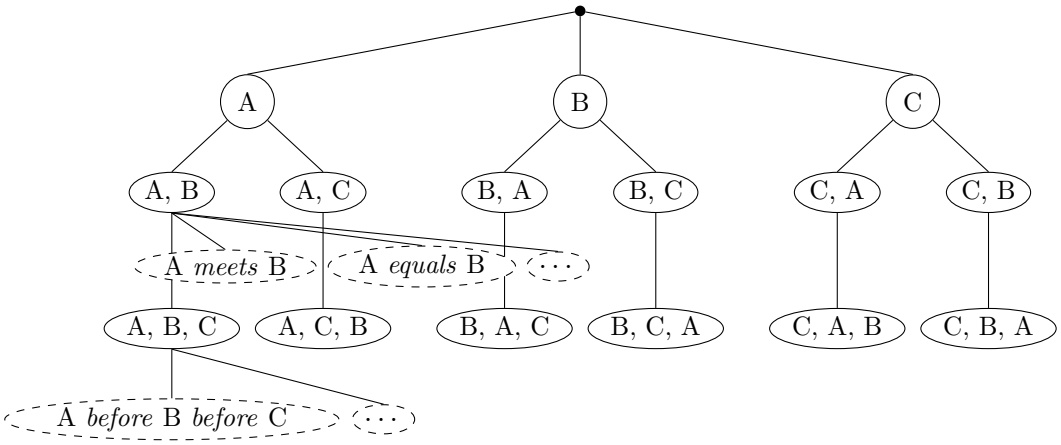


Abbildung 3.22: Beispiel für einen Arrangement Enumeration Tree

im AET enthalten immer die an einem Muster beteiligten Labels. Die Kindknoten eines inneren Knoten können entweder innere Knoten, die dem Elternknoten ein weiteres Label hinzufügen, oder Blätter sein. Jedes Blatt spezifiziert genau ein Muster anhand der Intervallrelationen zwischen den Labels des Elternknoten.

Der AET stellt eine hierarchische Ordnung über den Mustern der Hypothesensprache her. Papapetrou u. a. untersuchen drei Strategien, um den AET bei der Suche nach häufigen Mustern zu durchlaufen: Breitensuche, Tiefensuche und hybride Tiefensuche. Die hybride Tiefensuche realisiert eine Tiefensuche erst ab den Knoten der Tiefe zwei. Alle häufigen Muster mit zwei oder weniger Labels werden durch eine Breitensuche bestimmt. Unabhängig von der gewählten Suchstrategie erfolgt die Supportevaluation eines Musters. Zu jedem Blatt (Muster) wird eine ISId-Liste (*I*nterval *S*equence *Id*entification) gespeichert. Die ISId-Liste gibt an, in welcher Eingabesequenz das Muster vorkommt und welche Intervalle daran beteiligt sind. Für ein neues Muster bestimmen Papapetrou u. a. die ISId-Liste anhand eines Abgleichs der ISId-Listen der enthaltenen Muster der Größe 2 (zwei beteiligte Labels). Für ein Muster mit drei Labels (z.B. *A meets B*, *A before C*, *B before C*) müssen daher drei ISId-Listen miteinander verglichen werden (die ISId-Listen der Muster *A meets B*, *A before C* und *B before C*). Die Anzahl der verschiedenen Eingabesequenzen in der ISId-Liste ergeben den Support des Musters.

Papapetrou u. a. zeigen in verschiedenen Experimenten, dass die hybride Tiefensuche das beste Laufzeitverhalten besitzt.

ARMADA [Winarko u. Roddick, 2005, 2007]

Winarko u. Roddick übernehmen die Hypothesensprache von [Höppner, 2002a] zur Darstellung temporaler Muster. Des Weiteren dient als Support die Anzahl der verschiedenen Eingabesequenzen in denen ein Muster vorkommt. Daher lösen ARMADA und der zuvor beschriebene AET identische Probleme.

Neben der gleichen Problemstellung finden sich auch bei den Algorithmen ähnliche Strategien. ARMADA führt ebenfalls eine Tiefensuche nach den häufigen Mustern des Datensatzes durch. Im Gegensatz zum AET orientiert sich die Tiefensuche nicht nur an den Labels eines Musters, sondern an seinem gesamten Präfix (d.h. den ersten $k-1$ Labels und den dazugehörigen Intervallrelationen des Musters mit k Labels). Darüber hinaus kennt ARMADA eine Indexstruktur, die mit der ISId-Liste von Papapetrou u. a. vergleichbar ist. Die Indexstruktur wird für jedes Muster angelegt und enthält Referenzen auf die Intervalle der Eingabesequenzen, die das erste Vorkommen des Musters in der Eingabesequenz bilden (analog zur ISId-Liste). Als weitere Information ist die Position des letzten zum Muster gehörenden Intervalls in der Indexstruktur abgelegt. Bei der Supportevaluation neuer Muster unterscheiden sich beide Verfahren. ARMADA verknüpft keine ISId-Listen, sondern überprüft mögliche Erweiterungen des aktuellen Präfixes direkt auf der Datenbasis. Die Indexstruktur hilft bei dieser Aufgabe den algorithmischen Aufwand zu minimieren, da sie auf die abzuarbeitenden Eingabesequenzen und die Startposition innerhalb der Eingabesequenzen verweist.

Anwendungen

Lediglich Papapetrou u. a. haben bisher zwei Anwendungen für die Entdeckung häufiger Muster in intervallbasierten Daten mit vielen Eingabesequenzen präsentiert [Papapetrou u. a., 2005]. Beide Anwendungen werden mehr zur Darstellung der algorithmischen Leistungsfähigkeit verwendet, als dass sie ein tatsächlich zu lösendes Problem repräsentieren. In der ersten Anwendung wird der Netzwerkverkehr zwischen zwei Routern untersucht. Die gefundenen häufigen Muster geben einen Hinweis auf das typische Nutzungsverhalten. Bei der zweiten Anwendung werden annotierte Video-Daten der American Sign Language Datenbank [ASL] analysiert. Diese Datenbank enthält Gestiken und Äußerungen von Menschen, während sie sprechen. Papapetrou u. a. untersuchten, welche Gestiken häufig beim Stellen einer W-Frage (Was, Wer, Wie, etc.) vorkommen.

Es ist jedoch vorstellbar, dass die beschriebenen Verfahren bei den in [Last u. a., 2001] vorgestellten Anwendungen zur Analyse von Finanz- und Wetterdaten eingesetzt werden können. Weitere mögliche Anwendungen betreffen die Überwachung von Dialysesitzungen [Bellazzi u. a., 2000] oder die Untersuchung von Roboterdaten [Cohen, 2001]. In all diesen Anwendungen sollen Regeln aus intervallbasierten Daten abgeleitet werden. Jedoch verwenden die Autoren dazu nicht häufige Muster, sondern nutzen Expertenwissen (Bellazzi u. a.) oder verfolgen statistische Ansätze (Cohen und Last u. a.).

3.3 Sonstige Verfahren

Die oben beschriebenen Verfahren zur Suche nach *häufigen* Mustern in zeitbezogenen Daten bilden eine Gruppe von Verfahren innerhalb des temporalen Data Minings. Es gibt weite-

re Methoden, die eine andere Aufgabenstellung verfolgen bzw. eine spezielle Datengrundlage benötigen. Um den Unterschied zu den Verfahren der häufigen Muster darzustellen, werden im Folgenden zwei wichtige Gruppen von Methoden kurz näher beschrieben. Bei der ersten Gruppe handelt es sich um Algorithmen, die eine Menge von Regeln aus zeitbezogenen Daten lernen. Anschließend werden Verfahren betrachtet, die numerische Zeitreihen als Datenbasis voraussetzen.

3.3.1 Extraktion von Regeln

Wie bereits in Abschnitt 2.2.1 beschrieben wurde, lassen sich aus der Menge der häufigen Muster Regeln ableiten. Ein Vorteil dieser Art der Regelerzeugung besteht in ihrer Vollständigkeit bzgl. der minimalen Supportschranke. Das heißt, jede Regel $A \rightarrow B$ mit $\mathrm{Sup}(A \rightarrow B) \geq \mathrm{MinSup}$ kann durch die Menge der häufigen Muster erzeugt werden. Die im Folgenden beschriebenen Verfahren beschreiten einen anderen Weg. Anstatt zunächst häufige Muster zu bilden, extrahieren sie direkt Regeln aus der Datenbasis. Das Ziel der Regelextraktion ist nicht die Vollständigkeit bzgl. einer minimalen Supportschranke, sondern die Erfüllung bestimmter Bewertungsmaße. Prinzipiell gilt jedoch, dass die Menge der gefundenen Regeln immer eine Teilmenge der Regeln ist, die bei entsprechend gewählter minimaler Supportschwelle auch durch häufige Muster generierbar sind.

Der Algorithmus MSDD (*M*ulti-*S*tream *D*ependency *D*etection) [Oates u. Cohen, 1996; Oates u. a., 1997a,b] sucht Regeln der Form:

Wenn A zum Zeitpunkt t geschieht, *dann* folgt B zum Zeitpunkt $t + x$.

Zur Ausgabe von MSDD gehören nur die k besten Regeln gemäß einer benutzerdefinierten Bewertungsfunktion. Die Suche nach diesen Regeln erfolgt in einem Top-Down Ansatz beginnend mit der generellsten Regel (leere Prämisse und leere Konsequenz) hin zu spezifischeren Regeln (vgl. [Webb, 1995]). Eine Beschneidung des Suchraums erfolgt durch eine Abschätzung der oberen Grenze der Bewertungsfunktion mit Hilfe der G-Statistik [Oates u. Cohen, 1996].

Auf Basis von MSDD wurde in weiterführenden Arbeiten von Oates u. a. der Algorithmus MEDD (*M*ulti-*E*vent *D*ependency *D*etection) erstellt [Oates u. a., 1997a, 1998]. MEDD sucht wie bereits MSDD nach den k besten Regeln in einem Top-Down Ansatz. Als Bewertungsmaß für Regeln wird die G-Statistik verwendet [Oates u. a., 1998]. Die Semantik einer Regel $A \rightarrow B$ ist leicht verändert:

Wenn A zum Zeitpunkt t geschieht, *dann* geschieht B im Zeitraum zwischen $t - x$ und $t + x$.

Im Gegensatz zu MSDD können die Regeln von MEDD auch Aussagen über gleichzeitig erfolgende Ereignisse liefern.

Last u. a. präsentieren eine Regelextraktion auf der Basis eines Informationsnetzwerks (information-theoretic connectionist network vgl. [Maimon u. Last, 2000]). Das Netzwerk besteht zum einen aus einer variablen Anzahl von Knoten, die Attribute (Ereignisse) des Datensatzes präsentieren. Zum anderen enthält es Knoten für die Vorhersage der Werte eines zuvor festgelegten Zielattributs. Die Struktur des Netzwerks wird iterativ erlernt. Zunächst ist das Netzwerk leer. Anschließend werden so lange Knoten (samt Kanten und Kantengewichten) in das Netz eingefügt, wie dadurch die bedingte Entropie in Bezug auf das Zielattribut gesenkt

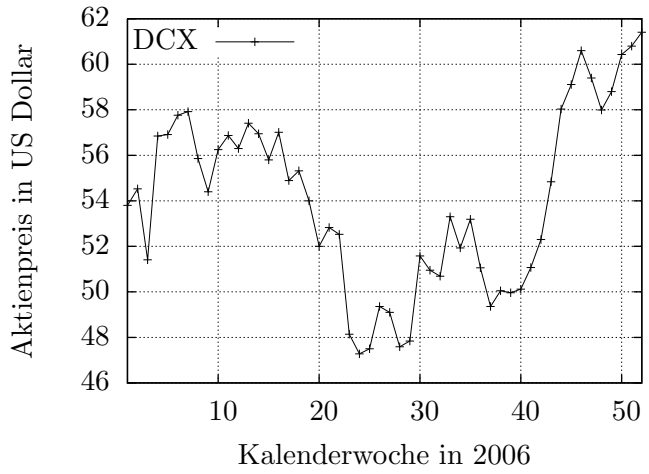


Abbildung 3.23: Beispiel einer Zeitreihe: Entwicklung des Aktienkurses der DaimlerChrysler AG im Jahre 2006

werden kann [Last u. a., 2001]. In einem weiteren Schritt leiten Last u. a. aus dem Netzwerk Regeln der Form: „*Wenn* Attribut A=a, *dann* Zielattribut O = o“ ab.

Analog zu den vorgestellten Verfahren für Regeln, gibt es weitere Verfahren, die nach einer beschränkten Menge an Mustern suchen. Zumeist werden solche Muster gesucht, die auf Grund eines Bewertungsmaßes ungewöhnlich erscheinen (siehe z.B. [Yang u. a., 2001, 2002; Cohen, 2001]).

3.3.2 Zeitreihen

Eine (numerische) Zeitreihe z ordnet einer Menge diskreter Zeitpunkte $\mathbb{T} = \{t_1, \dots t_n\}$ mit $n \in \mathbb{N}$ einen reellen Wert zu $z = \{(t_i, z_i) : t_i \in \mathbb{T}, z_i \in \mathbb{R}, i \in \mathbb{N}\}$. Abbildung 3.23 zeigt als Beispiel einer Zeitreihe den Verlauf eines Aktienkurses im Jahre 2006. In vielen Anwendungen stellen Zeitreihen die Rohform aller zeitbezogenen Daten dar, die durch verschiedene Sensoren gesammelt werden. Beispiele hierfür sind Wetterdaten [Höppner, 2002a, Seite 93ff.], medizinische Daten [Bellazzi u. a., 2000], biomechanische Vorgänge [Mörchen u. a., 2005] oder Finanzmarktdaten [Das u. a., 1998].

In der Literatur werden Zeitreihen mit vielen verschiedenen Zielen und Methoden analysiert. Die nachfolgende Liste an Zielen und Aufgaben für die Verarbeitung von Zeitreihen stammt aus [Antunes u. Oliveira, 2001; Keogh u. Kasetty, 2002, 2003; Lin u. a., 2003, 2004a].

- *Erkennen von Ausreißern* (Anomaly Detection) In einer gegebenen Zeitreihe sollen die Abschnitte mit überraschendem, interessantem, unerwartetem oder neuartigem Verhalten identifiziert werden (siehe z.B. [Dasgupta u. Forrest, 1996; Keogh u. a., 2002]).

- *Klassifikation* (Classification) Gegeben ist eine Klasseneinteilung. Die Klassifikationsaufgabe besteht darin einer noch unklassifizierten Zeitreihe eine Klasse zuzuordnen (vgl. [Geurts, 2001]).

- *Gruppierung* (Clustering) Die Gruppierung soll eine gegebene Menge von Zeitreihen in homogene Gruppen (Cluster) einteilen. Üblich ist die Forderung, dass die Anzahl der Cluster deutlich kleiner ist als die Anzahl der gegebenen Zeitreihen (vgl. [Oates u. a., 1999; Smyth, 1999; Lin u. a., 2004b]).

- *Indizierung* (Indexing) Zu einer gegebenen Zeitreihe sollen die ähnlichsten Zeitreihen bzgl. eines Distanzmaßes in einer Datenbank von Zeitreihen gefunden werden (vgl. [Yi u. Faloutsos, 2000; Keogh u. a., 2001]).

- *Erkennen von Motiven* (Motif Discovery) Motive sind häufig vorkommende Teile von Zeitreihen. In einer gegebenen Menge von Zeitreihen sollen neue Motive gefunden werden [Lin u. a., 2002; Patel u. a., 2002].

- *Vorhersage* (Prediction) Für eine gegebene Zeitreihe soll der weitere Verlauf prognostiziert werden (siehe z.B. [Bradley, 2003]).

- *Abgleich von Teilsequenzen* (Subsequence Matching) In einer (langen) Zeitreihe sollen die Vorkommen einer gegebenen (kurzen) Zeitreihe gefunden werden [Faloutsos u. a., 1994; Keogh u. a., 2001].

- *Segmentierung* (Segmentation) Eine Zeitreihe z mit n Werten soll durch ein aus k (mit $k \ll n$) Abschnitten (Segmenten) bestehendes Modell $\bar{z}$ angenähert werden [Keogh u. Kasetty, 2003]. Die Segmentierung erlaubt eine Abstraktion der Zeitreihen von ihren numerischen Werten zu einer symbolischen Repräsentation. In diesem Zusammenhang stellt sie für viele Anwendungen eine notwendige Form der Datenaufbereitung dar, um Zeitreihen in ereignis- oder intervallbasierte Eingabesequenzen zu konvertieren (siehe Abschnitte 3.1 und 3.2). Eine Übersicht über die verschiedenen Verfahren geben [Höppner, 2002; Keogh u. a., 2004a] sowie [Höppner, 2002a, Kapitel 2]. Zu den eingesetzten Methoden der Segmentierung gehören z.B. das Clustern von Teilen einer Zeitreihe [Das u. a., 1998], die stückweise lineare Annäherung [Keogh, 1997], Landmarks [Perng u. a., 2000] oder die Mehr-Skalen Analyse [Höppner, 2002b].

Für die meisten der oben aufgeführten Aufgaben ist eine Bewertung der Ähnlichkeit zwischen zwei Zeitreihen notwendig. Es wurden viele Ähnlichkeits- und Distanzmaße vorgeschlagen (für eine Übersicht siehe [Keogh u. Kasetty, 2003]), die unterschiedliche Aspekte von Zeitreihen berücksichtigen. Mögliche Anforderungen an Ähnlichkeitsmaße sind z.B. die Robustheit gegen Rauschen, gegen unterschiedliche Skalierungen und gegen Verschiebungen auf der Zeitachse [Antunes u. Oliveira, 2001]. Die Ähnlichkeitsmaße werden in [Antunes u. Oliveira, 2001] und [Mörchen, 2006c, Seite 23] in drei Gruppen eingeteilt: *formbasierte Maße*, *merkmalsbasierte Maße* und *modellbasierte Maße*. Mörchen nennt zusätzlich noch die Gruppe der *kompressionsbasierten Maße* (vgl. [Keogh u. a., 2004b]). Die ersten drei Gruppen werden nachfolgend kurz näher beschrieben.

Formbasierte Maße berücksichtigen bei der Ähnlichkeitsbestimmung das generelle Aussehen der Zeitreihen. Zwei Zeitreihen sind sich ähnlich, wenn sie die gleiche „Form“ besitzen (Auftreten von Bergen, Tälern, Anstiegen, etc.). Der prominenteste Vertreter der formbasierten Maße ist der Euklidsche Abstand. Im einfachsten Fall werden die Abstände der einzelnen Werte zu

jedem Zeitpunkt der Zeitreihe ermittelt und aufsummiert. Erweiterungen dieses Ansatzes ermöglichen den Vergleich unterschiedlich langer Zeitreihen, bzw. berücksichtigen Rauschen und Verschiebungen in den Daten [Agrawal u. a., 1995]. Aber auch andere Distanzmaße aus der $\mathcal{L}_p$-Familie[4] werden angewendet [Yi u. Faloutsos, 2000]. Der Euklidsche Abstand ist anfällig gegen Verzerrungen (bzw. unterschiedliche Skalierungen) auf der Zeitachse. Dynamic Time Warping (DTW) ermöglicht dennoch einen formbasierten Vergleich. DTW transformiert (staucht bzw. streckt) einzelne Abschnitte der Zeitreihen, so dass die resultierenden Zeitreihen ein vorgegebenes Distanzmaß minimieren. Die Summe der Transformationskosten ergeben einen Wert für die Ähnlichkeit der Zeitreihen [Berndt u. Clifford, 1996; Yi u. a., 1998; Keogh u. Pazzani, 1999].

Merkmalsbasierte Maße sind sehr domänenspezifisch. Die grundlegende Idee besteht darin, charakteristische Merkmale aus den Zeitreihen zu extrahieren und diese anschließend zur Bestimmung der Ähnlichkeit heranzuziehen. Der Erfolg dieses Ansatzes hängt stark von der Aussagekraft der gewählten Merkmale ab. Ein Beispiel hierfür sind Anwendungen aus der Betriebsfestigkeit (Teilgebiet des Maschinenbaus). Für Bauteile, die bei Beanspruchung durch Schwingungen verschleißen, lassen sich aus Zeitreihen so genannte „Lastkollektive“ ableiten. Lastkollektive sind durch spezielle Zählverfahren erzeugte Histogramme [Buxbaum, 1992, Seite 12ff.], die eine Aussage über die Schädigung des Bauteils ermöglichen [Buxbaum, 1992, Kapitel 3]. Ein Lastkollektiv ist somit eine Menge von Merkmalen, die aus einer Zeitreihe abgeleitet wurde. Ausgehend von den Lastkollektiven können zwei Zeitreihen einander ähnlich sein, wenn sie das Bauteil ähnlich stark schädigen. Es gibt aber auch merkmalsbasierte Maße, die unabhängig von der jeweiligen Anwendungsdomäne sind. Die diskrete Fourier-Transformation (DFT) [Agrawal u. a., 1993a; Faloutsos u. a., 1994; Rafiei u. Mendelzon, 1997] und die diskrete Wavelet-Transformation (DWT) [Mörchen, 2003] werden z.B. dazu genutzt, um Zeitreihen auf Grund ihres Frequenzspektrums zu vergleichen.

Modellbasierte Maße modellieren den Prozess der für die Entstehung der Zeitreihe verantwortlich ist. Die Güte mit der ein Modell eine gegebene Zeitreihe annähert, kann anschließend als Maß der Ähnlichkeit verwendet werden. Eine weitere Möglichkeit besteht darin die Modellparameter der Zeitreihen zu vergleichen. Beispiele für verwendete Modelle sind die aus der Statistik stammenden ARMA-Prozesse (*A*uto*R*egressive *M*oving *A*verage) [Bradley, 2003; Xiong u. Yeung, 2002, 2004] oder verborgene Markow-Modelle (HMM - Hidden Markov Model) [Rabiner, 1989; Bengio, 1999]. Darüber hinaus werden Mixturen von Modellen eingesetzt [Li u. Biswas, 2000; Cadez u. a., 2000; Xiong u. Yeung, 2004].

3.4 Zusammenfassung

Der erste Teil dieses Kapitels (Abschnitte 3.1 und 3.2) betrachtete Verfahren zur Entdeckung häufiger Muster in zeitbezogenen Daten. Um die Vielfalt an Verfahren überschaubar zu halten wurde eine Systematisierung erstellt, welche die Verfahren anhand der vorausgesetzten Datengrundlage in vier Kategorien klassifiziert. Die einzelnen Kategorien unterscheiden sich jeweils darin, ob die Daten ereignis- oder intervallbasiert sind, bzw. ob sie aus einer oder vielen Eingabesequenzen bestehen.

Die mit Abstand größte Anzahl an Verfahren ist in der Kategorie der ereignisbasierten Daten

[4] In der $\mathcal{L}_p$-Familie sind viele Distanzmaße zusammengefasst – z.B. Manhattan Abstand ($\mathcal{L}_1$), Euklidscher Abstand ($\mathcal{L}_2$) und Maximumabstand ($\mathcal{L}_\infty$).

mit vielen Eingabesequenzen angesiedelt. Insbesondere ist hier die Sequenzanalyse (siehe Seite 21) zu nennen. Für sie existieren mehrere Verfahren, die sich allein in ihren algorithmischen Lösungsansätzen unterscheiden. Einige Verfahren erweitern die Mustersprache der Sequenzen, indem sie die Angabe von Taxonomien, Platzhaltern oder zeitlicher Randbedingungen erlauben. In der Kategorie der ereignisbasierten Daten mit einer Eingabesequenz gibt es Verfahren, die mit Episoden eine zu Sequenzen alternative Mustersprache verwenden. Unter den intervallbasierten Verfahren gibt es zwei verschiedene Ansätze zur Definition von Mustern. Die meisten Verfahren verwenden eine auf Allens Intervallrelationen basierende Mustersprache. Der Gegenvorschlag zu Allens Intervallrelationen besteht in der hierarchisch organisierten Mustersprache TSKR.

Weitere Unterschiede zwischen den Verfahren zeigen sich bei der Wahl der Supportdefinition. Alle Verfahren für viele Eingabesequenzen (egal ob ereignis- oder intervallbasiert) definieren den Support eines Musters als die Anzahl der Eingabesequenzen, die das Muster enthalten. Das heißt, obwohl ein Muster in einer Eingabesequenz mehrfach vorkommen kann, wird bei der Bestimmung des Supports nur ein Vorkommen berücksichtigt. Im Gegensatz dazu müssen Verfahren, die nur mit einer Eingabesequenz arbeiten, auch multiple Vorkommen eines Musters betrachten. Hier haben sich Supportdefinitionen auf Basis von gleitenden Zeitfenstern und das Zählen der minimalen Vorkommen etabliert. Prinzipiell unterliegt die Wahl einer geeigneten Supportdefinition den konkreten Anforderungen der Anwendung. Kapitel 7 gibt einige Beispiele für Anwendungen mit vielen Eingabesequenzen, in denen multiple Vorkommen von Mustern innerhalb einer Eingabesequenz nicht vernachlässigt werden dürfen. Somit kann eine Übertragung der Supportdefinitionen aus den Verfahren für eine Eingabesequenz auf Daten mit vielen Eingabesequenzen helfen, dem temporalen Data Mining neue Anwendungsfelder zu erschließen. Weitere Untersuchungen in dieser Richtung erscheinen daher sinnvoll.

Der zweite Teil dieses Kapitels (Abschnitt 3.3) vervollständigte die Übersicht zur Musterentdeckung in zeitbezogenen Daten. Hier wurden Verfahren aufgezeigt, die Regeln ohne die Hilfe häufiger Muster aus den Daten extrahieren. Abschließend wurden die Aufgaben und Ziele von Verfahren besprochen, die auf numerischen Zeitreihen aufsetzen.

Kapitel 4

Neue Verfahren des temporalen Data Minings

Dieses Kapitel widmet sich einer bisher wenig betrachteten Problemklasse innerhalb der Wissensentdeckung aus zeitbezogenen Daten. Während es eine Vielzahl von Algorithmen gibt, die aus ereignisbasierten Daten lernen (vgl. Kapitel 3), ist die Auswahl an Verfahren für intervallbasierte Daten beschränkt. Durch die Anforderungen einer praktischen Anwendung ist es sogar möglich, dass keines der in Abschnitt 3.2 beschriebenen Verfahren einsetzbar ist.

Eine solche relevante Kombination von Anforderungen besteht darin, dass

- die Daten mehrere Eingabesequenzen enthalten,
- ein Muster mehrmals in einer Eingabesequenz auftreten kann,
- die genaue Anzahl der Vorkommen eines Musters ermittelt werden muss.

Für diese, in der Literatur bisher nicht betrachtete Problemklasse, werden im Folgenden Algorithmen zum Auffinden aller häufigen Muster vorgestellt.

4.1 Problemdefinition

Eine Formalisierung des Problems bildet zunächst die Grundlage für die Auswahl einer geeigneten Hypothesensprache. Anschließend werden mögliche Supportdefinitionen betrachtet und bewertet.

4.1.1 Datengrundlage

Ausgangspunkt eines jeden Wissensentdeckungsprozesses sind die Eingabedaten. Sie sollen Objekte und ihre Beziehungen in der realen Welt möglichst vollständig widerspiegeln. In dem hier betrachteten Fall handelt es sich dabei um Ereignisse, die zu einem bestimmten Zeitpunkt oder in einem bestimmten Zeitraum stattgefunden haben. Diese Informationen lassen sich in einem Tripel (b, e, l) zusammenfassen. Hierbei definieren b und e die Zeitpunkte, zu denen ein Ereignis begonnen bzw. geendet hat. Das Label l beschreibt das Ereignis textuell. Nachfolgend wird ein solches Tripel als *temporales Intervall* bezeichnet. Einige Beispiele für temporale Intervalle sind:

- (9.6.2006, 9.7.2006, 18. Fußballweltmeisterschaft)
- (14.3.1879, 14.3.1879, Geburtstag Albert Einstein)
- (1500 ms, 2315 ms, Druck = hoch)

Je nach Anwendungsdomäne können die Zeitpunkte eines temporalen Intervalls unterschiedlich codiert sein, z.B. als Datum, Zeitstempel oder Rangfolge in einer Ordnung. Im Folgenden wird zur Vereinfachung angenommen, dass sich alle Zeitpunkte durch reelle Zahlen darstellen lassen.

Definition 4.1 *(Temporales Intervall) Für eine gegebene Menge von Labels L wird ein Tripel $(b, e, l) \in \mathbb{R} \times \mathbb{R} \times L$ mit $b \leq e$ als temporales Intervall bezeichnet. Die Menge aller temporalen Intervalle über L ist durch I gegeben.*

Aufbauend auf temporalen Intervallen können auch Ereignisfolgen beschrieben werden, indem eine Sequenz von temporalen Intervallen angegeben wird. Die folgenden Sequenzen erweitern das vorherige Beispiel:

$S_1 =$ (9.6.2006, 9.7.2006, 18. Fußballweltmeisterschaft),
(9.6.2006, 9.6.2006, Eröffnungsspiel),
(9.7.2006, 9.7.2006, Finale)

$S_2 =$ (14.3.1879, 14.3.1879, Geburtstag Albert Einstein),
(1905, 1905, spezielle Relativitätstheorie),
(1916, 1916, allgemeine Relativitätstheorie),
(18.4.1955, 18.4.1955, Todestag)

$S_3 =$ (1500 ms, 2315 ms, Druck = hoch),
(2000 ms, 2600 ms, Temperatur = mittel),
(2320 ms, 2800 ms, Druck = niedrig)

Genügt eine Sequenz von temporalen Intervallen der folgenden Definition, so wird sie als Intervallsequenz bezeichnet.

Definition 4.2 *(Intervallsequenz) Eine gegebene Abfolge von temporalen Intervallen $S = (b_1, e_1, l_1), (b_2, e_2, l_2), \ldots, (b_n, e_n, l_n)$ wird Intervallsequenz genannt, falls die beiden folgenden Bedingungen gelten:*

$$\forall (b_i, e_i, l_i), (b_j, e_j, l_j) \in S, i \neq j : \quad b_i \leq b_j \wedge e_i \geq b_j \Rightarrow l_i \neq l_j \tag{4.1}$$

$$\forall (b_i, e_i, l_i), (b_j, e_j, l_j) \in S, i < j : \quad \begin{array}{r} (b_i < b_j) \vee (b_i = b_j \wedge e_i < e_j) \vee \\ (b_i = b_j \wedge e_i = e_j \wedge l_i < l_j). \end{array} \tag{4.2}$$

Die Menge $\mathbb{S}$ kennzeichnet eine gegebene Menge von Intervallsequenzen.

Gleichung 4.1 ist auch als *Maximalitätsannahme* bekannt und wurde bereits in [Höppner u. Klawonn, 2002] und [Höppner, 2002a, Seite 45ff.] verwendet. Die Maximalitätsannahme fordert, dass jedes temporale Intervall $(b_i, e_i, l_i) \in S$ maximal ist. Maximal bedeutet, dass es kein anderes temporales Intervall $(b_j, e_j, l_j) \in S$ gibt, das sich zeitlich mit (b_i, e_i, l_i) überschneidet und das gleiche Label trägt. Folgendes Beispiel zeigt eine Sequenz, bei der die Maximalitätsannahme nicht erfüllt ist:

$S =$ (9.6.2006, 17.6.2006, Gruppenphase),
(14.6.2006, 22.6.2006, Gruppenphase),
(22.6.2006, 23.6.2006, Gruppenphase)

Offensichtlich werden die Zeitbereiche vom 14.6.2006 bis 17.6.2006 und der 22.6.2006 von unterschiedlichen temporalen Intervallen mit dem gleichen Label belegt. Die Maximalitätsannahme verhindert solche Redundanzen für Intervallsequenzen. Jede Sequenz kann leicht derart transformiert werden, so dass sie die Maximalitätsannahme erfüllt. Dazu müssen lediglich die sich überlappenden temporalen Intervalle mit gleichem Label zu einem neuen temporalen Intervall zusammengefasst werden. Für das obige Beispiel ergibt sich so die einelementige Intervallsequenz: (9.6.2006, 23.6.2006, Gruppenphase).

Gleichung 4.2 verlangt, dass eine Intervallsequenz sortiert ist nach dem Beginn (primär), Ende (sekundär) und Label (tertiär) der enthaltenen temporalen Intervalle. Das heißt, die Sortierung einer beliebigen Sequenz ist zunächst immer durch die Sortierung der Anfangszeitpunkte ihrer temporalen Intervalle gegeben. Bei gleichen Anfangszeitpunkten erfolgt die Sortierung durch die Endzeitpunkte. Erst wenn auch die Endzeitpunkte zweier temporaler Intervalle identisch sind, werden die Labels zum Sortieren benutzt. Die Ordnungsrelation für Beginn und Ende ist durch die Relation $<$ in den reellen Zahlen gegeben. Für die Labels wird die alphabetische Ordnung verwendet.

Im Folgenden wird die Definition von Teilsequenzen benötigt.

Definition 4.3 *(Teilsequenz) Die Intervallsequenz* $S' = (b_i, e_i, l_i)_{1 \leq i \leq m}$ *ist eine Teilsequenz der Intervallsequenz* $S = (b_j, e_j, l_j)_{1 \leq j \leq n}$ *(*$S' \subseteq S$*), falls*

$$\forall (b, e, l): \quad (b, e, l) \in S' \Rightarrow (b, e, l) \in S$$

gilt. Gilt darüber hinaus $m < n$*, so ist* S' *eine echte Teilsequenz von* S *(*$S' \subset S$*).*

Für viele Anwendungen besteht die Datengrundlage aus einer Vielzahl von Intervallsequenzen. Dabei beschreibt jede einzelne Intervallsequenz die Ereignisse für ein Objekt aus der realen Welt, z.B. die Symptome und Behandlungen eines Patienten in einem Krankenhaus oder die Werkstattaufenthalte eines Fahrzeuges.

4.1.2 Temporale Muster

Die Aufgabe besteht darin, aus einer gegebenen Menge von Intervallsequenzen $\mathbb{S}$ die häufigen Muster zu extrahieren. Intuitiv soll ein Muster potentiell interessante Beziehungen zwischen Objekten der realen Welt widerspiegeln. Die Menge aller möglichen Muster bildet die Hypothesensprache. Die verwendete Hypothesensprache ist kritisch bei der Wahl eines geeigneten Verfahrens für ein Anwendungsproblem. Können die von einem Verfahren erzeugten Muster den Anforderungen der Anwendung nicht genügen, so wird es nicht zum Einsatz kommen. Je größer die Ausdrucksstärke einer Hypothesensprache ist, desto flexibler werden die Einsatzmöglichkeiten sein. Die meisten der in Abschnitt 3.2 beschriebenen Verfahren benutzen eine Hypothesensprache, die auf Allens Intervallrelationen (oder einer Teilmenge davon) basieren. Im Folgenden werden ebenfalls Allens Intervallrelationen benutzt, um temporale Muster zu definieren.

Für zwei Ereignisse ohne zeitliche Dauer gibt es zwei mögliche Relationen. Entweder ein Ereignis geschieht vor dem anderen oder sie geschehen zeitgleich. Auf Grund der zeitlichen Ausdehnung temporaler Intervalle sind die möglichen Relationen zwischen zwei temporalen Intervallen komplexer. Es gibt sieben mögliche Relationen (bzw. 13 inklusive der inversen

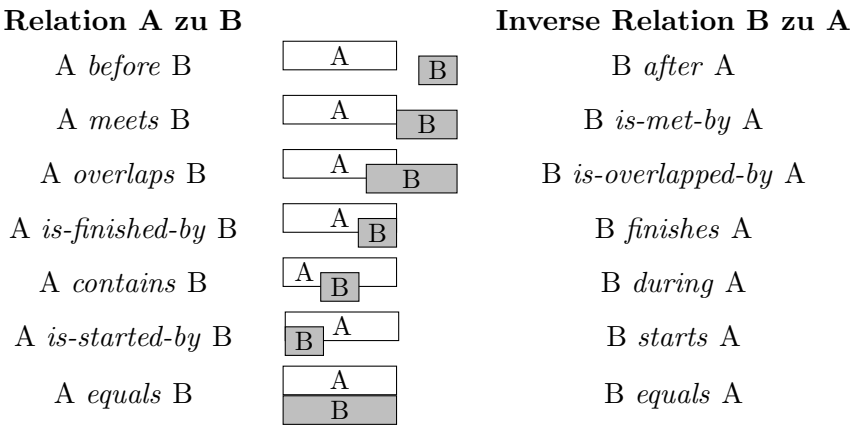


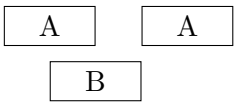
Abbildung 4.1: Allens Intervallrelationen $\mathbb{I}$

Relationen), in denen zwei temporale Intervalle zueinander stehen können. Diese Menge von Relationen wurde zuerst in [Allen, 1983] vorgestellt und ist in Abbildung 4.1 zusammengefasst. Nachfolgend wird die Menge von Allens Intervallrelationen durch $\mathbb{I}$ gekennzeichnet.

Jede Relation in Abbildung 4.1 kann als ein Muster betrachtet werden, das aus zwei temporalen Intervallen besteht. Aufbauend auf Allens Intervallrelationen ist es möglich, Muster zu erstellen, die aus beliebig vielen temporalen Intervallen zusammengesetzt sind. Dazu müssen lediglich alle paarweisen Relationen zwischen den beteiligten Intervallen bekannt sein.

Definition 4.4 *(Temporales Muster) Ein Tupel $P = (s, R)$ mit einer Matrix $R \in \mathbb{I}^{n \times n}$, $n \in \mathbb{N}$ und einer Abbildung $s : (1, \ldots, n) \rightarrow L$ heißt temporales Muster der Größe n bzw. n-Muster. Die Dimension eines Musters ist ebenfalls durch n gegeben ($dim(P) = n$).*

In der obigen Definition enumeriert die Abbildung s alle am Muster beteiligten Labels, während die paarweisen Relationen in der Matrix R gehalten werden. Ein Beispiel für ein temporales Muster der Größe 3 ist in Abbildung 4.2b dargestellt. Die Spalten- und Zeilenköpfe



(a)

	A	B	A
A	e	o	b
B	io	e	m
A	a	im	e

(b)

Abbildung 4.2: (a) graphische Darstellung eines temporalen Musters und (b) entsprechende tabellarische Darstellung

der Matrix entsprechen der Abbildung s und ein Eintrag in der Matrix an der Stelle $[i, j]$ gibt die Intervallrelation zwischen dem i-ten und j-ten Label wieder.

Hierbei stehen die Abkürzungen b, o, m, usw. für die jeweiligen Intervallrelationen. Tabelle 4.1 listet die in dieser Arbeit verwendeten Abkürzungen für Allens Intervallrelationen auf. Im obigen Beispiel aus Abbildung 4.2 sind drei temporale Intervalle (bzw. ihre Labels) beteiligt: zweimal das Label A und einmal das Label B. Das erste A steht zum zweiten A in der Relation

Relation	Abkürzung	inverse Relation	Abkürzung
before	*b*	*after*	*a*
meets	*m*	*is-met-by*	*im*
overlaps	*o*	*is-overlapped-by*	*io*
is-finished-by	*if*	*finishes*	*f*
contains	*c*	*during*	*d*
is-started-by	*is*	*starts*	*s*
equals	*e*		

Tabelle 4.1: Abkürzungen für Allens Intervallrelationen

before und zu B in der Relation *overlaps*. Des Weiteren steht B zum zweiten A in der Beziehung *meets*. Eine einfache graphische Repräsentation dieses 3-Musters ist in Abbildung 4.2a gegeben.

Ein wichtiges Hilfsmittel im Umgang mit temporalen Mustern ist die Definition von Teilmustern.

Definition 4.5 *(Teilmuster) Ein temporales Muster* $P' = (s', R')$ *ist ein Teilmuster des temporalen Musters* $P = (s, R)$ *(geschrieben als* $P' \subseteq P$*), falls* $dim(P') \leq dim(P)$ *und eine injektive Abbildung* $\pi : \{1, \ldots, dim(P')\} \rightarrow \{1, \ldots, dim(P)\}$ *mit*

$$\forall i, j \in \{1, \ldots, dim(P')\} : \qquad s'(i) = s(\pi(i)) \wedge R'[i, j] = R[\pi(i), \pi(j)]$$

existiert.

Die Definition von temporalen Mustern und Teilmustern entspricht im Wesentlichen der aus [Höppner, 2002a, Seite 47] und [Winarko u. Roddick, 2005]. Die Idee der Anordnung von paarweisen Relationen in Form einer Matrix ist bereits in [Vilain u. a., 1989] zu finden.

Es gibt temporale Muster, die nicht in einer Intervallsequenz vorkommen können. Dies ist der Fall, wenn die Einträge in einem temporalen Muster widersprüchlich sind. Ein Beispiel für ein widersprüchliches temporales Muster zeigt Abbildung 4.3. Aus der Matrix ist zu entneh-

	A	B	C
A	e	b	o
B	a	e	o
C	io	io	e

Abbildung 4.3: ein ungültiges temporales Muster

men, dass sowohl A als auch B zu C in der Relation *overlaps* stehen. Daraus folgt, dass der Anfangszeitpunkt von C zwischen den Anfangs- und Endzeitpunkten von A und denen von B liegen muss: $b_A < b_C < e_A$ und $b_B < b_C < e_B$. Jedoch soll auch A zu B in der Beziehung *before* stehen ($e_A < b_B$), was offensichtlich widersprüchlich ist ($b_C < e_A \wedge b_B < b_C \not\rightarrow e_A < b_B$).

Ein weiterer Aspekt sind Mehrdeutigkeiten. Durch Vertauschen von Zeilen und Spalten eines temporalen Musters kann ein neues temporales Muster erzeugt werden. Dieses neue temporale Muster beschreibt exakt dieselben Relationen wie das ursprüngliche Muster. Das heißt, mehrere temporale Muster können die gleiche Aussage über Objekte der realen Welt machen.

Sowohl die Mehrdeutigkeit als auch die Widersprüchlichkeit von temporalen Mustern wird bei der folgenden Definition von Instanzen berücksichtigt.

Definition 4.6 *(Instanz) Eine Intervallsequenz* $S = (b_i, e_i, l_i)_{1 \leq i \leq k}$ *ist eine Instanz des temporalen k-Musters* $P = (s, R)$*, falls*

$$\forall i, j \in \mathbb{N}, 1 \leq i \leq k, 1 \leq j \leq k : s(i) = l_i \wedge s(j) = l_j \wedge R[i, j] = \text{ir}([b_i, e_i], [b_j, e_j])$$

gilt. Hierbei ermittelt die Funktion ir *die Intervallrelation zwischen zwei gegebenen Intervallen. Eine Intervallsequenz* S' *enthält eine Instanz des temporalen Musters* P*, falls* $S \subset S'$ *gilt.*

Die Instanz eines temporalen Musters P ist somit eine Intervallsequenz, deren temporale Intervalle genau in den durch P definierten Relationen zueinander stehen. Daher können zu widersprüchlichen Mustern keine Instanzen existieren. Des Weiteren muss die Enumeration der Labels im temporalen Muster durch die Abbildung s der Reihenfolge der Labels in den temporalen Intervallen der Intervallsequenz entsprechen. Die Sortierung der temporalen Intervalle in einer Instanz erzwingt somit die Eindeutigkeit des temporalen Musters. Die Eindeutigkeit temporaler Muster wird in ähnlicher Weise auch in [Höppner, 2002a, Abschnitt 3.1] und [Winarko u. Roddick, 2005] erzwungen.

Offensichtlich ist die Menge der temporalen Muster, zu denen es Instanzen geben kann, eine Teilmenge aller temporalen Muster. In Definition 4.7 werden die Muster dieser Teilmenge als *gültig* bezeichnet.

Definition 4.7 *(gültiges temporales Muster) Ein temporales Muster wird als gültig bezeichnet, wenn es eine Instanz zu dem Muster geben kann.*

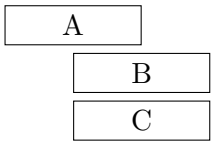
Im Folgenden beschränkt sich die Diskussion auf die gültigen temporalen Muster. Das heißt, nachfolgend ist mit einem temporalen Muster immer ein gültiges temporales Muster gemeint.

Vergleich mit anderen Hypothesensprachen

Sowohl in [Höppner, 2002a] als auch in [Winarko u. Roddick, 2005] werden Muster mit Hilfe einer Auflistung aller beteiligten Intervallrelationen nach Allen definiert. Auf Grund dieser gemeinsamenen Basis ist die vorgestellte Hypothesensprache gleichmächtig zu den in [Höppner, 2002a] und [Winarko u. Roddick, 2005] verwendeten Hypothesensprachen. Im Gegensatz dazu sind die Hypothesensprachen in [Papapetrou u. a., 2005] und [Villafane u. a., 2000] echte Teilmengen der obigen Hypothesensprache, da sie nur auf einem Teil von Allens Intervallrelationen aufbauen (vgl. Abschnitt 3.2).

Das Problem der mehrdeutigen Darstellung des gleichen Musters ist besonders stark bei den Hypothesensprachen aus [Cohen, 2001] und z.T. auch [Kam u. Fu, 2000] ausgeprägt. Jedes Muster, das aus 3 oder mehr temporalen Intervallen besteht, kann auf mehrere Arten repräsentiert werden. Abbildung 4.4 verdeutlicht das Problem an einem einfachen Beispiel.

Mörchen hat in [Mörchen, 2006a,b] und [Mörchen, 2006c] die bisher einzige Hypothesensprache zum Extrahieren häufiger Muster aus Intervalldaten vorgestellt, die nicht auf Allens Intervallrelationen aufbaut. Wie bereits in Abschnitt 3.2 beschrieben, ist Mörchens Hypothesensprache (*TSKR*) hierarchisch organisiert. Auf unterster Ebene stehen die so genannten *Tones*. Sie entsprechen temporalen Intervallen und kennzeichnen das Auftreten eines Ereignisses über



(a)

	A	B	C
A	e	o	o
B	io	e	e
C	io	e	e

(b)

A *overlaps* (B *equals* C)
(A *overlaps* B) *equals* C
A *overlaps* (C *equals* B)
...

(c)

Abbildung 4.4: (a) graphische Repräsentation (b) entsprechendes temporales Muster (c) mehrdeutige Darstellungen in [Cohen, 2001]

einen bestimmten Zeitraum. Aus einer gegebenen Menge von *Tones* werden *Chords* abgeleitet. Jedes *Chord* beschreibt Zeiträume, in dem mehrere *Tones* zusammenfallen. In der letzten Hierarchiestufe werden mehrere *Chords* durch eine partielle Ordnung zu *Phrasen* zusammengefasst. TSKR kann somit die Gleichzeitigkeit und Abfolge von Ereignissen ausdrücken.

Die Nachteile von Allens Intervallrelationen gegenüber TSKR liegen laut [Mörchen, 2006a] in

1. der geringeren Robustheit,
2. der Mehrdeutigkeit,
3. und der schlechteren Verständlichkeit der Muster.

Im Folgenden wird die Argumentation zur Unzulänglichkeit von Allens Intervallrelation im Vegleich zur TSKR aufgegriffen und abgeschwächt. Darüber hinaus werden auch Nachteile der TSKR aufgezeigt.

1. (Robustheit) Unter Robustheit wird die Eigenschaft einer Hypothesensprache verstanden, Rauschen in den Eingangsdaten zu kompensieren [Mörchen, 2006a]. Die Anfangs- und Endpunkte von temporalen Intervallen sind insbesondere dann mit Unsicherheit behaftet, wenn sie erst durch Datenaufbereitungsschritte aus numerischen Zeitreihen gewonnen werden müssen. Zum einen gibt es eine beschränkte Genauigkeit bei der Aufzeichnung numerischer Zeitreihen durch die eingesetzte Messtechnik. Zum anderen können auch die Datenaufbereitungsschritte selbst Verschiebungen der Anfangs- und Endpunkte verursachen (vgl. z.B. [Höppner, 2002a, Abschnitt 2.2]).

   Ein Teil von Allens Intervallrelationen überprüft, ob Zeitpunkte der beteiligten temporalen Intervalle zeitgleich sind. Mit Unsicherheit behaftete Zeitpunkte können dazu führen, dass Instanzen unterschiedlichen Mustern zugeordnet werden, obwohl sie einem menschlichen Betrachter gleichwertig erscheinen. Abbildung 4.5 zeigt ein Beispiel, in dem durch kleine Verschiebungen der Anfangs- und Endpunkte der temporalen Intervalle drei verschiedene temporale Muster entstehen. Im Gegensatz dazu werden in der TSKR alle drei Beispiele aus Abbildung 4.5 durch den Chord AB repräsentiert. Die Toleranz der TSKR gegenüber Verschiebungen der Anfangs- und Endpunkte liegt darin begründet, dass sich die TSKR auf die Darstellung der Konzepte von Gleichzeitigkeit und Abfolge beschränkt. Daher ist laut [Mörchen, 2006a] eine größere Robustheit der TSKR gegeben.

   Jedoch gibt es mehrere Ansätze, die Robustheit von Allens Intervallrelationen zu vergrößern. In [Aiello u. a., 2002] wird z.B. ein Schwellwert eingeführt. Erst wenn die Differenz

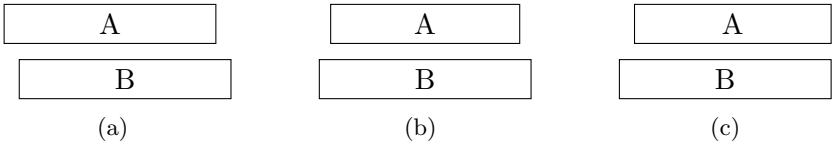


Abbildung 4.5: Kleine Verschiebungen der Anfangs- und Endpunkt führen zu unterschiedlichen Mustern: (a) A *overlaps* B, (b) A *during* B und (c) A *finishes* B.

zwischen zwei Zeitpunkten den Schwellwert überschreitet, werden sie als ungleich angesehen. Andere Ansätze modellieren die Unsicherheit der Zeitpunkte durch Fuzzymengen und konstruieren so fuzzyfizierte Intervallrelationen (vgl. u. a. [Badaloni u. Giacomin, 1999; Dubois u. a., 2003] oder [Schockaert u. a., 2006]). Darüber hinaus ist die Idee von Schwellwerten bereits in [Papapetrou u. a., 2005] bei der Suche nach häufigen Mustern in Intervalldaten erfolgreich angewendet worden.

2. (Mehrdeutigkeit) In [Mörchen, 2006a] werden zwei verschiedene Arten der Mehrdeutigkeit unterschieden. Die erste Art der Mehrdeutigkeit bezieht sich auf die bereits beschriebene mehrfache Darstellung derselben Instanz durch unterschiedliche Muster in [Cohen, 2001] und [Kam u. Fu, 2000] (siehe Abbildung 4.4). Die zweite Art der Mehrdeutigkeit umschreibt Situationen, die von einem menschlichen Betrachter intuitiv unterschieden aber durch Allens Intervallrelationen dem gleichen Muster zugeordnet werden. Abbildung 4.6 zeigt drei Instanzen für das temporale Muster A *overlaps* B. Ein menschlicher Betrachter könnte diese drei Instanzen aber in *kleine Überlappung*, *mittlere Überlappung* und *große Überlappung* unterscheiden. Auf Grund dieser beiden Mehrdeutigkeiten sieht Mörchen den Nachteil von Allens Intervallrelationen begründet.

   Die erste Art der Mehrdeutigkeit lässt sich vermeiden, wenn sich (wie oben beschrieben) die Hypothesensprache auf die gültigen temporalen Muster beschränkt. Auf diese Weise konnten auch in [Höppner, 2002a, Abschnitt 3.1] und [Winarko u. Roddick, 2005] alle Mehrdeutigkeiten verhindert werden.

   Die zweite Art der Mehrdeutigkeit ließe sich z.B. dadurch vermeiden, dass Allens Intervallrelationen um Parameter für Zeitdauern erweitert werden. Die drei Beispiele aus Abbildung 4.6 könnten so anhand der Dauer der Überlappung von A und B unterschieden werden. Eine weitere Möglichkeit ist in [Roddick u. Mooney, 2005] beschrieben. Roddick u. Mooney nutzen die Mittelpunkte von Intervallen, um Allens Intervallrelationen weiter zu verfeinern. In verschiedenen Abstufungen können sie so bis zu 49 verschiedene Rela-

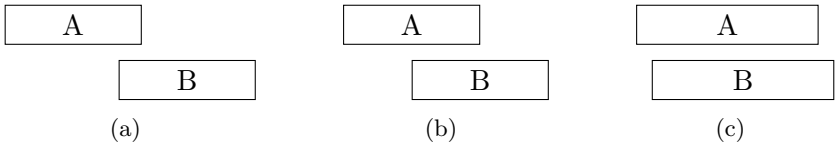


Abbildung 4.6: Drei Instanzen von A *overlaps* B mit (a) kleiner, (b) mittlerer und (c) großer Überlappung.

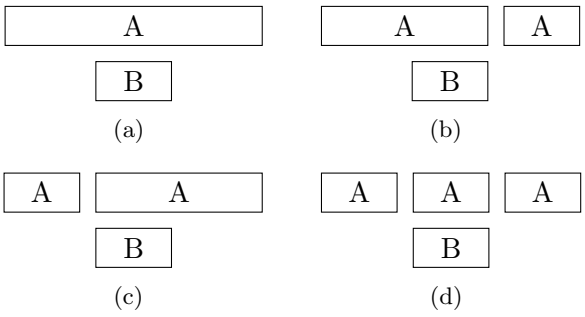


Abbildung 4.7: Vier unterschiedliche Instanzen genügen der Phrase A→AB→A.

tionen unterscheiden. Darunter befinden sich z.B. auch eine *kleine*, *mittlere* und *große Überlappung*. Es gibt daher mehrere Ansätze Allens Intervallrelationen so zu erweitern, dass die zweite Art der Mehrdeutigkeit vermieden bzw. reduziert wird. Jedoch sind diese Ansätze bisher nicht zum Auffinden häufiger Muster in Intervalldaten benutzt worden. Entscheidend ist aber, dass die TSKR selbst alle drei Beispiele aus 4.6 zu dem Chord AB zusammenfasst. Eine Unterscheidung dieser Art der Mehrdeutigkeit ist also auch in der TSKR nicht gegeben.

Durch die Beschränkung der TSKR auf Gleichzeitigkeit und Abfolge können Mehrdeutigkeiten sogar verstärkt auftreten. Als Beispiel soll die Phrase A→AB→A dienen. Wie Abbildung 4.7 zeigt, kann diese Phrase nicht zwischen den verschieden Instanzen unterscheiden. Insbesondere kann nicht unterschieden werden, ob ein, zwei oder drei temporale Intervalle mit dem Label A in der Instanz vorhanden sind. Diese Art der Mehrdeutigkeit ist bei temporalen Mustern auf Basis von Allens Intervallrelationen nicht vorhanden.

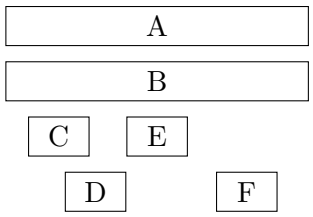
3. (Verständlichkeit) Die Verständlichkeit von Mustern lässt sich schwer quantifizieren. Intuitiv beschreibt sie, inwieweit ein Mensch mit den Mustern einer Hypothesensprache arbeiten kann, d.h. ob sie leicht zu interpretieren oder mit anderen Mustern zu vergleichen sind. Mörchen argumentiert, dass vor allem die paarweise Auflistung aller Intervallrelationen einer guten Verständlichkeit entgegenwirkt.

   Tatsächlich wächst die Anzahl der gelisteten Intervallrelationen quadratisch in Abhängigkeit zu den am Muster beteiligten Intervallen (vgl. Definition 4.4). Daher ist auch eine textuelle bzw. tabellarische Darstellung von temporalen Mustern gegenüber einem Benutzer in Frage zu stellen. Sinnvoller erscheint in diesem Zusammenhang eine graphische Darstellung, wie sie z.B. in Abschnitt 6.2 beschrieben wird. Abbildung 4.8 verdeutlicht den Unterschied zwischen der tabellarischen und graphischen Darstellung anhand eines 6-Musters. Während die tabellarische Darstellung in Abbildung 4.8a nur schwer zu interpretieren ist, ist seine graphische Darstellung in 4.8b intuitiv verständlich.

Die bisherige Diskussion zu den beiden Hypothesensprachen soll eine differenzierte Sichtweise ermöglichen. Die größere Robustheit und größere Mehrdeutigkeit der TSKR sind zwei Seiten derselben Medaille. In beiden Fällen liegt sie in der Beschränkung auf die Konzepte Gleichzeitigkeit und Abfolge begründet. Einerseits können dadurch die Vergleiche auf Gleichzeitigkeit von Zeitpunkten vermieden werden. Andererseits müssen durch die Beschränkung

	A	B	C	D	E	F
A	e	e	c	c	c	c
B	e	e	c	c	c	c
C	d	d	e	o	b	b
D	d	d	io	e	m	b
E	d	d	a	im	e	b
F	d	d	a	a	a	e

(a)



(b)

Abbildung 4.8: (a) ein 6-Muster (b) und seine graphische Darstellung

unterschiedliche Situationen der gleichen Phrase zugeordnet werden (Abbildung 4.7). Welche Hypothesensprache das bessere Werkzeug ist, lässt sich daher nicht pauschal, sondern nur im Kontext der jeweiligen Anwendung feststellen. Erfordert eine Anwendung den Umgang mit unsicheren Daten, die über die unter 1. (Robustheit) beschriebenen Möglichkeiten von Schwellwerten und Fuzzymengen hinausgeht, so sollte die TSKR in Betracht gezogen werden. Ist aber eine Unterscheidung einzelner Muster notwendig (vgl. 2. (Mehrdeutigkeit)), so stellt die TSKR keine Alternative dar. Für die im Rahmen dieser Arbeit untersuchten Anwendungen (vgl. Kapitel 7) spielte die Robustheit gegenüber Rauschen in den Daten keine Rolle. Auch aus diesem Grund entfiel die Wahl auf eine Hypothesensprache, die auf Allens Intervallrelationen beruht.

4.1.3 Supportdefinition

Mit Hilfe der obigen Definitionen von Intervallsequenzen und temporalen Mustern, kann die Problemstellung folgendermaßen beschrieben werden: „Finde alle häufigen temporalen Muster für eine gegebene Menge von Intervallsequenzen $\mathbb{S}$“. Um diese Aufgabe zu lösen, ist die Klärung notwendig, wie die Häufigkeit (bzw. der Support) eines temporalen Musters definiert ist.

Maximale Instanzanzahl

Ein erster Ansatz ist es, die maximal mögliche Anzahl an Instanzen, die sich für ein temporales Muster in einer Intervallsequenz finden lassen, als Supportdefinition zu nutzen. Die Instanzen sollen dabei disjunkt sein. Das heißt, ein temporales Intervall darf höchstens einer Instanz zugeordnet sein.

Höppner zeigt jedoch in [Höppner u. Klawonn, 2002] anhand des folgenden Beispiels, dass diese Supportdefinition neben ihrer schweren Berechenbarkeit zu einer nicht intuitiven Auswahl von Instanzen führt. Abbildung 4.9a zeigt eine Intervallsequenz, die aus temporalen Intervallen mit zwei verschiedenen Labels besteht. Wie oft kommt das Muster aus Abbildung 4.10 in dieser Intervallsequenz vor? Eine einfache Suche kann die Instanzen gemäß der unterschiedlichen Schraffuren in Abbildung 4.9b finden. Da die unbelegten temporalen Intervalle keine Instanz des gesuchten Musters bilden, ist der Support drei. Es ist aber möglich den Support auf vier zu erhöhen, wenn eine andere Zuordnung von temporalen Intervallen zu Instanzen vorgenommen wird. Diese Zuordnung ist in Abbildung 4.9c gekennzeichnet. Jedes einzelne temporale Intervall ist einer Instanz zugeordnet. Somit ergibt sich der maximale Support des gesuchten Musters als vier. Es fällt jedoch auf, dass die einzelnen Instanzen nun eine größere zeitliche

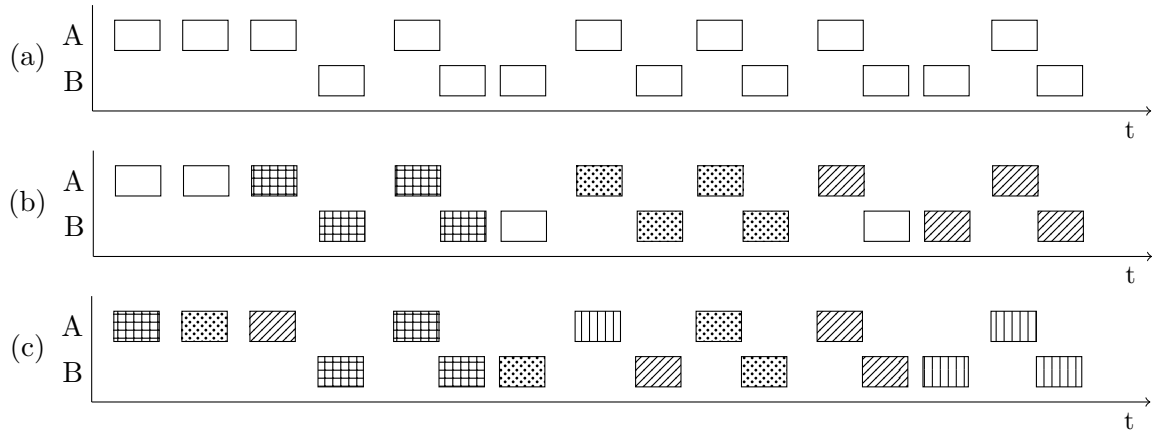
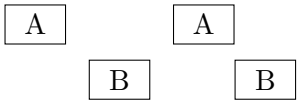


Abbildung 4.9: (a) Eine Intervallsequenz mit zwei verschiedenen Labels. (b) Drei Instanzen des temporalen Musters aus Abbildung 4.10. (c) Eine andere Zuordnung der temporalen Intervalle führt zu einer Maximierung der Instanzanzahl. (Beispiel aus [Höppner u. Klawonn, 2002])

	A	B	A	B
A	e	b	b	b
B	a	e	b	b
A	a	a	e	m
B	a	a	im	e

(a)



(b)

Abbildung 4.10: (a) Temporales Muster für Abbildung 4.9 (b) graphische Repräsentation

Ausdehnung besitzen als die Instanzen in Abbildung 4.9b. Des Weiteren zeigt das Beispiel aus Abbildung 4.9b, dass eine gefundene Zuordnung von temporalen Intervallen zu Instanzen nicht zwingend den maximal möglichen Support liefert. Daher muss eine Supportberechnung verschiedene Zuordnungsmöglichkeiten überprüfen und die maximale Zuordnung ausgeben. Diese Aufgabe ist zwar z.B. durch Backtracking lösbar, kann aber durch die Abhängigkeit zur Länge der Intervallsequenz zu extremen Laufzeiten führen.

Das obige Beispiel soll aufzeigen, dass das Auszählen der Instanzen eines temporalen Musters in einer gegebenen Intervallsequenz keinesfalls trivial ist. Im Folgenden werden daher zunächst weitere mögliche Supportdefinitionen aufgezeigt und auf Ihre Eignung überprüft.

Temporaler Support

Höppner umging die Probleme des Zählens von Instanzen in [Höppner, 2002a], indem er eine auf Zeitfenstern basierende Supportdefinition aus [Mannila u. a., 1997] für Intervallsequenzen adaptierte. Der *temporale Support* (bzw. *frequency* in [Mannila u. a., 1997]) benutzt ein Zeitfenster mit einer vom Anwender spezifizierten Breite Δt. Bei der Bestimmung des Supports

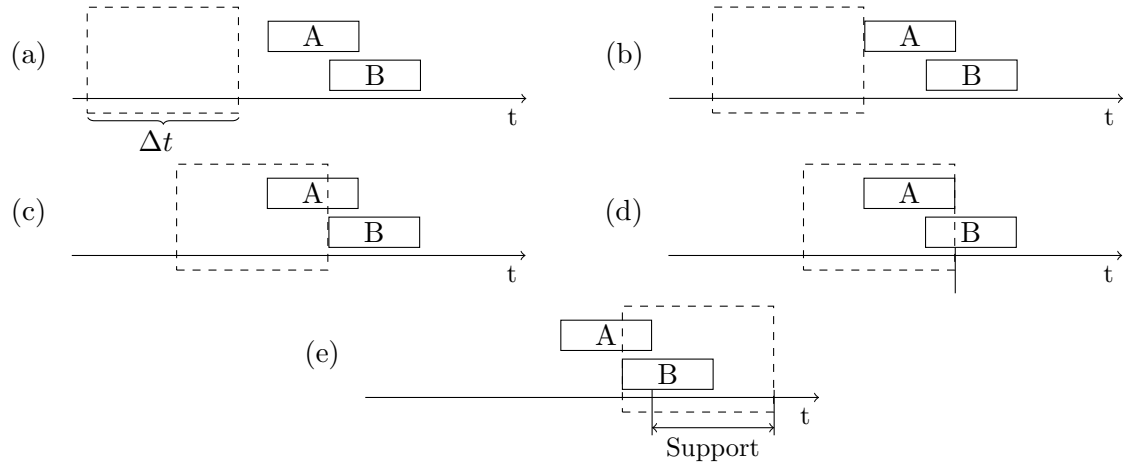


Abbildung 4.11: Beispiel zur Berechnung des temporalen Supports: Das Zeitfenster wird über die Intervallsequenz geschoben. Die Menge aller Fensterpositionen, in denen das temporale Muster *A overlaps B* zu sehen ist, ergeben den temporalen Support.

für ein gegebenes Muster gleitet (engl. *sliding window*) dieses Zeitfenster von links nach rechts über die Eingabesequenz.[1] Der absolute Support ergibt sich anschließend aus der Anzahl der verschiedenen Positionen des Zeitfensters, in denen das gesuchte Muster zu erkennen war. Ein relativer Support ist durch das Verhältnis des absoluten Supports zur Anzahl aller möglichen Fensterpositionen gegeben.

$$\text{Temporaler Support}(P) = \frac{\text{Anzahl der Fenster mit P}}{\text{Anzahl aller Fenster}}$$

Ein Beispiel für die Berechnung des temporalen Supports des Musters *A overlaps B* ist in Abbildung 4.11 gegeben. Zunächst ist kein temporales Muster im Zeitfenster enthalten (Abbildung 4.11a). In Abbildung 4.11b stößt die rechte Kante des Zeitfensters auf das erste temporale Intervall. Aber auch jetzt ist das gesuchte Muster noch nicht im Zeitfenster enthalten. Der Inhalt des Zeitfensters ändert sich das nächste Mal, wenn der rechte Rand auf das zweite temporale Intervall trifft (Abbildung 4.11c). Jetzt ist zwar sowohl ein temporales Intervall mit dem Label A als auch mit dem Label B im Zeitfenster enthalten, aber die Endzeitpunkte beider Intervalle liegen außerhalb des Zeitfensters. Für die im Zeitfenster sichtbaren Daten kann daher noch nicht entschieden werden, ob es sich um ein Vorkommen von *A overlaps B* handelt. Erst wenn der rechte Rand auf das Ende des temporalen Intervalls mit dem Label A trifft, ist die erste Position mit einem Vorkommen des Musters gefunden (Abbildung 4.11d). Von nun an ist immer ein Vorkommen im Zeitfenster sichtbar, bis der Anfang des temporalen Intervalls mit dem Label B auf den linken Rand des Zeitfensters trifft. In Abbildung 4.11e sind die verschieden Fensterpositionen (anhand des rechten Randes) markiert, in denen *A overlaps B* zu erkennen war. Dieser Zeitbereich ist der (absolute) temporale Support von *A overlaps B*.

[1] Die Eingabedaten bestehen nur aus einer einzelnen Sequenz von temporalen Intervallen (bei Höppner) bzw. Ereignissen (bei Mannila u. a.).

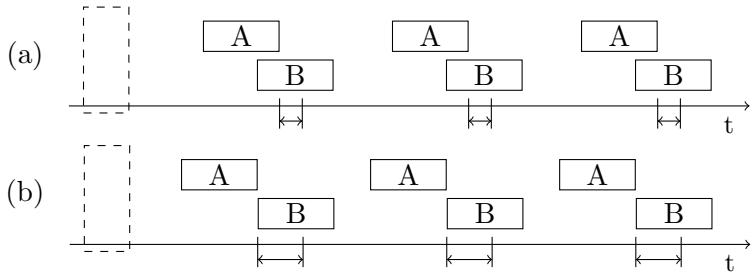


Abbildung 4.12: Obwohl die temporalen Muster *A overlaps B* und *A meets B* in beiden Intervallsequenzen gleich oft Vorkommen, berechnet der temporale Support einen doppelt so hohen Support für das Muster *A meets B*.

Offensichtlich hat die Wahl der Fensterbreite einen direkten Einfluss auf die Berechnung des temporalen Supports. Im obigen Beispiel kann der Support auch direkt als Differenz zwischen der Fensterbreite Δt und der Zeitdauer der Überlappung berechnet werden. Der relative temporale Support kann auch als die Wahrscheinlichkeit interpretiert werden, ein Vorkommen des gesuchten Musters in einem beliebig auf der Eingabesequenz platzierten Zeitfenster zu finden. Und diese Wahrscheinlichkeit steigt, je größer das Zeitfenster ist. Generell betrachtet der temporale Support Zeitdauern von Mustern und nicht deren Häufigkeit, wie auch Abbildung 4.12 demonstriert. In Abbildung 4.12a soll der Support des temporalen Musters *A overlaps B* und in Abbildung 4.12b des Musters *A meets B* bestimmt werden. Die Fensterbreite Δt ist so gewählt, dass sie dem Doppelten der Dauer der Überlappung der temporalen Intervalle in Abbildung 4.12a entspricht. In den jeweiligen Abbildungen ist auch der Support der Muster gekennzeichnet. Obwohl beide Muster intuitiv dreimal in den Intervallsequenzen vorkommen, bestimmt der temporale Support für das Muster *A meets B* einen doppelt so hohen Wert wie für *A overlaps B*. Dieser Unterschied liegt in dem Fehlen einer Überlappung im Muster *A meets B* begründet. Dieses Fehlen führt wiederum zu einer längeren Sichtbarkeit eines Vorkommens innerhalb des gleitenden Zeitfensters.

Es ist daher festzustellen, dass anhand des temporalen Supports eines Musters keine Rückschlüsse auf die Anzahl der Instanzen des temporalen Musters möglich sind. So könnte ein hoher temporaler Support durch viele Vorkommen eines Musters, die nur kurz oder aber durch wenige Muster, die lang im Zeitfenster sichtbar sind, entstehen. Auf Grund dieser Eigenschaften muss der temporale Support (bzw. die Frequency) als mögliche Supportdefinition für den Fokus dieser Arbeit verworfen werden.

Kombinatorisches Zählen

Ein Nachteil der maximalen Instanzanzahl kann am Beispiel der Intervallsequenz in Abbildung 4.13 aufgezeigt werden. Das temporale Muster *A contains B* ist nach der maximalen Instanzanzahl einmal enthalten. Die Zuordnung der temporalen Intervalle zu der Instanz verwendet das temporale Intervall mit dem Label A und eines der temporalen Intervalle mit dem Label B. Das übrig gebliebene temporale Intervall kann kein 2-Muster bilden. Eventuell ist es aber für eine Anwendung sinnvoll, hier zwei Vorkommen des temporalen Musters *A contains*

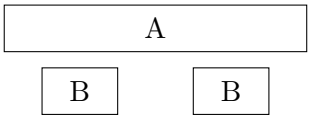


Abbildung 4.13: Kombinatorisches Zählen ermöglicht es, zwei Vorkommen des Musters *A contains B* zu finden.

B zu zählen. Eines für die Dauer des ersten B und eines für die Dauer des zweiten B.

Offensichtlich muss eine Supportdefinition dann die mehrfache Verwendung des temporalen Intervalls A für verschiedene Instanzen erlauben. Eine solche Supportdefinition könnte das kombinatorische Zählen sein. Beim kombinatorischen Zählen bestimmt sich der Support eines temporalen Musters durch die Anzahl aller möglichen Instanzen, die sich in einer Intervallsequenz für das Muster finden lassen.

Agrawalsches Zählen

In [Agrawal u. Srikant, 1995] wurde das Problem der Sequenzanalyse als eine Erweiterung der Warenkorbanalyse eingeführt. Wie bereits in Abschnitt 3.1.2 erläutert, wird der Supportzähler einer Sequenz (eines Musters) immer dann inkrementiert, wenn eine Transaktion das Muster enthält. Dieses Prinzip lässt sich direkt auf Intervallsequenzen übertragen. Der Support eines temporalen Musters P ist somit definiert durch die Anzahl der Intervallsequenzen, die eine Instanz von P enthalten. Diese Supportdefinition wurde auch in [Kam u. Fu, 2000; Papapetrou u. a., 2005] und [Winarko u. Roddick, 2005] für Intervallsequenzen verwendet.

Minimale Vorkommen

Mannila definiert in [Mannila u. a., 1997] die minimalen Vorkommen (minimal occurrences) eines Musters. Ein Vorkommen ist ein Zeitintervall $[b, e]$, das zu einer Sequenz von Ereignissen definiert ist. Außerdem beinhaltet der durch $[b, e]$ definierte Ausschnitt der Ereignissequenz eine Instanz des gesuchten Musters. Das Vorkommen $[b, e]$ ist ein minimales Vorkommen, wenn es kein echtes Teilintervall $[b', e']$ zu $[b, e]$ gibt, das auch ein Vorkommen des Musters ist (vgl. Abschnitt 3.1.1).

Abbildung 4.14 verdeutlicht die Definition von minimalen Vorkommen an einer einfachen Intervallsequenz. Das gesuchte Muster P sei *A before A*. Das Intervall $[1, 5]$ ist kein Vorkommen

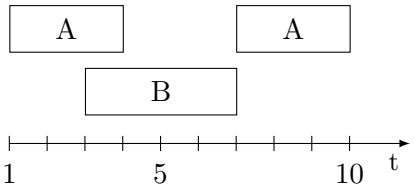


Abbildung 4.14: Das minimale Vorkommen des temporalen Musters *A before A* ist $[4, 7]$.

von P, da in diesem Zeitintervall keine Instanz von P existiert. Im Gegensatz dazu ist $[1, 10]$ ein Vorkommen von P. Allerdings ist $[1, 10]$ kein minimales Vorkommen, da auch das Intervall $[1, 8]$ eine Instanz des Musters enthält. Das einzige minimale Vorkommen ist das Intervall $[4, 7]$. Das Intervall kann nicht weiter verkleinert werden, ohne die Instanz von P zu verlieren. Offensichtlich muss sich ein minimales Vorkommen nicht über die gesamte Zeitdauer einer Instanz erstrecken. Es ist hinreichend, wenn alle Relationen und Labels des temporalen Musters im gegeben Zeitintervall erkenntlich sind. Der Support eines Musters P ist durch die Anzahl aller minimalen Vorkommen von P in den Eingabedaten definiert.

Vergleich der Supportdefinitionen

Tabelle 4.2 zeigt wie unterschiedlich die vorgestellten Definitionen den Support für das Muster *A before B* berechnen. Die Spaltenköpfe zeigen die Intervallsequenzen in denen nach dem Muster gesucht wurde.

Insbesondere die letzte Intervallsequenz im obigen Beispiel zeigt, dass die einzelnen Supportdefinitionen zu vollkommen unterschiedlichen Ergebnissen kommen können. Allerdings gibt es für drei der genannten Supportdefinitionen praktische Hindernisse, die gegen ihren Einsatz sprechen. Bei der maximalen Instanzanzahl gilt die bereits besprochene schwere Berechenbarkeit und die nicht intuitive Instanzzuordnung. So lässt sich für die letzte Intervallsequenz in Tabelle 4.2 die maximale Instanzanzahl auf vier verschiedene Arten erreichen. In einer praktischen Anwendung kann sich die mehrdeutige Instanzzuordnung als irreführend erweisen.

Das Zählen nach Agrawal ist die bisher einzige Supportdefinition, die bereits zum Zählen von Mustern in Intervallsequenzen eingesetzt worden ist. Allerdings ignoriert sie das mögliche mehrfache Auftreten eines Musters in einer Intervallsequenz. Sie ist daher nur einsetzbar, wenn eine Anwendung diese Einschränkung erlaubt. Das definierte Ziel ist es jedoch, auch mehrfache Vorkommen eines Musters zu berücksichtigen. Daher stellt diese Supportdefinition keine Alternative dar.

Auch gegen das kombinatorische Zählen sprechen praktische Erwägungen. Für die zweite Intervallsequenz aus Tabelle 4.2 berechnet das kombinatorische Zählen einen Support von vier für das temporale Muster *A before B*. Allerdings besitzen die beiden Teilmuster *A* und *B* lediglich einen Support von zwei. Bei Verwendung des kombinatorischen Zählens können daher Teilmuster eine geringere Häufigkeit besitzen. Damit ist eine Beschneidung des Suchraums der häufigen Muster durch das Apriori-Kriterium nicht mehr möglich (vgl. Abschnitt 2.2.1, [Agrawal u. a., 1993b]). Alle temporalen Muster des Hypothesenraumes auf ihren Support zu untersuchen, ist jedoch auf Grund der exponentiellen Größe des Hypothesenraumes praktisch nicht durchführbar.

Support(*A before B*)	A B A B	A A B B	A A B A B B
Maximale Instanzanzahl	2	2	3
Kombinatorisches Zählen	3	4	8
Agrawalsches Zählen	1	1	1
Minimale Vorkommen	2	1	2

Tabelle 4.2: unterschiedliche Ergebnisse der verschiedenen Supportdefinitionen

Minimale Vorkommen besitzen nicht die Nachteile der anderen Supportdefinitionen. Die Suche nach minimalen Zeitintervallen ist nicht mehrdeutig und führt zu einer leicht verständlichen Instanzauswahl. Ein Beispiel für eine Supportevaluation gemäß der minimalen Vorkommen ist im Beispiel b der Abbildung 4.9 zu finden. Des Weiteren ist das mehrfache Zählen des Musters *A contains B* in Abbildung 4.13 möglich. Die minimalen Vorkommen ergeben sich direkt aus den beiden temporalen Intervallen mit dem Label B. Minimale Vorkommen sind darüber hinaus gut geeignet, um zeitliche Randbedingungen auf den Instanzen handhaben zu können. Eine Randbedingung kann z.B. sein, dass eine Instanz nur dann gültig ist, wenn sie einen gegebenen Zeitrahmen nicht überschreitet. In diesem Fall brauchen nur die minimalen Vorkommen berücksichtigt werden, deren Zeitintervalle den Zeitrahmen nicht verletzen.

Die folgende Definition erweitert die minimalen Vorkommen aus [Mannila u. a., 1997] für das Zählen der Häufigkeit von temporalen Mustern.

Definition 4.8 *(Minimales Vorkommen) Ein Zeitintervall $[b, e]$ wird für eine gegebene Intervallsequenz S als minimales Vorkommen des k-Musters P ($k \geq 2$) bezeichnet, falls folgende Eigenschaften gelten:*

1. *In dem durch $[b, e]$ definierten Zeitfenster auf S ist eine Instanz von P enthalten. Dabei gilt eine Instanz von P in $[b, e]$ als enthalten, wenn alle an P beteiligten Intervallrelationen eindeutig aus den in $[b, e]$ vorhandenen Informationen ableitbar sind.*

2. *In jedem durch ein echtes Teilintervall $[b', e']$ von $[b, e]$ definierten Zeitfenster auf S ist keine Instanz von P enthalten.*

Das Zeitintervall $[b, e]$ ist ein minimales Vorkommen des temporalen Musters P der Größe 1 ($k = 1$) in der Intervallsequenz S, falls

1. *das temporale Muster (b, e, l) in S enthalten ist und*

2. *l das Label aus P ist.*

Temporale Muster der Größe 1 stellen einen Spezialfall dar. Diese Muster bestehen lediglich aus einem Label (z.B. l). Daher würde jedes temporale Intervall (b, e, l) mit $b < e$ zu einer unendlich großen Anzahl an minimalen Vorkommen führen.[2] Die gegebene Definition von minimalen Vorkommen umgeht dieses Problem, indem sie die Intervalle $[b, e]$ zu jedem temporalen Intervall (b, e, l) als minimale Vorkommen des Musters P (mit dem Label l) heranzieht.

Basierend auf der Definition von minimalen Vorkommen kann der Support eines temporalen Musters definiert werden.

Definition 4.9 *(Support, Häufigkeit) Der Support eines temporalen Musters P für eine gegebenen Menge Intervallsequenzen $\mathbb{S}$ ist definiert durch die Anzahl der minimalen Vorkommen von P in $\mathbb{S}$.*

$$Sup_{\mathbb{S}}(P) = |\{([b, e], S) : [b, e] \textit{ist ein minimales Vorkommen von } P \textit{ in } S \ \wedge S \in \mathbb{S}\}|$$

Definition 4.10 *(Häufiges Muster) Ein temporales Muster wird als häufig bezeichnet, falls sein Support dem Schwellwert MinSup genügt ($Sup_{\mathbb{S}}(P) \geq MinSup$).*

[2] Jedes Zeitintervall $[x, x]$ mit $b \leq x \leq e$ wäre ein minimales Vorkommen.

Die Definition der häufigen Muster durch Vorgabe eines benutzerdefinierten Parameters (MinSup) folgt dem üblichen Vorgehen der Warenkorb- und Sequenzanalyse (vgl. Abschnitte 2.2.1, 3.1 und 3.2). Nachdem die grundlegenden Eigenschaften der hier zu behandelnden Data Mining-Aufgabe geklärt sind, kann die Problemstellung folgendermaßen formuliert werden:

> *Finde alle häufigen temporalen Muster für die gegebene Menge von Intervallsequenzen $\mathbb{S}$ und dem Schwellwert MinSup.*

4.2 Eigenschaften des Problems

Im vorherigen Abschnitt wurde das Problem der Suche nach häufigen temporalen Mustern in Intervallsequenzen formal definiert. Auf dieser Grundlage aufbauend, folgt in diesem Abschnitt eine Untersuchung der Eigenschaften des Problems. Insbesondere werden die Beziehungen zur Warenkorb- und Sequenzanalyse aufgezeigt.

4.2.1 Mächtigkeit

Die Sequenzanalyse erweitert die Warenkorbanalyse in dem Sinne, dass Muster (Sequenzen) aus zeitlichen Abfolgen von Warenkörben bestehen (vgl. [Agrawal u. Srikant, 1995] und Abschnitt 3.1.2). Jeder Algorithmus, der eine Sequenzanalyse durchführt, muss auch die häufigen Warenkörbe (frequent itemsets) berechnen. Aus den Definitionen der verwendeten Hypothesensprachen der Sequenzanalyse und der Warenkorbanalyse folgt daher direkt:

Satz 4.11 *Die Hypothesensprache der Warenkorbanalyse (frequent itemsets) ist eine echte Teilmenge der Hypothesensprache der Sequenzanalyse, Warenkörbe $\subset$ Sequenzen.*

Die Teilmengenbeziehung ist echt, da mit Warenkörben allein keine zeitlichen Abfolgen dargestellt werden können (siehe Abschnitt 2.2.1). Die folgenden Sätze und Beweise zeigen, dass Warenkörbe und Sequenzen durch temporale Muster darstellbar sind.

Satz 4.12 *Jeder Warenkorb ist durch ein temporales Muster darstellbar.*

Der Beweis erfolgt durch Konstruktion. Durch das folgende Vorgehen kann aus jedem Warenkorb ein temporales Muster erzeugt werden.

Beweis 4.13 *Sei $w = \{i_1, i_2, \ldots, i_n\}$ ein Warenkorb. Des Weiteren seien o.B.d.A. die Items des Warenkorbes alphabetisch geordnet. Das zugehörige temporale Muster ist dann gegeben durch:*

	i_1	i_2	$\ldots$	i_n
i_1	e	e	$\ldots$	e
i_2	e	e	$\ldots$	e
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
i_n	e	e	$\ldots$	e

□

Der obige Beweis nutzt die alphabetische Ordnung der Items (bzw. Labels) als Enumeration. Die Relationsmatrix enthält an jeder Stelle den Eintrag *equals*. Durch den Eintrag *equals* bleibt die Semantik des Warenkorbs erhalten. Das temporale Muster besagt, dass $i_1, i_2, \ldots, i_n$ gleichzeitig auftreten. Diese Aussage ist identisch zu einem Warenkorb, der besagt, dass seine Items zusammen vorkommen. Es ist daher möglich mit einem Algorithmus zur Suche nach häufigen temporalen Mustern eine Warenkorbanalyse durchzuführen. Dazu müssen lediglich alle Transaktionen des Datensatzes als Intervallsequenzen codiert werden.[3] Die gefundenen temporalen Muster, die ausschließlich *equals* als Einträge in ihrer Relationsmatrix besitzen, sind die häufigen Warenkörbe.

Auch die Sequenzanalyse ist ein Spezialfall der Suche nach temporalen Mustern, wie der folgende Satz zeigt.

Satz 4.14 *Jede Sequenz ist durch ein temporales Muster darstellbar.*

Analog zur Warenkorbanalyse erfolgt dieser Beweis durch Konstruktion.

Beweis 4.15 *Sei $s = w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_n$ eine Sequenz bestehend aus den Warenkörben $w_k = \{i_1^k, i_2^k, \ldots, i_{|w_k|}^k\}$ mit $|w_k|$ als Anzahl der Items im k-ten Warenkorb. Des Weiteren seien o.B.d.A. die Items jedes Warenkorbs alphabetisch geordnet. Das zugehörige temporale Muster ist dann gegeben durch:*

	i_1^1	i_2^1	$\ldots$	$i_{\lvert w_1\rvert}^1$	i_1^2	i_2^2	$\ldots$	$i_{\lvert w_2\rvert}^2$	$\ldots$	i_1^n	i_2^n	$\ldots$	$i_{\lvert w_n\rvert}^n$
i_1^1	e	e	$\ldots$	e	b	b	$\ldots$	b		b	b	$\ldots$	b
i_2^1	e	e	$\ldots$	e	b	b	$\ldots$	b		b	b	$\ldots$	b
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\vdots$	$\ddots$	$\vdots$
$i_{\lvert w_1\rvert}^1$	e	e	$\ldots$	e	b	b	$\ldots$	b		b	b	$\ldots$	b
i_1^2	a	a	$\ldots$	a	e	e	$\ldots$	e		b	b	$\ldots$	b
i_2^2	a	a	$\ldots$	a	e	e	$\ldots$	e		b	b	$\ldots$	b
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\vdots$	$\ddots$	$\vdots$
$i_{\lvert w_2\rvert}^2$	a	a	$\ldots$	a	e	e	$\ldots$	e		b	b	$\ldots$	b
$\vdots$													
i_1^n	a	a	$\ldots$	a	a	a	$\ldots$	a		e	e	$\ldots$	e
i_2^n	a	a	$\ldots$	a	a	a	$\ldots$	a		e	e	$\ldots$	e
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\vdots$	$\ddots$	$\vdots$
$i_{\lvert w_n\rvert}^n$	a	a	$\ldots$	a	a	a	$\ldots$	a		e	e	$\ldots$	e

□

[3] Eine solche Codierung ist einfach durchzuführen: Jede Transaktion wird in eine Intervallsequenz überführt. Jedes Item der Transaktion bildet ein temporales Intervall der Intervallsequenz. Alle temporalen Intervalle der gleichen Intervallsequenz besitzen identische Beginn- und Endzeiten.

Jeder Warenkorb w_k der Sequenz bildet ein Teilmuster gemäß der Konstruktion aus Beweis 4.13. Die Relation zwischen den Items (Labels) zweier verschiedener Warenkörbe ist *before* bzw. *after*. Die Abbildung auf temporale Muster verändert nicht die Semantik der Sequenz. Die Relation *equals* zeigt an, welche Labels zusammen auftreten (einen Warenkorb bilden). Und die Relation *before* bzw. *after* trennt die einzelnen Warenkörbe und setzt sie in eine zeitliche Abfolge. Analog zur Warenkorbanalyse ist es daher auch möglich, eine Sequenzanalyse mit einem Algorithmus für temporale Muster durchzuführen. Die Eingabedaten der Sequenzanalyse können direkt in Intervallsequenzen überführt werden, indem für jedes Ereignis (Tupel aus Item und Zeitstempel) ein temporales Intervall angelegt wird. Die Beginn- und Endzeiten des temporalen Intervalls entsprechen dem Zeitstempel des Ereignisses. Die häufigen temporalen Muster, deren Relationsmatrizen dem Matrixschema aus Beweis 4.15 genügen, entsprechen dann den häufigen Sequenzen.

Satz 4.16 fasst die Beziehungen der Hypothesensprachen von Warenkörben, Sequenzen und temporalen Mustern zusammen.

Satz 4.16 *Die Hypothesensprachen der Warenkorbanalyse, Sequenzanalyse und der temporalen Muster stehen in folgender Teilmengenbeziehung:*

$$\textit{Warenkörbe} \subset \textit{Sequenzen} \subset \textit{temporale Muster}$$

Beweis 4.17 *Die Teilmengenbeziehungen sind durch die Sätze 4.11, 4.12 und 4.14 gegeben. Die Teilmengenbeziehung zwischen Sequenzen und temporalen Mustern ist echt, da Sequenzen nur einen echten Teil von Allens Intervallrelationen darstellen können (*equals*,* before *und* after*). Als Beispiel dient das temporale Muster:* A overlaps B*. Dieses Muster kann nicht als Sequenz dargestellt werden.*

□

4.2.2 Exponentielle Größe des Hypothesenraumes

Algorithmen, die den Hypothesenraum nach Mustern durchsuchen, hängen in ihrem Laufzeitverhalten von der Größe des Hypothesenraumes ab. In diesem Abschnitt werden Aussagen über die Größe der Hypothesenräume für Warenkörbe, Sequenzen und temporale Muster hergeleitet.

Satz 4.18 *Sei n die Anzahl der verschiedenen Items in einer Warenkorbanalyse. Die Anzahl aller verschiedenen Warenkörbe mit genau k Items ist durch g_w gegeben.*

$$g_w(k,n) = \binom{n}{k}$$

Die Anzahl aller Warenkörbe mit maximal k Items beträgt:

$$G_w(k,n) = \sum_{i=1}^{k} g_w(i,n) \tag{4.3}$$

Beweis 4.19 *Die Items in einem Warenkorb müssen verschieden sein. Daher ist das Problem identisch zu der kombinatorischen Fragestellung „Wie viele Möglichkeiten gibt es k Elemente aus einer n-elementigen Menge zu wählen?“ und kann über den Binomialkoeffizienten $\binom{n}{k}$ berechnet werden.*

□

Auch für die Sequenzanalyse ist die genaue Berechnung der Größe des Hypothesenraumes möglich.

Satz 4.20 *Sei n die Anzahl der verschiedenen Items in einer Sequenzanalyse. Die Anzahl aller verschiedenen Sequenzen mit genau k Items ist durch g_s gegeben.*

$$g_s(k,n) = \begin{cases} n, & \text{falls } k = 1 \\ \binom{n}{k} + \sum_{i=1}^{k-1} \binom{n}{i} g_s(k-i,n), & \text{anderenfalls} \end{cases}$$

Die Anzahl aller Sequenzen mit maximal k Items beträgt:

$$G_s(k,n) = \sum_{i=1}^{k} g_s(i,n) \tag{4.4}$$

Beweis 4.21 *Das Problem bei der Berechnung der Anzahl aller Sequenzen mit k Items besteht in der variablen Anzahl der enthaltenen Warenkörbe. Im Gegensatz zu einfachen Warenkörben kann in einer Sequenz ein Item in mehreren verschiedenen Warenkörben auftreten. Die grundlegende Idee besteht darin, eine vollständige Fallunterscheidung über die Größe des ersten Warenkorbs in einer Sequenz mit k Items durchzuführen. Für eine Sequenz der Größe k gibt es k Möglichkeiten, die Größe des ersten Warenkorbs in der Sequenz zu wählen:*

1.: der erste Warenkorb besteht aus dem ersten Item,

2.: der erste Warenkorb besteht aus den ersten beiden Items,

*..., *

k.: der erste (und einzige) Warenkorb besteht aus den ersten k (allen) Items.

In jedem einzelnen Fall kann die Anzahl der Möglichkeiten, den ersten Warenkorb zu bilden, über den Binomialkoeffizienten berechnet werden (Satz 4.18). Die Bildung des ersten Warenkorbs ist unabhängig von den weiteren Warenkörben in der Sequenz. Daher ist die Anzahl der Sequenzen für jeden Fall das Produkt aus der Anzahl der Möglichkeiten den ersten Warenkorb zu bilden und der Anzahl der Möglichkeiten aus den restlichen Items eine Sequenz zu bilden:

1.: $\binom{n}{1}$ *· Anzahl der Sequenzen mit* $k-1$ *Items,*

2.: $\binom{n}{2}$ *· Anzahl der Sequenzen mit* $k-2$ *Items,*

...,

k.: $\binom{n}{k}$ *· Anzahl der Sequenzen mit* 0 *Items.*

Die Anzahl der Sequenzen kann somit in Abhängigkeit der Anzahl kürzerer Sequenzen dargestellt werden. Sei $g_s(k,n)$ die Anzahl aller Sequenzen mit k Items, so folgt für die Fallunterscheidung:

1.: $\binom{n}{1} g_s(k-1,n)$,

2.: $\binom{n}{2} g_s(k-2,n)$,

$\ldots$,

$k.$: $\binom{n}{k}$.

Da die Fallunterscheidung eine vollständige Partitionierung aller Sequenzen mit k Items durchführt, müssen die Anzahlen über alle Fälle summiert werden.

$$g_s(k,n) = \binom{n}{k} + \sum_{i=1}^{k-1} \binom{n}{i} g_s(k-i,n)$$

Die Rekursion in g_s endet bei $g_s(1,n)$, da eine Sequenz mit einem Item nur aus einem einzelnen Warenkorb bestehen kann.

$$g_s(1,n) = \binom{n}{1} = n$$

In partiell definierter Schreibweise folgt:

$$g_s(k,n) = \begin{cases} n, & \textit{falls } k = 1 \\ \binom{n}{k} + \sum_{i=1}^{k-1} \binom{n}{i} g_s(k-i,n), & \textit{anderenfalls} \end{cases}$$

□

Wie bereits bei der Definition von gültigen temporalen Mustern erläutert wurde (siehe Abschnitt 4.1.2 und das Beispiel aus Abbildung 4.3), sind viele temporale Muster durch die transitiven Eigenschaften von Allens Intervallrelationen widersprüchlich und damit nicht gültig. Eine bekannte Aufgabe des temporalen Schlussfolgerns (temporal reasoning) ist es, für eine gegebene (begrenzte) Menge von Intervallrelationen zu entscheiden ob sie konsistent ist (d.h. ob sie gültig sein kann). Die Lösung dieses Problems ist NP-hart [Vilain u. a., 1989]. Im Folgenden wird daher nur eine obere Schranke für die Größe des Hypothesenraumes gültiger temporaler Muster bestimmt.

Satz 4.22 *In der oberen Dreiecksmatrix eines gültigen temporalen Musters können nur die Relationen* before, meets, overlaps, is-finished-by, contains, starts *und* equals *stehen.*

Beweis 4.23 *Aus der Definition gültiger temporaler Muster (Definition 4.7) folgt, dass die Labels des temporalen Musters dieselbe Reihenfolge haben müssen wie in den Instanzen des Musters. Die temporalen Intervalle jeder Instanz sind wiederum sortiert nach ihren Beginn- (primär) und Endpunkten (sekundär) (vgl. Definition 4.2). Ein gültiges temporales Muster $P = (s, R)$ kann in der Relationsmatrix an der Stelle $R[i,j]$ mit $i < j$ nicht den Eintrag* after *besitzen, da aus der Sortierung der temporalen Intervalle nach ihren Anfangszeitpunkten $i > j$ folgen müsste.*[4] *Die gleiche Argumentation gilt für die Relationen* is-met-by, is-overlapped-by, finishes *und* during. *Die Relation* is-started-by *ist ebenfalls nicht möglich, da $i > j$ aus der (sekundären) Sortierung nach den Endzeitpunkten folgen müsste. Im Gegensatz dazu können sich in der oberen Dreiecksmatrix eines gültigen temporalen Musters die Relationen* before, meets, overlaps, is-finished-by, contains, starts *und* equals *befinden, wie anhand der folgenden Beispiele deutlich wird. Die obere Dreiecksmatrix enthält den Eintrag* before *für das gültige*

[4] $R[i,j]$ mit $i < j$ spezifiziert einen Eintrag in der oberen Dreiecksmatrix.

temporale Muster A before B. *Die anderen Relationen folgen aus den Beispielen* A meets B, A overlaps B, A is-finished-by B, A contains B, A starts B *und* A equals B.

□

Satz 4.24 *Sei n die Anzahl der verschiedenen Labels für eine gegebene Menge von Intervallsequenzen. Eine obere Schranke für die Anzahl aller verschiedenen (gültigen) temporalen Muster der Größe k ist durch g_t gegeben.*

$$g_t(k,n) = n^k 7^{\frac{k(k-1)}{2}}$$

Eine obere Schranke für die Anzahl aller temporalen Muster mit maximal k Labels beträgt:

$$G_t(k,n) = \sum_{i=1}^{k} g_t(i,n) \tag{4.5}$$

Beweis 4.25 *Ein temporales Muster besteht aus zwei Teilen, der Enumeration der Labels und der Relationsmatrix. Eine obere Schranke für die Anzahl aller möglichen temporalen Muster der Größe k kann daher als Produkt der Anzahl der verschiedenen Enumerationen und der Anzahl aller möglichen Relationsmatrizen geschrieben werden. Im Gegensatz zur Warenkorbanalyse gibt es keine Einschränkungen wie oft ein Label in einem temporalen Muster vorkommen darf. Daher gibt es für ein k-Muster n^k verschiedene Enumerationen. Eine Relationsmatrix für k Labels enthält k^2 Einträge. Allerdings steht in der Hauptdiagonalen eines gültigen temporalen Musters immer die Relation* equals *($R[i,i]$ = equals mit $1 \leq i \leq k$). Außerdem sind die Einträge $R[i,j]$ und $R[j,i]$ immer die zu einander inversen Intervallrelationen. Daher beschränkt sich die Anzahl der frei wählbaren Relationen auf maximal $\frac{k^2-k}{2}$. O.b.d.A. seien dies die Einträge in der oberen Dreiecksmatrix. Nach Satz 4.22 können in der oberen Dreiecksmatrix nur sieben verschiedene Intervallrealtionen auftreten. Daher folgt als obere Schranke für die Anzahl der verschiedenen Relationsmatrizen:*

$$7^{\frac{k(k-1)}{2}}$$

Und letztendlich als obere Schranke für die Anzahl der temporalen Muster der Größe k:

$$g_t(k,n) = n^k 7^{\frac{k(k-1)}{2}}$$

□

Anhand der Gleichungen 4.3 und 4.4 kann die Größe des Hypothesenraumes in Abhängigkeit von der maximalen Größe eines Musters für die Warenkorb- und Sequenzanalyse berechnet werden. Gleichung 4.5 liefert eine obere Grenze für die Größe des Hypothesenraumes von temporalen Mustern. In allen drei Fällen wächst der Hypothesenraum rapide mit steigender maximaler Größe der Muster. Abbildung 4.15 zeigt die Entwicklung der Werte von G_i, G_s und G_t für 50 Items bzw. Labels und bis zu einer maximalen Größe der Muster von 15. Das Zahlenbeispiel macht deutlich, dass für die Suche nach häufigen temporalen Mustern im Hypothesenraum eine intelligente Vorgehensweise notwendig ist. Es ist selbst bei kleinsten Problemen nahezu unmöglich, die Häufigkeit aller möglichen Muster zu überprüfen.

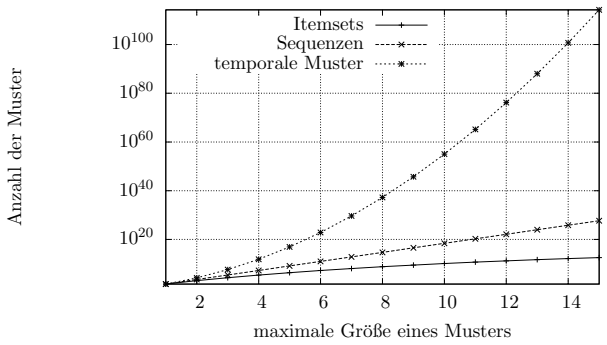


Abbildung 4.15: Hypothesenraumgröße für Warenkörbe, Sequenzen und temporale Muster.

4.2.3 Das Apriori-Kriterium

Alle bisherigen Verfahren (vgl. Abschnitte 3.1 und 3.2) nutzen bei der Suche nach häufigen Mustern das Apriori-Kriterium aus. Das Apriori-Kriterium besagt, dass alle Teilmuster eines häufigen Musters selbst häufig sein müssen. Die Suche nach allen häufigen Mustern lässt sich durch das Apriori-Kriterium effizient organisieren. Zunächst werden die kurzen häufigen Muster ermittelt. Anschließend werden nur solche langen Muster auf ihre Häufigkeit untersucht, die bekannte (kurze) häufige Muster als Teilmuster enthalten. Auf diese Weise lässt sich die Suche von kurzen zu langen Mustern hin organisieren. Je nach Wahl des Schwellwertes *MinSup* kann die Anzahl der tatsächlich überprüften Muster auf einen Bruchteil der Größe des Hypothesenraumes beschränkt werden. Gleichzeitig ist garantiert, dass die Suche alle häufigen Muster findet.

Die Gültigkeit des Apriori-Kriteriums ergibt sich für die Warenkorb- und Sequenzanalyse direkt aus den Problemdefinitionen. Auch bei den von Mannila u. a. und Höppner untersuchten Problemen lässt sich die Gültigkeit nachweisen ([Mannila u. a., 1997] und [Höppner, 2002a, Seite 55]). In dem hier untersuchten Problem ist das Apriori-Kriterium jedoch verletzt. Ursächlich ist die Supportdefinition der minimalen Vorkommen und die Eigenschaften von Allens Intervallrelationen.

Abbildung 4.16 zeigt ein Beispiel für die Verletzung des Apriori-Kriteriums. Es gibt zwei minimale Vorkommen für das temporale Muster A *contains* B, eines für die Zeitdauer des ersten B und eines für die Zeitdauer des zweiten B ($[2, 4]$ und $[6, 8]$). Somit ist der Support

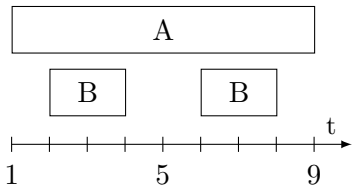


Abbildung 4.16: Verletzung des Apriori-Kriteriums: Der Support des Musters A *contains* B ist zwei, aber der Support des Teilmusters A ist nur eins.

für das Muster A *contains* B im obigen Beispiel Sup(A contains B) = 2. Allerdings ist der Support des 1-Musters A nur Sup(A) = 1. Nach dem Apriori-Kriterium darf aber ein Muster keine Teilmuster mit geringerem Support enthalten. In Abbildung 4.16 ist auch die Ursache für die Verletzung des Apriori-Kriteriums ersichtlich. Die Supportdefinition der minimalen Vorkommen erlaubt die mehrfache Verwendung des temporalen Intervalls mit dem Label A für verschiedene minimale Vorkommen des Musters A *contains* B. Erst durch die mehrfache Verwendung des temporalen Intervalls mit dem Label A wird die Kombination aus dem nicht häufigen Muster A und dem häufigen Muster B häufig.

Eine genauere Untersuchung soll aufzeigen, welche Aussagen über das Apriori-Kriterium trotz der Verwendung minimaler Vorkommen als Supportdefinition gemacht werden können. Als Ausgangspunkt hierfür muss geklärt werden, wie für die gegebene Instanz eines Musters das minimale Vorkommen bestimmt wird. Bereits bei der Einführung minimaler Vorkommen (vgl. Seite 62) und im Beispiel zu Abbildung 4.16 wurde deutlich, dass ein minimales Vorkommen $[b, e]$ nicht die komplette zeitliche Ausdehnung einer Instanz umfassen muss. Vielmehr ist es hinreichend, wenn alle am temporalen Muster beteiligten Labels und Intervallrelationen aus dem Zeitintervall $[b, e]$ ableitbar sind.

Das minimale Vorkommen $[2, 4]$ im Beispiel der Abbildung 4.16 verdeutlicht diese Eigenschaft. Die rechte Grenze des Zeitfensters dürfte nicht kleiner gewählt werden (z.B. $[2, 3]$), da die Information verloren ginge, in welcher Relation die rechten Grenzen der temporalen Intervalle mit dem Label A und B zueinander stehen. Für das Zeitfenster $[2, 3]$ kann daher nicht entschieden werden, ob A zu B in der Relation *is-finished-by*, *contains* oder *overlaps*[5] steht. Der Zeitpunkt 4 ist der kleinste Zeitpunkt an dem entschieden werden kann, dass $e_A > e_B$ gilt. Des Weiteren ist es für die Bestimmung der Intervallrelation irrelevant, wie groß der Abstand zwischen e_A und e_B ist.

Aus dem minimalen Vorkommen $[b, e]$ eines temporalen Musters müssen alle Intervallrelationen des Musters ableitbar sein. Die Entscheidung, welche Intervallrelation zwischen zwei temporalen Intervallen vorliegt, wird anhand des Vergleichs der Anfangs- und Endzeitpunkte getroffen. Daher muss das Zeitintervall $[b, e]$ alle Anfangs- und Endzeitpunkte der betreffenden Instanz umfassen. Lediglich der erste Anfangszeitpunkt und der zeitlich letzte Endzeitpunkt können außerhalb des Intervalls $[b, e]$ liegen, da die Information genügt, dass die Zeitpunkte vor bzw. nach allen anderen Zeitpunkten liegen (vgl. Beispiel aus vorherigem Abschnitt). Abbildung 4.17 verdeutlicht die Vorgehensweise zur Bestimmung des minimalen Vorkommens einer gegebenen Instanz. Zunächst werden alle Anfangs- und Endzeitpunkte der beteiligten Intervalle bestimmt. Der erste (t_{b_1}) und letzte (t_{e_1}) Zeitpunkt sind nicht relevant, um die am Muster beteiligten Intervallrelationen zu bestimmen. Daher ist das minimale Vorkommen der Instanz durch den zweiten (t_{b_2}) und vorletzten (t_{e_2}) Zeitpunkt definiert. Ein Sonderfall entsteht, wenn mehrere verschiedene Anfangs- und Endzeitpunkte aufeinandertreffen (z.B. bei A *equals* B). In diesem Fall müssen die Zeitpunkte mehrfach berücksichtigt werden, damit bei der Entfernung von t_{b_1} (oder t_{e_1}) nicht gleichzeitig t_{b_2} (bzw. t_{e_2}) vernachlässigt wird. Die Bestimmung des Zeitintervalls $[t_{b_2}, t_{e_2}]$ ist ebenfalls ein Zwischenschritt für die Berechnung des *temporalen Supports* (vgl. [Höppner, 2002a, Seite 54ff.]).

Aus den Eigenschaften minimaler Vorkommen folgt die Gültigkeit von Satz 4.26.

[5] Für *contains* muss erkennbar sein, dass das Ende von A größer ist als das Ende von B: $e_A > e_B$. Des Weiteren muss für *is-finished-by* $e_A = e_B$ und für *overlaps* $e_A < e_B$ ableitbar sein.

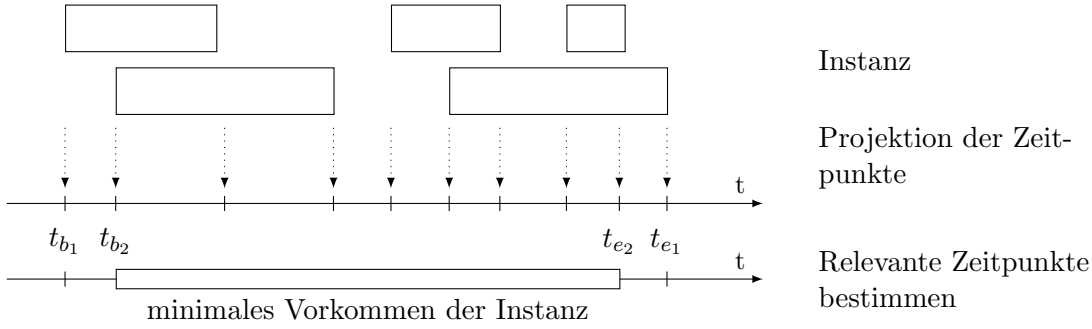


Abbildung 4.17: Durch Projektion der Anfangs- und Endzeitpunkte der temporalen Intervalle auf die Zeitachse wird das minimale Vorkommen $[t_{b_2}, t_{e_2}]$ der Instanz gewonnen. Der erste und letzte Zeitpunkt können außerhalb des minimalen Vorkommens liegen.

Satz 4.26 *(1. Satz über häufige Teilmuster) Sei* P *ein häufiges temporales k-Muster mit* $k > 2$. *Jedes Teilmuster der Größe l von* P *mit* $2 \leq l < k$ *ist häufig.*

Beweis 4.27 *Sei* P *ein temporales Muster der Größe k* ($k > 2$) *mit Sup*$(P) = n$. *Die geordnete Liste aller Anfangs- und Endzeitpunkte der temporalen Intervalle einer Instanz von* P *besitzt* $2k$ *Einträge* $(t_1, t_2, \ldots, t_{2k-1}, t_{2k})$. *Ein minimales Vorkommen von* P *wird durch* $[t_2, t_{2k-1}]$ *gebildet. Aus der Definition minimaler Vorkommen folgt, dass keines der minimalen Vorkommen ein anderes minimales Vorkommen als Teilintervall enthalten kann (vgl. Definition 4.8).*

Sei Q *ein Teilmuster von* P *der Größe l mit* $2 \leq l < k$. *Aus der Teilmusterbeziehung zwischen* P *und* Q *folgt, dass jede Instanz von* P *auch eine Instanz von* Q *enthalten muss. Das minimale Vorkommen von* Q *ergibt sich dann aus einer Teilmenge der Einträge in* $(t_1, t_2, \ldots, t_{2k-1}, t_{2k})$. *Sei* $(t_1^Q, t_2^Q, \ldots, t_{2l-1}^Q, t_{2l}^Q)$ *die geordnete Liste der Zeitpunkte für die Instanz von* Q. *Das minimale Vorkommen von* Q *ist durch* $[t_2^Q, t_{2l-1}^Q]$ *gegeben. Da* Q *mindestens die Größe* 2 *besitzt (d.h. die Liste mindestens vier Zeitpunkte umfasst), muss* $t_1 \leq t_1^Q$, $t_2 \leq t_2^Q$, $t_{2k-1} \geq t_{2l-1}^Q$ *und* $t_{2k} \geq t_{2l}^Q$ *gelten. Das heißt, das minimale Vorkommen von* Q *in einer Instanz von* P *ist ein Teilintervall des minimalen Vorkommens von* P. *Es gibt n minimale Vorkommen von* P *und n dazu entsprechende Instanzen von* P. *Die n minimalen Vorkommen von* P *stehen untereinander in keiner Teilintervallbeziehung, daher können die aus den Instanzen von* P *abgeleiteten minimalen Vorkommen von* Q *nicht in Teilintervallbeziehung zueinander stehen. Somit gibt es mindestens n minimale Vorkommen von* Q*:*

$$Sup(Q) \geq Sup(P).$$

□

Der obige Beweis zu Satz 4.26 verwendet das Viertupel aus den Zeitpunkten $(t_1^Q, t_2^Q, t_{2l-1}^Q, t_{2l}^Q)$. Q ist hierbei das l-Teilmuster des häufigen k-Musters P ($2 \leq l < k$). Damit aus einer Instanz des Musters Q überhaupt das Viertupel abgeleitet werden kann, muss Q mindestens ein temporales Muster der Größe 2 sein. Daher muss die Instanz von Q aus mindestens zwei

temporalen Intervallen bestehen. Die Nebenbedingung garantiert demzufolge, dass das Viertupel $(t_1^Q, t_2^Q, t_{2l-1}^Q, t_{2l}^Q)$ existiert. Um im Folgenden die Häufigkeit von Teilmustern der Größe 1 untersuchen zu können, werden zunächst Suffixe und Präfixe definiert.

Definition 4.28 *(Suffix, Präfix) Sei* P *ein temporales Muster der Größe n. Das Teilmuster der Größe k ($k < n$), welches durch die letzten (ersten) k Labels von* P *definiert ist, wird als k-Suffix (k-Präfix) von* P *bezeichnet.*

Abbildung 4.18 verdeutlicht Präfixe und Suffixe an einem Beispiel.

P	A	B	C
A	e	o	b
B	io	e	m
C	a	im	e

(a)

Q	A	B
A	e	o
B	io	e

(b)

R	B	C
B	e	m
C	im	e

(c)

Abbildung 4.18: (a) P ist ein temporales Muster der Größe 3. (b) Q ist das 2-Präfix und (c) R das 2-Suffix von P.

Satz 4.29 *(2. Satz über häufige Teilmuster) Das* 1*-Suffix eines häufigen temporalen Musters der Größe* 2 *ist häufig.*

Beweis 4.30 *O.B.d.A. sei* P *ein häufiges temporales Muster, das sich aus den beiden Labels* A *und* B *zusammensetzt. Des Weiteren bestehe das* 1*-Suffix von* P *aus dem Label* B*. Gemäß Satz 4.22 sind sieben verschiedene Varianten von* P *zu unterscheiden:* A before B*,* A meets B*,* A overlaps B*,* A is-finished-by B*,* A contains B*,* A starts B *und* A equals B*. Im Folgenden wird für jeden der sieben Fälle ein indirekter Beweis geführt.*

Wäre Satz 4.29 falsch, so müsste der Support von B *kleiner sein als der Support von* P*: Sup(B) < Sup(P). Daraus folgt, dass mindestens ein temporales Intervall mit dem Label* B *an mehreren verschiedenen minimalen Vorkommen von* P *beteiligt sein muss. Für jede der sieben Intervallrelationen ergibt sich ein Widerspruch, wenn ein* B *für verschiedene minimale Vorkommen von* P *wieder verwendet werden soll.*

1. (A before B) *Das minimale Vorkommen einer Instanz des Musters* A before B *setzt sich aus dem Ende von* A *und dem Beginn von* B *zusammen $[e_A, b_B]$. Der Zeitpunkt b_B ist Bestandteil jedes minimalen Vorkommens von* A before B*. Wenn ein temporales Intervall mit dem Label* B *an mehreren minimalen Vorkommen beteiligt ist, so müssten diese minimalen Vorkommen in Teilintervallbeziehungen zueinander stehen, da b_B immer die rechte Grenze der minimalen Vorkommen bildet. Aus Definition 4.8 folgt jedoch, dass minimale Vorkommen nicht in Teilintervallbeziehung zueinander stehen können. (Auf Grund des Widerspruchs muss* B *mindestens so häufig sein wie* A before B*.)*

2. (A meets B, A overlaps B, A is-finished-by B, A contains B, A starts B *und* A equals B) *Für die Wiederverwendung eines temporalen Intervalls* B *für verschiedene minimale Vorkommen des Musters* A meets B *muss es mehrere temporale Intervalle mit dem Label*

A *geben, deren Endzeitpunkte mit dem Beginn von* B *übereinstimmen. In einer Intervallsequenz können auf Grund der Maximalitätsannahme (vgl. Definition 4.2) aber keine zwei temporalen Intervalle existieren, die für denselben Zeitpunkt definiert sind (b_B) und das gleiche Label tragen. Die Verletzung der Maximalitätsannahme folgt analog für den Zeitpunkt b_B bei den Mustern* A overlaps B, A is-finished-by B, A contains B, A starts B *und* A equals B

Aus der vollständigen Fallunterscheidung folgt, dass das 1*-Suffix eines* 2*-Musters nicht seltener sein kann als das* 2*-Muster selbst.*

□

Die Sätze 4.26 und 4.29 lassen sich zu einer Aussage über die Häufigkeit von Suffixen zusammenfassen.

Satz 4.31 *(Partielles Apriori-Kriterium) Sei* P *ein häufiges temporales Muster. Jedes Suffix von* P *ist häufig.*

Beweis 4.32 *Sei* P *ein häufiges temporales k-Muster. Für $k = 2$ gilt die Behauptung, da dieser Fall bereits durch Satz 4.29 abgedeckt ist. Es muss daher nur der Fall $k > 2$ bewiesen werden. Gemäß Satz 4.26 ist jedes l-Suffix von* P *häufig, solange $2 \leq l < k$ gilt (ein Suffix ist ein Teilmuster). Die Häufigkeit des* 1*-Suffixes ergibt sich aus folgender Überlegung. Sei* Q *das* 2*-Suffix von* P *und* R *das* 1*-Suffix von* Q*. Nach Satz 4.29 muss* R *mindestens so häufig sein wie* Q*. Auf Grund von Satz 4.26 muss* Q *mindestens so häufig sein wie* P*. Des Weiteren ist* R *nicht nur das* 1*-Suffix von* Q *sondern auch von* P*. Somit muss jedes Suffix von* P *häufig sein.*
□

Offensichtlich stellt Satz 4.31 eine abgeschwächte Form des Apriori-Kriteriums dar. Beim Apriori-Kriterium der Warenkorb- oder Sequenzanalyse gilt, dass alle Teilmuster eines häufigen Musters selbst auch häufig sind. In dem hier untersuchten Problem ist die Häufigkeit der Teilmuster nur für die Suffixe garantiert. Das partielle Apriori-Kriterium erlaubt es dennoch den Hypothesenraum aller häufigen Muster effizient zu durchlaufen. Prinzipiell müssen dazu zunächst alle kurzen häufigen Muster identifiziert werden (beginnend bei der Größe 1). Anschließend werden nur solche langen Muster auf ihre Häufigkeit hin überprüft, welche die kurzen häufigen Muster als *Suffixe* enthalten. Die bekannte Lösungsstrategie zur Suche nach den häufigen Mustern ist daher aus der Warenkorb- und Sequenzanalyse auf Intervallsequenzen übertragbar. Jedoch muss sichergestellt sein, dass ein konkreter Algorithmus die Suche von den kurzen zu den langen häufigen Mustern auf Basis gemeinsamer Suffixe organisiert. Dies ist insofern bemerkenswert, da etablierte Verfahren der Warenkorb- und Sequenzanalyse die Suche vorwiegend über die Präfixe durchführen (vgl. Kapitel 3, z.B. [Agrawal u. Srikant, 1994a; Masseglia u. a., 1998; Zaki, 1998; Pei u. a., 2001]).

Das partielle Apriori-Kriterium ermöglicht eine effiziente Suche nach allen häufigen temporalen Mustern. Es macht aber keine Aussage darüber, bei welchen temporalen Mustern das (allgemeine) Apriori-Kriterium verletzt ist. Diese Frage wird im Folgenden adressiert.

Definition 4.33 *(Contains-Muster) Ein temporales k-Muster $P = (s, R)$ mit $k \geq 2$ wird Contains-Muster genannt, falls das erste Label des Musters zu allen anderen Labels in der Relation* contains *steht. Das heißt, wenn $\forall i, 1 < i \leq k : R[1, i] =$* contains *gilt.*

Satz 4.34 *(3. Satz über häufige Teilmuster) Sei* P *ein häufiges temporales Muster der Größe* 2 *und* Q *sein* 1*-Präfix. Ist* P *ein Contains-Muster, so kann* $Sup(Q) < Sup(P)$ *gelten. Ist* P *jedoch kein Contains-Muster, so muss* Q *mindestens so häufig sein wie* P*.*

Beweis 4.35 *O.B.d.A. sei* P *ein häufiges temporales Muster, das sich aus den beiden Labels* A *und* B *zusammensetzt. Des Weiteren bestehe das* 1*-Präfix von* P *aus dem Label* A*. Gemäß Satz 4.22 sind sieben verschiedene Varianten von* P *zu unterscheiden:* A before B, A meets B, A overlaps B, A is-finished-by B, A contains B, A starts B *und* A equals B*. Von den sieben möglichen Mustern ist nur das Muster* A contains B *ein Contains-Muster.*

1. (A contains B) *Wie bereits im Beispiel zu Abbildung 4.16 deutlich wurde, kann ein temporales Intervall mit dem Label* A *beliebig viele temporale Intervalle mit dem Label* B *enthalten. Daher kann für eine gegebene Intervallsequenz* $Sup(A) < Sup($A contains B$)$ *gelten.*

Im Folgenden wird für jeden der sechs verbliebenen Fälle ein indirekter Beweis geführt.

Wäre Satz 4.34 falsch, so müsste der Support von A *kleiner sein als der Support von* P*:* $Sup(A) < Sup(P)$*. Daraus folgt, dass mindestens ein temporales Intervall mit dem Label* A *an mehreren verschiedenen minimalen Vorkommen von* P *beteiligt sein muss. Für jede der sechs Intervallrelationen ergibt sich ein Widerspruch, wenn ein* A *für verschiedene minimale Vorkommen von* P *wieder verwendet werden soll.*

2. (A before B) *Das minimale Vorkommen einer Instanz des Musters* A before B *setzt sich aus dem Ende von* A *und dem Beginn von* B *zusammen* $[e_A, b_B]$*. Der Zeitpunkt* e_A *ist Bestandteil jedes minimalen Vorkommens von* A before B*. Wenn ein temporales Intervall mit dem Label* A *an mehreren minimalen Vorkommen beteiligt ist, so müssten diese minimalen Vorkommen in Teilintervallbeziehungen zueinander stehen, da* e_A *immer die linke Grenze der minimalen Vorkommen bildet. Aus Definition 4.8 folgt jedoch, dass minimale Vorkommen nicht in Teilintervallbeziehung zueinander stehen können. (Auf Grund des Widerspruchs muss* A *mindestens so häufig sein wie* A before B*.)*

3. (A meets B, A overlaps B, A is-finished-by B, A starts B *und* A equals B) *Für die Wiederverwendung eines temporalen Intervalls* A *für verschiedene minimale Vorkommen des Musters* A meets B *muss es mehrere temporale Intervalle mit dem Label* B *geben, deren Anfangszeitpunkte mit dem Ende von* A *übereinstimmen. In einer Intervallsequenz können auf Grund der Maximalitätsannahme (vgl. Definition 4.2) aber keine zwei temporalen Intervalle existieren, die für denselben Zeitpunkt definiert sind (*e_A*) und das gleiche Label tragen. Die Verletzung der Maximalitätsannahme folgt analog auch für den Zeitpunkt* e_A *bei den Mustern* A overlaps B, A is-finished-by B, A starts B *und* A equals B

Aus der vollständigen Fallunterscheidung folgt, dass das 1*-Präfix eines* 2*-Musters nur im Falle der Intervallrelation* contains *seltener als das* 2*-Muster seien kann.*

□

Zuletzt können die bisherigen Erkenntnisse über die Eigenschaften häufiger Teilmuster genutzt werden, um die Menge der temporalen Muster, bei denen das allgemeine Apriori-Kriterium nicht gilt, genau zu beschreiben.

Satz 4.36 *(nicht-häufige Teilmuster) Sei* Q *ein Teilmuster des temporalen Musters* P *mit* $Sup(Q) < Sup(P)$*. Dann ist* P *ein Contains-Muster und* Q *ist das* 1*-Präfix von* P*.*

Beweis 4.37 *Sei* P *ein temporales* k*-Muster mit* $Sup(P) = n$*. Nach Satz 4.26 muss jedes Teilmuster der Größe* l $(2 \leq l < k)$ *von* P *mindestens so häufig sein wie* P*. Daher kann ein Teilmuster von* P *nur dann einen geringeren Support aufweisen, wenn es die Größe* 1 *hat. Sei* Q *ein Teilmuster von* P *der Größe* 1 *mit* $Sup(Q) < Sup(P)$*.*

Des Weiteren sei R *ein* 2*-Teilmuster von* P *und* Q *das* 1*-Suffix von* R*. Dann müsste nach Satz 4.26 und 4.29* $Sup(P) \leq Sup(R) \leq Sup(Q)$ *folgen. Auf Grund von* $Sup(Q) < Sup(P)$ *darf* Q *daher nicht als Suffix eines* 2*-Teilmusters von* P *darstellbar sein. Das* 1*-Präfix von* P *(das erste Label in* P*) ist das einzige Teilmuster, welches diese Eigenschaft besitzt.* Q *muss daher das* 1*-Präfix von* P *sein.*

Es gibt $k - 1$ *Teilmuster der Größe* 2 *von* P*, die* Q *als* 1*-Präfix enthalten (*Q *jeweils in Kombination mit dem* 2*-ten,* 3*-ten, . . . ,* k*-ten Label aus* P*). Sei* S *ein beliebiges Teilmuster der Größe* 2 *von* P*, das* Q *als* 1*-Präfix enthält.* S *muss nach Satz 4.26 mindestens so häufig sein wie* P*. Des Weiteren folgt nach Satz 4.34* $Sup(P) \leq Sup(S) \leq Sup(Q)$*, falls* S *kein Contains-Muster ist. Aus diesem Grund müssen alle Teilmuster der Größe* 2 *von* P *mit* Q *als* 1*-Präfix Contains-Muster sein. Daraus folgt, dass* P *selbst ein Contains-Muster ist.*

□

Satz 4.36 bestätigt das intuitive Verständnis, dass ein temporales Intervall (A) beliebig viele Instanzen eines temporalen Musters beinhalten (*contains*) kann. Obwohl die Kombination beider Teile häufig ist, kann A einen geringeren Support aufweisen. Satz 4.36 macht jedoch zusätzlich die starke Aussage, dass die Verletzung des allgemeinen Apriori-Kriteriums ausschließlich Contains-Muster betrifft.

Im folgenden Abschnitt werden Algorithmen vorgestellt, die sowohl das partielle Apriori-Kriterium als auch die Eigenschaften von Contains-Mustern ausnutzen, um eine effiziente Suche nach allen häufigen temporalen Mustern zu realisieren.

4.3 Algorithmen

Im Folgenden werden drei Algorithmen zur Suche nach häufigen temporalen Mustern in einer gegebenen Menge von Intervallsequenzen präsentiert. Der erste Algorithmus *FSMSet* stellt dabei einen direkten Lösungsansatz auf Basis des Apriori-Ansatzes dar. Im zweiten Algorithmus *FSMTree* werden die Ideen aus *FSMSet* aufgegriffen und durch den Einsatz einer effizienten Datenstruktur, in Bezug auf den Speicher- und Laufzeitbedarf, erheblich verbessert. Der dritte Algorithmus *Dip* bildet das Gegenstück zu *FSMSet* und *FSMTree*. Im Gegensatz zum Apriori-Ansatz durchläuft *Dip* den Hypothesenraum mit Hilfe einer Tiefensuche.

4.3.1 FSMSet

Aus der Warenkorbanalyse stammt der Apriori-Ansatz zum Auffinden aller häufigen Muster [Agrawal u. a., 1993b]. Wie bereits in Abschnitt 2.2.1 beschrieben, besteht der Apriori-Ansatz aus zwei Teilen: der Generierung von Kandidaten und der Supportevaluation dieser Kandidaten. Beide Schritte werden abwechselnd so lange wiederholt, bis keine Kandidaten mehr

Muster P	l_1^p	$l_2 \ldots l_k$
l_1^p	e	C
l_2		
$\vdots$	B	A
l_k		

$\cup$

Muster Q	l_1^q	$l_2 \ldots l_k$
l_1^q	e	E
l_2		
$\vdots$	D	A
l_k		

$\Longrightarrow$

Muster R	l_1^p	l_1^q	$l_2 \ldots l_k$
l_1^p	e	?	C
l_1^q	?	e	E
l_2			
$\vdots$	B	D	A
l_k			

Abbildung 4.19: Die temporalen Muster P, Q teilen ein gemeinsames $(k-1)$-Suffix. Die Verknüpfung von P und Q ergibt R.

generiert werden können. Der Apriori-Ansatz beginnt mit den häufigen 1-Mustern und generiert anschließend alle Kandidaten der Größe k aus der Menge der häufigen $(k-1)$-Muster.

Kandidatengenerierung

Generell nutzt die Abfolge von Kandidatengenerierung und Supportevaluation das Apriori-Kriterium auf zwei Arten aus:

(a) Die Generierung der Kandidaten der Größe $(k+1)$ durch Verknüpfung zweier häufiger k-Muster mit einem gemeinsamen $(k-1)$-Präfix garantiert, dass die Menge der Kandidaten eine Obermenge der häufigen $(k+1)$-Muster ist.

(b) Kandidaten, die mindestens ein nicht häufiges Teilmuster enthalten, können (a priori) von der Supportevaluation ausgeschlossen werden.

Wie jedoch im Abschnitt 4.2.3 deutlich wurde, ist das Apriori-Kriterium für Contains-Muster verletzt. Um dennoch eine Beschneidung der Kandidatenmenge zu ermöglichen und die Vollständigkeit der gefundenen häufigen Muster zu garantieren, muss der Apriori-Ansatz angepasst werden.

Apriori verwendet zur Kandidatengenerierung alle Paare von Mustern mit einem gemeinsamen $(k-1)$-Präfix. Für temporale Muster aber muss die Kandidatengenerierung auf Basis gemeinsamer $(k-1)$-Suffixe durchgeführt werden, damit garantiert ist, dass alle häufigen $(k+1)$-Muster in der Menge der Kandidaten enthalten sind (siehe (a) oben).

Die Kandidaten der Größe $k+1$ werden aus den häufigen k-Mustern abgeleitet. Dazu müssen alle Paare von temporalen Mustern P und Q verknüpft werden, die ein gemeinsames $(k-1)$-Suffix besitzen. Abbildung 4.19 zeigt die Vereinigung von P und Q über ihr gemeinsames $(k-1)$-Suffix zu dem vorläufigen Muster R. P und Q beschreiben jeweils den angestrebten $(k+1)$-Kandidaten bis auf ein Label. Das vorläufige Muster R enthält daher alle Intervallrelationen aus den Mustern P und Q. Die einzige unbekannte Information ist die Relation zwischen dem ersten Label aus P und dem ersten Label aus Q (gekennzeichnet durch „?“). Durch Einsetzen von Allens Intervallrelationen für die unbekannte Relation zwischen l_1^p und l_1^q kann das vorläufige Muster R auf bis zu 7 Kandidaten der Größe $k+1$ expandiert werden. Obwohl Allens Intervallrelationen insgesamt aus 13 Relationen bestehen, brauchen maximal nur 7 Kandidaten generiert werden, da laut Satz 4.22 die obere Dreiecksmatrix eines gültigen temporalen Musters nur die Relationen *before*, *meets*, *overlaps*, *is-finished-by*, *contains*, *starts* und *equals* enthalten kann.

Für die Kandidatengenerierung ist es notwendig, in der Menge der häufigen Muster der Größe k (F_k) alle Paare von Mustern mit einem gemeinsamen $(k-1)$-Suffix zu identifizieren. Eine effiziente Lösung dieses Problems ist in [Mannila u. a., 1997] zu finden. Die grundlegende Idee besteht darin, F_k so zu sortieren, dass nach der Sortierung alle Muster mit einem gemeinsamen $(k-1)$-Suffix direkt aufeinander folgen. Die Sortierung kann leicht anhand einer Vektordarstellung der temporalen Muster durchgeführt werden. Sei $P = (s, R)$ ein temporales Muster der Größe k, so ist seine Vektordarstellung wie folgt definiert:[6]

$$P = (\underbrace{\underbrace{\underbrace{s(k)}_{\text{1-Suffix}}, s(k-1), R[k-1,k]}_{\text{2-Suffix}}, s(k-2), R[k-2,k-1], R[k-2,k], \ldots)}_{\text{3-Suffix}}$$

Eine einfache alphabetische Sortierung von F_k nach der Vektordarstellung erzwingt, dass alle temporalen Muster mit einem gemeinsamen $(k-1)$-Suffix einen zusammenhängenden Block in der sortierten Liste bilden. Zwei temporale Muster P und Q können daher nur dann ein gemeinsames $(k-1)$-Suffix besitzen, wenn sie dem gleichen Block angehören.

Die Kandidatengenerierung auf Basis der Vektordarstellung und der Verknüpfung von temporalen Mustern mit gemeinsamen Suffix (Abbildung 4.19) ist in Algorithmus 4.1 dargestellt. Algorithmus 4.1 besteht aus zwei Schleifen. Die äußere Schleife (Zeilen 6–24) iteriert über alle temporalen Muster in der übergebenen Menge F_k. Die innere Schleife (Zeilen 9–23) durchläuft alle temporalen Muster, die zum gleichen Block gehören, wie das gegenwärtig durch die äußere Schleife gekennzeichnete Muster. Die Kombination beider Schleifen iteriert somit über alle Paare von temporalen Mustern, die ein gemeinsames $(k-1)$-Suffix besitzen. Dazu verwendet der Algorithmus die Hilfsfunktion *getBlockStart* (Zeilen 7 und 9). Die Funktion *getBlockStart* liefert für den Index eines Musters in der sortierten Liste F_k den Index des ersten temporalen Musters, das sich im selben Block (d.h. dasselbe $(k-1)$-Suffix besitzt) befindet.

Für jedes Paar von temporalen Mustern führt Algorithmus 4.1 die Verknüpfung beider Muster auf Basis ihres gemeinsamen Suffixes durch (vgl. Abbildung 4.19 und Zeile 10). Zu dem so entstandenen vorläufigen Muster X werden alle Intervallrelationen ermittelt, durch die es zu einem gültigen Muster erweitert werden kann (Hilfsfunktion *getAllowedRelations* Zeile 11). Anschließend benutzt Algorithmus 4.1 alle diese Relationen (Schleife von Zeile 12–21), um X zu dem temporalen Muster P zu erweitern (Hilfsfunktion *expand* Zeile 13).

Wie Satz 4.36 gezeigt hat, ist das allgemeine Apriori-Kriterium nur für Contains-Muster verletzt. Die Hilfsfunktion *isContainsPattern* (Zeile 14) überprüft, ob P ein Contains-Muster ist und fügt P in diesem Fall direkt zu der Menge der Kandidaten (Zeile 15). Ist P jedoch kein Contains-Muster, so wird P einem weiteren Test unterzogen. Die Hilfsfunktion *allSubFrequent* (Zeile 17) testet, ob alle k-Teilmuster von P auch in der Menge der häufigen Muster F_k enthalten sind. Nur wenn dieser Test erfüllt ist, kann P auch selbst häufig sein und wird der Menge der Kandidaten hinzugefügt (Zeile 18).

An zwei Stellen verwendet Algorithmus 4.1 Techniken, die der Reduktion der Anzahl der Kandidaten dient. Zum einen überprüft *allSubFrequent*, ob alle k-Teilmuster von P häufig sind. Ist das nicht der Fall, so wird P nicht in die Menge der Kandidaten aufgenommen. Zum

[6] Vektordarstellungen von temporalen Mustern, allerdings auf Basis ihrer Präfixe, sind bereits in [Höppner, 2002a, Seite 64] und [Papapetrou u. a., 2005] verwendet worden.

Algorithmus 4.1 Kandidatengenerierung für FSMSet

```
1:  F_k: Menge der häufigen k-Muster in sortierter Vektordarstellung
2:  C_k: Menge der Kandidaten der Größe k
3:
4:  procedure CandidateGeneration(F_k)
5:      C_{k+1} := ∅
6:      for i:=1 to |F_k| do
7:          currentBlockStart := getBlockStart(F_k, i)
8:          j := currentBlockStart
9:          while currentBlockStart = getBlockStart(F_k, j) and j ≤ |F_k| do
10:             X := F_k[i] ∪ F_k[j]    ▷ Verknüpfung von F_k[i] und F_k[j] gemäß Abbildung 4.19
11:             R := getAllowedRelations(X)
12:             for all r ∈ R do
13:                 P := expand(X, r)
14:                 if isContainsPattern(P) then
15:                     C_{k+1} := C_{k+1} ∪ {P}
16:                 else
17:                     if allSubFrequent(P, F_k) then
18:                         C_{k+1} := C_{k+1} ∪ {P}
19:                     end if
20:                 end if
21:             end for
22:             ++j
23:         end while
24:     end for
25:     return C_{k+1}
26: end procedure
```

	equals	*starts*	*contains*	*is-finished-by*	*overlaps*	*meets*	*before*
equals	e	—	—	—	—	—	—
starts	s	e, s	—	—	—	—	—
contains	c	c	e, s, c, if, o	c	c	c	c
is-finished-by	if	c	s, o	e, if	c	c	c
overlaps	o	c, if, o	s, o	s, o	e, s, c, if, o	c	c
meets	m	m	s, o	s, o	s, o	e, if	c
before	b	b	s, o, m, b	s, o, m, b	s, o, m, b	s, o, m, b	e, s, c, if, o, m, b

Tabelle 4.3: Abbildungstabelle für die transitiven Eigenschaften von Allens Intervallrelationen (aus den Relationen von A zu C und B zu C folgt A zu B). In den Zeilen ist die Relation zwischen A und C und in den Spalten die Relation zwischen B und C abgetragen. Der Eintrag an der entsprechenden Stelle enthält alle möglichen Relation von A zu B.

anderen dienen nur die Intervallrelationen zur Erweiterung von X, die durch *getAllowedRelations* (Zeile 11) zurückgegeben werden. Die Hilfsfunktion *getAllowedRelations* verwendet die transitiven Eigenschaften von Allens Intervallrelationen, um die Anzahl der Kandidaten zu minimieren. Zur Illustration dient das folgende Beispiel: Sei P das temporale Muster A *before* C und Q das Muster B *starts* C. P und Q besitzen das gemeinsame Suffix C. Sie können daher gemäß Abbildung 4.19 verknüpft werden $X = P \cup Q$. Der fehlende Eintrag in X beschreibt die Relation zwischen A und B. Da B und C den gleichen Anfangszeitpunkt besitzen (B *starts* C), kann für den fehlenden Eintrag nur die Relation *before* in Frage kommen. Alle anderen Relationen würden zu ungültigen (widersprüchlichen) Mustern führen. Im obigen Beispiel treffen die beiden Relationen *before* und *starts* aufeinander. Auch für die anderen Kombinationen lässt sich angeben, welche Intervallrelationen sich zur Erweiterung von X eignen. Tabelle 4.3 gibt die Abbildungstabelle von den beobachteten zu den möglichen Relationen wieder. Die Einträge in Tabelle 4.3 sind auf die Relationen beschränkt, die in der oberen Dreiecksmatrix eines gültigen temporalen Musters stehen können (Satz 4.22). Die Hilfsfunktion *getAllowedRelations* nutzt die Abbildungstabelle unabhängig von der Größe des vorläufigen Musters X. Die Labels von X seien l_1^p, l_1^q, $l_2 \ldots l_k$ (vgl. Abbildung 4.19). Für jedes Label l_i, $2 \leq i \leq k$ liefert Tabelle 4.3 die Menge der möglichen Relationen für die Relation zwischen l_1^p und l_1^q. Die dazu benötige Abfrage besteht aus der Relation zwischen l_1^p und l_i sowie zwischen l_1^q und l_i. Die Schnittmenge aus den einzelnen Abfragen ist somit die Menge der Relationen, die mit den übrigen Einträgen in der Relationsmatrix konsistent sind und zu einem gültigen Muster führen können. Folglich brauchen nur diese temporalen Muster für die Kandidatengenerierung berücksichtigt werden.

Supportevaluation

Nachdem alle Kandidaten der Größe $k + 1$ generiert wurden, folgt die Supportevaluation der Kandidaten. Dazu muss die Anzahl der minimalen Vorkommen jedes Kandidatenmusters bestimmt werden. Wie bereits erwähnt, müssen die Labels in einem gültigen temporalen Muster

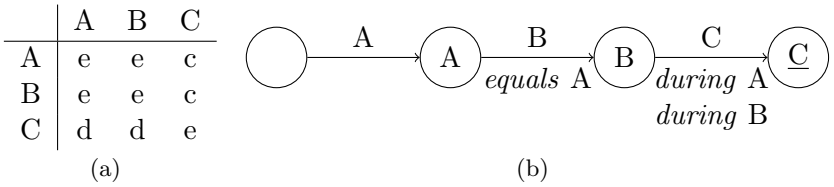


Abbildung 4.20: (a) ein temporales Muster (b) sein endlicher Automat

dieselbe Reihenfolge haben wie in einer Instanz des Musters. Endliche Automaten[7] können diese Eigenschaft ausnutzen, indem sie nacheinander alle temporalen Intervalle einer Intervallsequenz verarbeiten, um die Vorkommen eines temporalen Musters in der Intervallsequenz zu finden. Auch in [Mannila u. a., 1997] werden endliche Automaten zur Supportevaluation genutzt.

Dazu bildet jedes Kandidatenmuster die Grundlage für einen endlichen Automaten. Der endliche Automat besitzt neben einem Anfangszustand für jedes Label des Kandidaten einen individuellen Zustand. Zunächst befindet sich der Automat in seinem Anfangszustand. Alle temporalen Intervalle der Intervallsequenz werden nacheinander in den Automaten eingegeben. Der endliche Automat erreicht seinen nächsten Zustand, sobald am Eingang ein temporales Intervall mit dem gleichen Label wie das erste Label des zu Grunde liegenden Kandidaten erscheint. Um die nächsten Zustände des Automaten zu erreichen, müssen zwei Eigenschaften gegeben sein. Zum einen muss das Label des eingegebenen temporalen Intervalls mit dem nächsten Zustand des Automaten übereinstimmen. Zum anderen muss die Intervallrelation des anliegenden temporalen Intervalls zu allen zuvor akzeptieren temporalen Intervallen den Einträgen in der Relationsmatrix des zugrunde liegenden temporalen Musters entsprechen. Sobald der Automat das letzte fehlende temporale Intervall zu seinem Muster akzeptiert hat, befindet er sich in einem finalen akzeptierten Zustand. Abbildung 4.20 verdeutlicht die Ableitung des endlichen Automaten aus einem temporalen Muster. Für jedes Label des 3-Musters besitzt der Automat einen eigenen Zustand. Der Übergang vom Anfangszustand erfolgt, sobald dem Automaten ein temporales Intervall mit dem Label A eingegeben wird. Für den nächsten Zustand muss am Eingang ein temporales Intervall mit dem Label B anliegen, das zu dem bereits akzeptierten A in der Relation *equals* steht. Der Automat erreicht seinen finalen akzeptierten Zustand, wenn ein temporales Intervall mit dem Label C übergeben wird, das zu den bereits akzeptierten A und B in der Relation *during* steht.

Ein einzelner endlicher Automat nach der obigen Beschreibung kann jedoch „stecken“ bleiben. Als Beispiel dienen die Intervallsequenz aus Abbildung 4.21 und der endliche Automat des temporalen Musters A *meets* B. Der endliche Automat akzeptiert das erste temporale Intervall mit dem Label A. Anschließend folgt aber kein temporales Intervall mit dem Label B, das zu A in der Relation *is-met-by* steht. Der endliche Automat dürfte vom Anfangszustand aus nur das zweite A akzeptieren, um das Vorkommen von A *meets* B zu finden. Auch falls eine Intervallsequenz mehrere Vorkommen eines Musters enthielte, könnten sie mit einem einzelnen endlichen Automaten nicht entdeckt werden.

Um alle Vorkommen eines temporalen Musters zu finden und das Problem von stecken ge-

[7] Eine Einführung in die Theorie der endlichen Automaten ist z.B. in [Wagner, 1994, Seite 178ff.] zu finden.

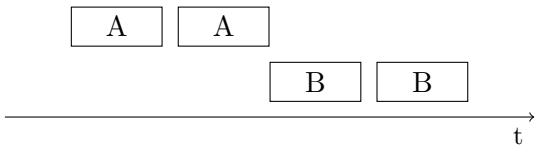


Abbildung 4.21: Der endliche Automat des temporalen Musters *A meets B* bleibt nach akzeptieren des ersten *A* stecken.

bliebenen Automaten zu beheben, wird mit einer Menge von endlichen Automaten gearbeitet.[8] Zunächst enthält die Menge einen endlichen Automaten, der aus dem Kandidatenmuster abgeleitet wird. Anschließend dient jedes temporale Intervall der Intervallsequenz jedem Automaten der Menge als Eingabe. Falls ein Automat ein temporales Intervall akzeptieren kann, so wird eine Kopie des Automaten angelegt und der Menge hinzugefügt. Nur einem der beiden Automaten dient das aktuelle temporale Intervall als Eingabe. Der zweite Automat verbleibt somit in dem Zustand, den der erste Automat vor dem Eingeben des temporalen Intervalls hatte. Auf diese Weise gibt es immer eine Kopie des allerersten Automaten, der noch weitere Instanzen des Musters in der Intervallsequenz finden kann. Generell findet diese Vorgehensweise sogar alle Instanzen eines Musters in einer Intervallsequenz, da jede mögliche Entscheidung (akzeptieren oder nicht akzeptieren eines temporalen Intervalls) von einem der Automaten in der Menge übernommen wird. Somit ist das Problem der stecken gebliebenen Automaten gelöst.

Hat ein endlicher Automat seinen finalen akzeptierten Zustand erreicht, so bilden die akzeptierten temporalen Intervalle eine Instanz des Musters. Aus den Anfangs- und Endzeitpunkten der beteiligten temporalen Intervalle lässt sich das kleinste Zeitfenster bestimmen, in dem diese spezielle Instanz noch sichtbar ist. Zwar ist dieses Zeitfenster ein Vorkommen des Musters, aber es ist nicht eindeutig, ob dieses Vorkommen auch ein minimales Vorkommen des Musters ist. Daher ergibt sich für die Supportevaluation ein 2-schrittiger Prozess. Im ersten Schritt werden alle Vorkommen eines temporalen Musters mit Hilfe von endlichen Automaten gefunden. Im zweiten Schritt, werden alle nicht minimalen Vorkommen herausgefiltert.

Für den zweiten Schritt wird eine Liste der minimalen Vorkommen für jedes Kandidatenmuster angelegt. Erreicht ein Automat seinen finalen akzeptierten Zustand, so wird sein Zeitfenster mit der Liste der minimalen Vorkommen verglichen. Das Zeitfenster kann verworfen werden, wenn die Liste bereits ein Teilintervall des aktuellen Zeitfensters enthält (das Vorkommen war nicht minimal). Umgekehrt ersetzt das aktuelle Zeitfenster alle Vorkommen in der Liste, die das aktuelle Zeitfenster enthalten. Die Liste wird um das aktuelle Zeitfenster erweitert, wenn es zu keinem enthaltenen Zeitfenster in einer Teilintervallbeziehung steht. Durch diesen Filterprozess enthält die Liste am Ende alle minimalen Vorkommen eines temporalen Musters.

Algorithmus 4.2 zeigt das Zusammenspiel der beschriebenen Ideen. Die Parameter von Algorithmus 4.2 sind zum einen die Menge der Kandidatenmuster C und zum anderen die Menge der Intervallsequenzen $\mathbb{S}$. Algorithmus 4.2 iteriert zunächst über alle Intervallsequenzen in $\mathbb{S}$ (Zeilen 5–18). Mit jeder neuen Intervallsequenz initialisiert der Algorithmus die Menge der endlichen Automaten *fsms* mit Hilfe der Funktion *createStateMachines* (Zeile 6). Die Funktion *createStateMachines* legt dabei einen Automaten für jedes Kandidatenmuster an. Anschließend

[8] Die Arbeit mit Mengen von endlichen Automaten motivierte die Benennung des Algorithmus *FSMSet* — finite state machine set.

Algorithmus 4.2 Supportevaluation der Kandidatenmuster für FSMSet

```
 1: 𝕊: Menge der Intervallsequenzen
 2: C: Menge der Kandidaten
 3:
 4: procedure EVALUATESUPPORT(C, 𝕊)
 5:     for all S ∈ 𝕊 do
 6:         fsms := createStateMachines(C)
 7:         for i := 1 ... length(S) do
 8:             (b, e, l) := i-th element of S
 9:             new-fsms := process(fsms, (b, e, l))
10:             for all fsm ∈ new-fsms do
11:                 if fsm.isFinallyAccepted() then
12:                     putSupport(S, fsm)
13:                 else
14:                     fsms := fsms ∪ {fsm}
15:                 end if
16:             end for
17:         end for
18:     end for
19: end procedure
```

iteriert die Supportevaluation über alle temporalen Intervalle der aktuellen Intervallsequenz (Zeilen 7–17). Die Hilfsfunktion *process* nimmt das jeweils aktuelle temporale Intervall und die Menge der Automaten als Argumente (Zeile 9). *Process* implementiert den oben behandelten Kopiervorgang der Automaten. Von jedem Automaten in *fsms*, der das aktuelle temporale Intervall akzeptieren kann, wird eine Kopie angelegt. Alle Automatenkopien bekommen das aktuelle temporale Intervall als Eingabe. Die Menge der Kopien bildet den Rückgabewert von *process* und wird *new-fsms* zugewiesen. Somit bleibt *fsms* durch *process* selbst unverändert und enthält nach wie vor alle Automaten. In *new-fsms* sind alle Kopien der Automaten zu finden, die das temporale Intervall akzeptiert haben. Anschließend überprüft Algorithmus 4.2, ob alle Automaten in *new-fsms* ihren finalen akzeptierten Zustand erreicht haben (Zeile 11 innerhalb der Schleife von Zeile 10 bis 16). Ist das der Fall, so wird mit Hilfe der Funktion *putSupport* das gefundene Vorkommen in der aktuellen Intervallsequenz gegen die Liste der minimalen Vorkommen geprüft. In *putSupport* ist daher der Filterschritt des 2-stufigen Prozesses implementiert (Zeile 12). Hat der Automat seinen finalen akzeptierten Zustand noch nicht erreicht, so wird er der Menge *fsms* hinzugefügt (Zeile 14). Dadurch steht der Automat später für weitere temporale Intervalle der Intervallsequenz zur Verfügung.

In einer Anwendung kann es zeitliche Rahmenbedingungen für Instanzen geben. Z.B. könnte eine Instanz nur dann gültig sein, wenn sie eine gewisse zeitliche Ausdehnung nicht überschreitet. Diese zeitlichen Bedingungen lassen sich leicht in die Supportevaluation einfügen. Bei der Bestimmung der minimalen Vorkommen durch *putSupport* dürfen lediglich die minimalen Vorkommen in die Liste aufgenommen werden, die die zeitlichen Bedingungen nicht verletzen. Zeitliche Bedingungen helfen aber auch, die Anzahl der endlichen Automaten auf einfache Weise zu beschränken. Viele Automaten in der Menge können stecken bleiben und werden

ihren finalen akzeptierten Zustand nie erreichen. Diese Automaten verlangsamen die Laufzeit, da sie dennoch algorithmischen Aufwand verursachen, ohne jemals zum Support eines Kandidaten beizutragen. Daher ist es hilfreich, diese Automaten frühzeitig zu identifizieren und zu entfernen. Die Hilfsfunktion *process* (Zeile 9) überprüft alle Automaten, ob sie das aktuelle temporale Intervall akzeptieren können. Anhand des temporalen Intervalls kann jeder Automat die gegenwärtige Mindestdauer seines Vorkommens überprüfen (unabhängig davon, ob er das temporale Intervall akzeptieren kann oder nicht). Auf Grund der Sortierung der Intervallsequenzen können später kommende temporale Intervalle nur zu längeren Vorkommen führen. Überschreitet die gegenwärtige Mindestdauer eines Automaten bereits die zeitlichen Rahmenbedingungen, so kann er aus der Menge der Automaten entfernt werden, ohne dass die Gefahr besteht, ein minimales Vorkommen (das den zeitlichen Rahmenbedingungen genügt) nicht zu finden.

Apriori-Ansatz

Die Kandidatengenerierung und die Supportevaluation werden gemäß des Apriori-Ansatzes abwechselnd aufgerufen, bis es keine weiteren Kandidaten mehr gibt. Da für temporale Muster nur das partielle Apriori-Kriterium (Satz 4.31) gilt, muss der Apriori-Ansatz für die Anwendung auf temporale Muster angepasst werden. Erst temporale Muster der Größe 2 besitzen echte Suffixe, die zur Kandidatengenerierung genutzt werden können. Der Ausgangspunkt des Apriori-Ansatzes ist daher die Menge der häufigen 2-Muster, deren Vollständigkeit durch *FSMSet* sichergestellt werden muss.

Algorithmus 4.3 zeigt das Zusammenspiel von Kandidatengenerierung und Supportevaluation. *FSMSet* erstellt zuerst die Menge der 1-Kandidaten, indem es jedes verfügbare Label des Datensatzes benutzt (Zeile 9). Anschließend folgt so lange die Schleife aus Supportevaluation der Kandidaten und Generierung der neuen Kandidaten, bis keine weiteren Kandidaten mehr erstellt werden können (Zeilen 10–23). *FSMSet* bestimmt zunächst den Support der Kandidaten aus der vorherigen Iteration mit Hilfe der Funktion *evaluateSupport* (Zeile 11). Jeder Kandidat, dessen Support der minimalen Supportschranke *MinSup* genügt, wird anschließend in die Menge der häufigen Muster aufgenommen (Zeilen 13–15). Nachdem *FSMSet* die häufigen Muster der aktuellen Iteration bestimmt hat, folgt die Kandidatengenerierung für die nächste Iteration. Hierbei werden zwei Fälle unterschieden (Zeilen 17–21). Befindet sich die Schleife noch in ihrem ersten Durchlauf (d.h. es müssen die Kandidaten der Größe 2 generiert werden), so nutzt der Algorithmus die Menge der 1-Kandidaten (Zeile 18). Für alle späteren Schleifendurchläufe erfolgt die Generierung der Kandidaten auf Basis der häufigen Muster der aktuellen Iteration. Die Menge der Kandidaten der Größe 1 und 2 enthält daher alle temporalen Muster, die mit der gegebenen Menge von Labels L möglich sind. Daher ist garantiert, dass die Supportevaluation alle häufigen Muster der Größe 1 und 2 findet. Die Vollständigkeit der häufigen k-Muster $k \geq 3$ ist anschließend durch das partielle Apriori-Kriterium garantiert.

4.3.2 FSMTree

Der Hauptaufwand von *FSMSet* liegt in der Verwaltung der endlichen Automaten. Für jedes temporale Intervall in den Eingabedaten überprüft *FSMSet*, ob es von einem der Automaten akzeptiert werden kann. Des Weiteren wächst die Menge der Automaten durch jedes akzep-

Algorithmus 4.3 FSMSet – Algorithmus zum Auffinden aller häufigen Muster

```
1:  𝕊: Menge aller Intervallsequenzen
2:  L: Menge der Labels in 𝕊
3:  MinSup: minimaler Support
4:  C_i: Menge aller Kandidatenmuster in der i-ten Iteration
5:  F_i: Menge aller häufigen Muster in der i-ten Iteration
6:
7:  procedure FindFrequentPatterns(𝕊, L)
8:      k := 1
9:      C_k := {l ∈ L}
10:     repeat
11:         evaluateSupport(C_k, 𝕊)
12:         for all c ∈ C_k do
13:             if getSupport(c) ≥ MinSup then
14:                 F_k := F_k ∪ {c}
15:             end if
16:         end for
17:         if k = 1 then
18:             C_2 := generateCandidates(C_1)
19:         else
20:             C_{k+1} := generateCandidates(F_k)
21:         end if
22:         k := k+1
23:     until C_k = ∅
24:     return F_{1...k}
25: end procedure
```

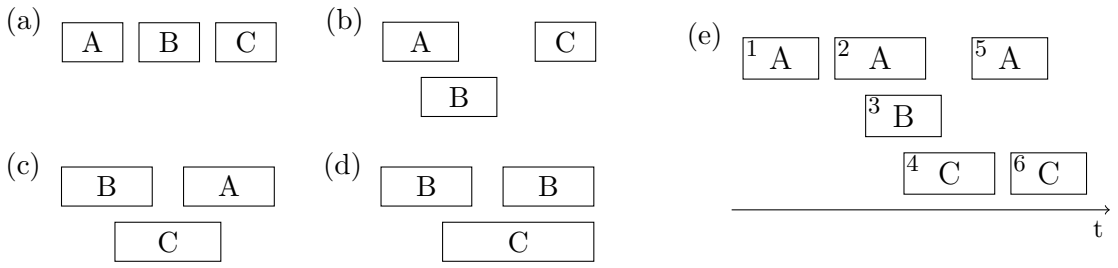


Abbildung 4.22: (a) – (d) vier Kandidaten der Größe 3 (e) eine Intervallsequenz

0	1	2	3	4	5	6
A_1	$A_1(1)$	$A_1(2)$	$A_3(3)$	$A_3(3,4)$	$A_1(5)$	$\mathbf{A_1(1,3,6)}$
A_2	$A_2(1)$	$A_2(2)$	$A_4(3)$	$A_4(3,4)$	$A_2(5)$	$\mathbf{A_2(2,3,6)}$
A_3			$A_1(1,3)$		$\mathbf{A_3(3,4,5)}$	
A_4			$A_2(2,3)$			

Tabelle 4.4: Menge der endlichen Automaten von *FSMSet* für die Kandidaten aus Abbildung 4.22 und die Intervallsequenz in Abbildung 4.22e. Jede Spalte führt die neuen Automaten der entsprechenden Iteration von *FSMSet* auf.

tierbare temporale Intervall. *FSMTree* ist eine Weiterentwicklung von *FSMSet*, die mit Hilfe einer effizienten Datenstruktur die Anzahl der benötigten Automaten verringert.

Zur Illustration dient das folgende Beispiel zu *FSMSet*. Abbildung 4.22 zeigt vier temporale Muster der Größe 3. Diese Muster seien Kandidaten, deren Support anhand der Intervallsequenz in Abbildung 4.22e bestimmt werden soll.

Die Supportevaluation von *FSMSet* beginnt mit dem Ableiten der endlichen Automaten aus den Kandidaten. Anschließend überprüft *FSMSet*, welche Automaten das erste temporale Intervall der Intervallsequenz akzeptieren können. Diese Automaten werden kopiert. Den kopierten Automaten übergibt *FSMSet* das erste temporale Intervall zum Akzeptieren und fügt sie der Menge der Automaten hinzu, bevor das zweite temporale Intervall der Sequenz abgearbeitet wird. Tabelle 4.4 gibt zum obigen Beispiel die Menge der Automaten wieder. Die i-te Spalte in Tabelle 4.4 zeigt die Automaten, die durch die i-te Iterationen von *FSMSet* über den temporalen Intervallen der Intervallsequenz hinzugefügt wurden. Hierbei entspricht A_1 (A_2, A_3 und A_4) dem endlichen Automaten zum Kandidaten aus Abbildung 4.22a (4.22b, 4.22c und 4.22d). Die übrigen Einträge zeigen die bereits vom Automaten akzeptierten Intervalle an. So ist z.B. $A_4(3,4)$ ein Automat des Musters aus Abbildung 4.22d, der bereits das 3-te und 4-te temporale Intervall der Intervallsequenz akzeptiert hat. Dieser Automat ist aus dem Automaten $A_4(3)$ entstanden und während der 4-ten Iteration über der Intervallsequenz hinzugefügt worden. Fettgedruckt sind Automaten, die ihren finalen akzeptierten Zustand erreicht haben. Insgesamt muss *FSMSet* 19 Automaten anlegen, um die Intervallsequenz abzuarbeiten und die drei Vorkommen der temporalen Muster zu finden.

Bei näherer Betrachtung der Automaten in Tabelle 4.4 zeigt sich, dass viele der Automaten ein ähnliches Verhalten besitzen. So akzeptieren z.B. die beiden Automaten A_3 und A_4 bis

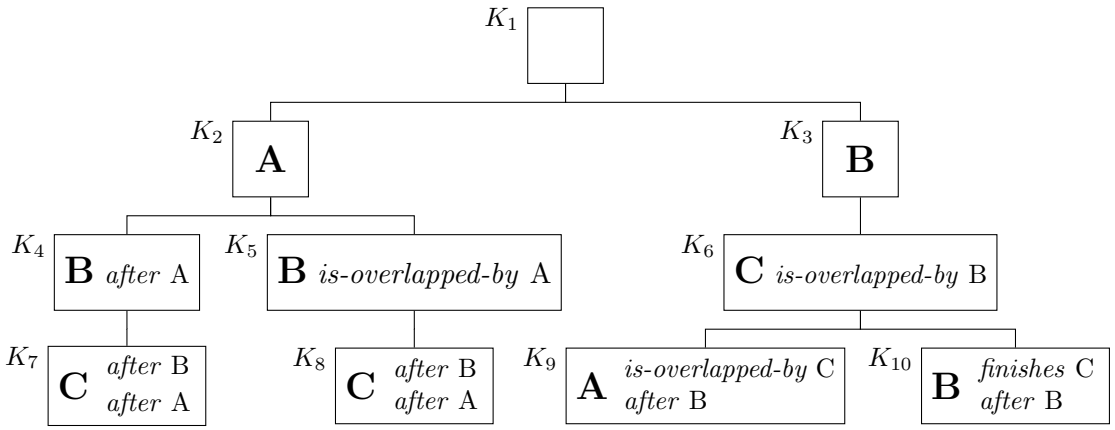


Abbildung 4.23: Präfixbaum der endlichen Automaten basierend auf den Mustern aus Abbildung 4.22

zur vierten Iteration von *FSMSet* die gleichen temporalen Intervalle ($A_3(3,4)$ und $A_4(3,4)$ in Spalte 4). Erst in der fünften Iteration unterscheidet sich das Verhalten der beiden Automaten, da das entsprechende temporale Intervall nur von A_3 akzeptiert werden kann. Das gemeinsame Verhalten der beiden Automaten erklärt sich durch die zugrunde liegenden Muster. Beide Muster verfügen über das gemeinsame 2-Präfix *B overlaps C*. Erst nachdem das gemeinsame 2-Präfix abgearbeitet ist, können die Automaten unterschiedlich agieren. Eine Idee, den algorithmischen Aufwand von *FSMSet* zu minimieren, besteht daher darin, alle Automaten mit einem gemeinsamen Präfix zusammenzufassen. Bereits in [Masseglia u. a., 1998] oder [Pei u. a., 2001] sind Muster anhand ihrer Präfixe zusammengefasst worden. Die dabei entstandene Baumstruktur ist vergleichbar mit dem im Folgenden beschriebenen Präfixbaum für die Automaten von temporalen Mustern.

Abbildung 4.23 zeigt den Präfixbaum der endlichen Automaten aus dem obigen Beispiel.[9] Der Wurzelknoten (K_1) entspricht dem Anfangszustand, über den jeder Automat verfügt. K_1 besitzt zwei Kindknoten (K_2 und K_3). Der Übergang von K_1 nach K_2 erfolgt durch Akzeptieren eines temporalen Intervalls mit dem Label A. Alle Automaten mit dem gemeinsamen Präfix A sind daher in K_2 zusammengefasst. K_2 besitzt wiederum die beiden Kindknoten K_4 und K_5. Der Übergang von K_2 nach K_4 kann nur erfolgen, wenn es ein temporales Intervall mit dem Label B gibt, das zum bereits durch K_2 akzeptierten A in der Relation *after* steht. Im Knoten K_4 sind daher alle Automaten mit dem Präfix *A before B* zusammengefasst (im obigen Beispiel ist dies allerdings nur ein einziges Muster). Analog erfolgt der Übergang vom Knoten K_4 nach K_7 durch ein temporales Intervall mit dem Label C, das zu allen vorherig akzeptierten temporalen Intervallen in den angegebenen Relationen steht (*C after B* und *C after A*). Da alle Kandidaten die gleiche Größe besitzen, ist jeder Zweig des Präfixbaumes gleich lang. Jedes Blatt des Baumes entspricht einem finalen akzeptierten Zustand eines Automaten.

Mit Hilfe des Präfixbaumes kann ein neuer Algorithmus für die Supportevaluation der Kandidaten formuliert werden — *FSMTree*. Anstelle der Menge von Automaten verwaltet *FSMTree*

[9] Der Präfixbaum aus endlichen Automaten ist der Namensgeber des zweiten Algorithmus — *FSMTree*.

0	1	2	3	4	5	6
K_1	$K_2(1)$	$K_2(2)$	$K_3(3)$	$K_6(3,4)$	$K_2(5)$	$\mathbf{K_7(1,3,6)}$
			$K_4(1,3)$		$\mathbf{K_9(3,4,5)}$	$\mathbf{K_8(2,3,6)}$
			$K_5(2,3)$			

Tabelle 4.5: Menge der Knoten von *FSMTree* für den Präfixbaum aus Abbildung 4.23 und die Intervallsequenz in Abbildung 4.22e. Jede Spalte führt die neuen Knoten der entsprechenden Iteration von *FSMTree* auf.

eine Menge von Knoten aus dem Präfixbaum. Als Beispiel soll zunächst die Bestimmung des Supports für die Muster des Präfixbaumes aus Abbildung 4.23 anhand der Intervallsequenz aus Abbildung 4.22e erfolgen. Im ersten Schritt enthält die Menge lediglich den Wurzelknoten K_1. K_1 kann das erste temporale Intervall A der Sequenz akzeptieren. Um alle Vorkommen eines Musters zu finden, müssen wie bei *FSMSet* alle Automaten (Knoten), die ein temporales Intervall akzeptieren können, kopiert werden. Der Kopie von K_1 übergibt *FSMTree* das temporale Intervall A als Eingabe. Der so entstandene Knoten $K_2(1)$[10] wird der Menge der Knoten hinzugefügt. Gleichermaßen erweitert sich die Menge der Knoten durch das zweite temporale Intervall der Intervallsequenz um $K_2(2)$. Tabelle 4.5 zeigt an, welche Knoten in jeder Iteration von *FSMTree* hinzugefügt wurden. Offensichtlich benötigt *FSMTree* lediglich 11 Knoten im Gegensatz zu den 19 Automaten von *FSMSet*, um die drei Vorkommen der temporalen Muster zu finden.

Algorithmus 4.4 zeigt die Umsetzung der Supportevaluation von *FSMTree*. *FSMTree* beginnt mit der Erstellung des Präfixbaumes aus der Menge der Kandidaten mit Hilfe der Funktion *createPrefixTree* (Zeile 5). Anschließend folgt eine Schleife (Zeilen 6–19), die über alle Intervallsequenzen des übergebenen Datensatzes $\mathbb{S}$ iteriert. Bevor die einzelnen temporalen Intervalle der aktuellen Intervallsequenz abgearbeitet werden (Schleife über die Zeilen 8–18), initialisiert *FSMTree* die Menge der Knoten (*nodes*) mit dem Wurzelknoten des Präfixbaumes (Zeile 7). Die Hilfsfunktion *processNodes* (Zeile 10) überprüft alle Knoten der Menge *nodes*, ob sie das aktuelle temporale Intervall akzeptieren können. Alle Knoten, die das aktuelle temporale Intervall akzeptieren können, werden kopiert. Anschließend bekommen alle kopierten Knoten das aktuelle temporale Intervall als Eingabe. Die so entstandene Menge der kopierten Knoten bildet den Rückgabewert von *processNodes* und wird der Variable *new-nodes* zugewiesen. *FSMTree* überprüft jeden der neuen Knoten, ob er seinen finalen akzeptierten Zustand erreicht hat (Schleife über die Zeilen 11–17). Ist dies der Fall, so wird das gefundene Vorkommen durch die Hilfsfunktion *putSupport* weiter verarbeitet.[11] Anderenfalls fügt *FSMTree* den Knoten der Menge aller Knoten (*nodes)* hinzu (Zeile 15). Dadurch steht der Knoten für kommende temporale Intervalle der Intervallsequenz zur Verfügung.

Die bereits bei der Supportevaluation von *FSMSet* angesprochene Integration von zeitlichen Rahmenbedingungen lässt sich wiederum leicht durch die Hilfsfunktion *putSupport* realisieren. Auch die Beschränkung der Anzahl der Knoten auf Basis der zeitlichen Bedingungen kann

[10] In Analogie zum Beispiel von *FSMSet* beschreibt $K_2(1)$ den Knoten K_2 des Präfixbaumes, der das erste temporale Intervall der Sequenz akzeptiert hat.

[11] Die Funktion *putSupport* ist identisch zum Pendant in Algorithmus 4.2 Zeile 12. Sie implementiert die Filterfunktion aller Vorkommen zu den minimalen Vorkommen.

Algorithmus 4.4 Supportevaluation der Kandidatenmuster für FSMTree

```
 1: 𝕊: Menge der Intervallsequenzen
 2: C: Menge der Kandidaten
 3:
 4: procedure EVALUATESUPPORT(C, 𝕊)
 5:     fsmtree := createPrefixTree(C)
 6:     for all S ∈ 𝕊 do
 7:         nodes := {getRootNode(fsmtree)}
 8:         for i := 1 ... length(S) do
 9:             (b, e, l) := i-th element of S
10:             new-nodes := processNodes(nodes, (b, e, l))
11:             for all node ∈ new-nodes do
12:                 if node.isFinallyAccepted() then
13:                     putSupport(S, node)
14:                 else
15:                     nodes := nodes ∪ {node}
16:                 end if
17:             end for
18:         end for
19:     end for
20: end procedure
```

direkt auf die Funktion *processNodes* übertragen werden (vgl. Seite 84).

Algorithmus 4.4 (Supportevaluation) bildet zusammen mit den bereits von *FSMSet* bekannten Algorithmen 4.1 (Kandidatengenerierung) und 4.3 (angepasster Apriori-Ansatz) die Algorithmen von *FSMTree*.

4.3.3 Dip

Die beiden Algorithmen *FSMSet* und *FSMTree* suchen nach den häufigen Mustern mit Hilfe des Apriori-Ansatzes, d.h. in Form einer Breitensuche. Wie bereits in Abschnitt 3 (Seite 33ff.) beschrieben wurde, stellt die Tiefensuche eine Alternative zum Apriori-Ansatz dar. Bei der Tiefensuche werden zunächst die Muster eines Zweiges des Hypothesenraums komplett ausgewertet, bevor die Muster benachbarter Zweige verarbeitet werden. Eine Eigenschaft der Tiefensuche besteht daher darin, dass längere Muster eines Zweiges vor den kürzeren Mustern des benachbarten Zweiges gefunden werden. Insbesondere das Beschneiden von Kandidatenmustern, anhand der enthaltenen Teilmuster ist hiervon beeinträchtigt. Der nachfolgend beschriebene Algorithmus *Dip* (*D*eep Search *I*nterval *P*atterns) basiert auf dem Ansatz der Tiefensuche.

Wie bereits bei *FSMTree* soll der Hypothesenraum durch eine Teilmusterbeziehung hierarchisch organisiert werden. *FSMTrees* Supportevaluation orientiert sich dabei an den Präfixen der Kandidatenmuster (siehe Abbildung 4.23). *Dip* nutzt prinzipiell die gleiche hierarchische Ordnung der Muster, allerdings auf Basis der Suffixe. Abbildung 4.24 zeigt ein Beispiel für einen Suffixbaum. Im Gegensatz zu Abbildung 4.23 sind hier die Kandidatenmuster aus Abbildung 4.22 mit Hilfe ihrer Suffixe zusammengefasst.

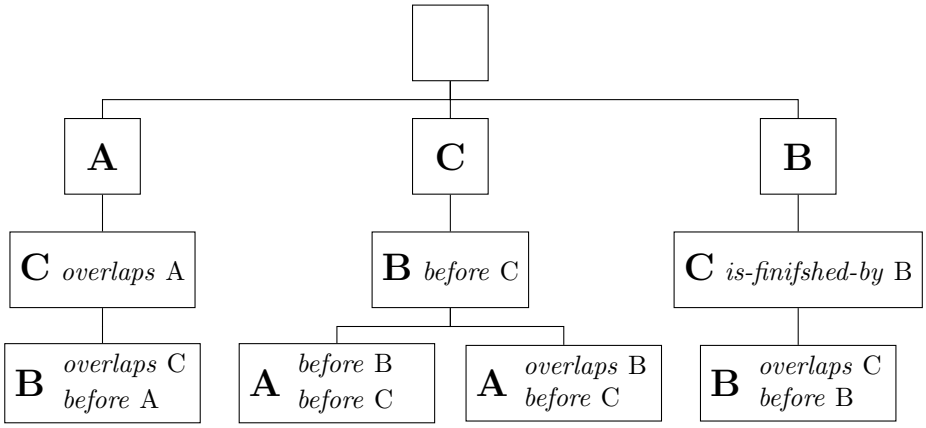
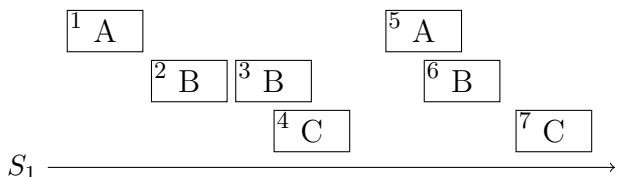


Abbildung 4.24: Suffixbaum zu den vier Kandidaten aus Abbildung 4.22

Jeder Knoten des Suffixbaumes repräsentiert ein temporales Muster, zu dem *Dip* eine Liste von Instanzen speichert. Die Bestimmung des Supports eines Musters erfolgt anschließend nicht durch Auszählen auf den Ausgangsdaten (wie bei *FSMSet* und *FSMTree*), sondern durch eine geeignete Verknüpfung der Instanzlisten. Ein Beispiel für die Verknüpfung von Instanzlisten ist in Abbildung 4.25 gegeben. Abbildung 4.25a zeigt eine Intervallsequenz mit sieben temporalen Intervallen. Die Instanzlisten der temporalen Muster *A before C* und *B before C* sind in den Abbildungen 4.25b und 4.25c dargestellt. Die Instanzlisten liefern die Informationen, aus welchen temporalen Intervallen einer Intervallsequenz eine Instanz des entsprechenden temporalen Musters gebildet werden kann. Der Eintrag (S_1, 1, 4) in Abbildung 4.25b gibt daher an, dass das erste und vierte temporale Intervall der Intervallsequenz S_1 (Abbildung 4.25a) eine Instanz des Musters *A before C* bilden. Die Muster *A before C* und *B before C* besitzen das gemeinsame Suffix *C*. Analog zur Kandidatengenerierung von *FSMSet* (siehe Abbildung 4.19) können die Instanzlisten beider Muster über ihr gemeinsames Suffix verknüpft werden. Die Verknüpfungslogik lässt sich wie folgt durch die Datenbanksprache SQL (siehe z.B. [Heuer u. Saake, 1995]) angeben. Sei P die Instanzliste des temporalen Musters *A before C* und Q die Instanzliste des temporalen Musters *B before C*, so liefert die Abfrage



(a)

sid	l_1	l_2
S_1	1	4
S_1	1	7
S_1	5	7

(b)

sid	l_1	l_2
S_1	2	4
S_1	2	7
S_1	3	7
S_1	6	7

(c)

Abbildung 4.25: (a) eine Intervallsequenz, Instanzlisten der temporalen Muster *A before C* (b) und *B before C* (c)

SELECT P.sid, $P.l_1$, $Q.l_1$, $Q.l_2$
FROM P, Q
WHERE
P.sid = Q.sid *AND*
$P.l_2 = Q.l_2$ *AND*
$P.l_1 < Q.l_1$

die Instanzliste des vorläufigen Musters R, mit $R = P \cup Q$ (vgl. Abbildung 4.19 auf Seite 78). Die Verknüpfung der Instanzlisten erfolgt über den Spalten des gemeinsamen Suffixes sowie über der Sequenzidentifikation (sid). Die zusätzliche Bedingung $P.l_1 < Q.l_1$ verhindert, dass unsortierte Einträge in der resultierenden Instanzliste entstehen. Die Instanzliste und das vorläufige Muster R sind in Abbildung 4.26 dargestellt. Im vorläufigen Muster R ist die

R	A	B	C
A	e	?	b
B	?	e	b
C	a	a	e

(a)

sid	l_1	l_2	l_3
S_1	1	2	4
S_1	1	2	7
S_1	1	3	7
S_1	1	6	7
S_1	5	6	7

(b)

Abbildung 4.26: (a) vorläufiges Muster R (b) und dazugehörige Instanzliste

Intervallrelation zwischen den Labels A und B noch undefiniert. Dementsprechend enthält die Instanzliste aus Abbildung 4.26b sowohl Einträge für die Relation *before* (z.B. (S_1, 1, 2, 4)) als auch für die Relation *overlaps* (S_1, 5, 6, 7). Ein einmaliges Durchlaufen der Instanzliste erlaubt es Instanzlisten für jede vorkommende Intervallrelation in den Daten zu erstellen. Wie bereits bei *FSMSet* und *FSMTree* müssen die minimalen Vorkommen aus der Menge von Instanzen herausgefiltert werden, um den Support eines Musters zu bestimmen.

Dip verbindet die Ideen der Tiefensuche und Verknüpfung von Instanzlisten zu einem Algorithmus. Der Pseudocode von *Dip* ist in Algorithmus 4.5 wiedergegeben. Den Einstiegspunkt bildet die gleichnamige Prozedur in den Zeilen 5–12. Zunächst generiert *Dip* den Wurzelknoten des Suffixbaumes mit Hilfe der Funktion *generate1Patterns* (Zeile 6). Diese Funktion fügt dem Wurzelknoten auch alle temporalen Muster der Größe 1 als Kindknoten hinzu und ermittelt ihre Instanzlisten. Anschließend durchläuft *Dip* alle 1-Muster (Schleife über die Zeilen 7–10) und speichert ihren Support (Zeile 8). Der Kern von *Dip* ist die Hilfsfunktion *findFrequentPatterns* (Zeilen 14–26), die in Zeile 9 mit dem aktuellen Knoten und allen Geschwisterknoten aufgerufen wird.

FindFrequentPatterns ist eine rekursive Prozedur, welche die Tiefensuche von *Dip* realisiert. Die erste Schleife (Zeilen 15–22) iteriert über alle Geschwisterknoten[12] des übergebenen Knotens *node*. Die Instanzlisten von *node* und des aktuellen Geschwisterknotens werden mit Hilfe der Funktion *join* (Zeile 16) verknüpft. Die Verknüpfung erfolgt, wie oben erläutert, auf Basis des gemeinsamen Suffixes. Als Rückgabe liefert *join* die aus der Verknüpfung erzeugten

[12] Die Menge der Geschwisterknoten umfasst in diesem Fall auch den Knoten *node* selbst.

Algorithmus 4.5 Algorithmus *Dip*

```
1:  𝕊: Menge der Intervallsequenzen
2:  MinSup: minimaler Support
3:  n, s: Knoten des Suffixbaums
4:
5:  procedure DIP(𝕊)
6:      root := generate1Patterns(𝕊)
7:      for all n ∈ root.getChildren() do
8:          supportEvaluation(n)
9:          findFrequentPatterns(n, root.getChildren())
10:     end for
11:     return root
12: end procedure
13:
14: procedure FINDFREQUENTPATTERNS(node, siblings)
15:     for all s ∈ siblings do
16:         candidates := join(s.getInstanceList(), node.getInstanceList())
17:         for all c ∈ candidates do
18:             if supportEvaluation(c) ≥ MinSup then
19:                 node.addChild(c)
20:             end if
21:         end for
22:     end for
23:     for all n ∈ node.getChildren() do
24:         findFrequentPatterns(n, node.getChildren())
25:     end for
26: end procedure
```

temporalen Muster sowie ihre Instanzlisten. Jedes so entstandene temporale Muster wird anschließend mit Hilfe der Instanzliste auf seinen Support überprüft (Zeile 18 in der Schleife über die Zeilen 17–21). Genügt der Support der minimalen Supportschranke, so fügt *Dip* das Muster dem Knoten *node* als Kindknoten hinzu (Zeile 19). Nachdem alle Kindknoten zu *node* gefunden wurden, beginnt der rekursive Abstieg über die Kindknoten von *node* (Zeilen 23–25).

Dip zeigt in seiner Vorgehensweise trotz der Tiefensuche und Instanzlisten deutliche Parallelen zu *FSMSet* und *FSMTree* auf. Zum einen wird jedes temporale Muster der Größe $k+1$ durch eine Verknüpfung zweier k-Muster über ein gemeinsames $(k-1)$-Suffix erzeugt. Zum anderen müssen die minimalen Vorkommen in einem separaten Schritt aus den gefundenen Instanzen herausgefiltert werden, um den Support zu bestimmen. Diese beiden Eigenschaften sind auch in den zuvor beschriebenen Algorithmen zu finden. Der Hauptunterschied besteht in der Ermittlung der Instanzen zu einem temporalen Muster. *Dip* benötigt keine zusätzlichen Durchläufe über die Ausgangsdaten, sondern ist in der Lage die Instanzen eines Musters aus den Instanzen der kürzeren Muster abzuleiten. Kapitel 5 testet die unterschiedlichen algorithmischen Ansätze in verschiedenen Experimenten auf ihre Leistungsfähigkeit und stellt sie kritisch einander gegenüber.

4.4 Zusammenfassung

Dieses Kapitel stellte die drei neuen Algorithmen *FSMSet*, *FSMTree* und *Dip* zum Auffinden aller häufigen temporalen Muster aus einer intervallbasierten Datengrundlage vor.

Im ersten Teil des Kapitels wurde zunächst das Problem der Suche nach häufigen temporalen Mustern in Intervallsequenzen formal definiert. Neben der Definition der Datengrundlage, bestehend aus temporalen Intervallen und Intervallsequenzen, gehörten dazu die Wahl einer geeigneten Supportdefinition und Hypothesensprache. In beiden Fällen wurden detailliert die Vor- und Nachteile bekannter Supportdefinitionen und Hypothesensprachen beleuchtet. Auf Grund der Ausdrucksstärke wurden temporale Muster schließlich auf Basis von Allens Intervallrelationen definiert, während der Support auf der Anzahl der minimalen Vorkommen beruht.

Der zweite Teil untersuchte eingehend die Eigenschaften des zuvor definierten Problems. Hier konnte gezeigt werden, dass die Hypothesensprachen der Warenkorb- und Sequenzanalyse Spezialfälle von temporalen Mustern sind. Des Weiteren konnte die Größe der Hypothesenräume der Warenkorb- und Sequenzanalyse bzw. eine obere Schranke für die Größe des Hypothesenraumes temporaler Muster ermittelt werden. Im Mittelpunkt stand die Untersuchung, dass im Allgemeinen das Apriori-Kriterium verletzt ist. Das heißt, ein häufiges temporales Muster kann nichthäufige Teilmuster enthalten. Die Gültigkeit des Apriori-Kriteriums ist von zentraler Bedeutung für alle effizienten Algorithmen, die in Daten häufige Muster suchen. Es konnte jedoch die Gültigkeit des partiellen Apriori-Kriteriums für das definierte Problem nachgewiesen werden: Alle Suffixe eines häufigen temporalen Musters sind häufig.

Im dritten Teil wurden zunächst zwei Algorithmen beschrieben, die das partielle Apriori-Kriterium mit Hilfe eines Schemas aus Kandidatengenerierung und Supportevaluation ausnutzen, um alle häufigen temporalen Muster in den Eingabedaten zu finden. Sowohl *FSMSet* als auch *FSMTree* verwenden endliche Automaten zur Bestimmung des Supports der Muster. *FSMTree* verbessert hierbei den Ansatz von *FSMSet*, indem es mit Hilfe einer effizienteren

Datenstruktur den algorithmischen Aufwand minimiert. Bei der Datenstruktur handelt es sich um einen Präfixbaum, in dem alle Automaten zusammengefasst sind. Der dritte Algorithmus *Dip* stellt einen Gegenentwurf zu *FSMSet* und *FSMTree* dar. Anstelle des Apriori-Ansatzes führt *Dip* eine Tiefensuche durch und findet die Instanzen eines temporalen Musters nicht durch endliche Automaten, sondern durch die Verknüpfung von Instanzlisten.

Alle drei Verfahren zeichnen sich durch folgende Vorteile aus, die in dieser Kombination kein anderes bekanntes Verfahren besitzt:

- Die Eingabedaten können aus einer Vielzahl von Intervallsequenzen bestehen.
- Das mehrfache Vorkommen eines Musters innerhalb einer Intervallsequenz wird bei der Bestimmung des Supports berücksichtigt.
- Der Support eines Musters basiert auf der Anzahl seiner Vorkommen in den Intervallsequenzen.
- Es können zeitliche Rahmenbedingungen für die Gültigkeit eines Vorkommens gesetzt werden.

Insbesondere die verwendete Supportdefinition der minimalen Vorkommen ist hierbei herauszustellen. Zum einen ignorieren die meisten bekannten Verfahren (siehe Abschnitt 3.2) mehrfache Vorkommen innerhalb einer Intervallsequenz, obwohl sie von besonderer praktischer Bedeutung sind. Zum anderen ist der Support für unerfahrene Anwender intuitiv verständlich, da er lediglich die Anzahl der minimalen Vorkommen eines Musters wiedergibt. Daher eröffnen sich den Verfahren viele neue Anwendungsgebiete, die den bekannten Verfahren verschlossen blieben. Kapitel 7 gibt hierzu einige Beispiele aus der Automobilbranche.

Kapitel 5

Evaluation

In diesem Kapitel wird das Verhalten der drei neuen Algorithmen *FSMSet*, *FSMTree* und *Dip* in verschiedenen Experimenten untersucht. Der Fokus liegt hierbei vor allem auf dem Laufzeitverhalten und dem Speicherbedarf der Algorithmen. Für die Auswertung kommen sowohl künstlich generierte Daten als auch reale Anwendungsdaten (vgl. Kapitel 7) zum Einsatz. Des Weiteren wird das Skalierungsverhalten der Verfahren in Abhängigkeit von der Größe der Eingabedaten und die Auswirkungen zeitlicher Randbedingungen untersucht.

5.1 Grundlagen

Für die Evaluation wurden die Verfahren implementiert und Datensätze generiert oder beschafft. Im Folgenden werden diese notwendigen Grundlagen näher beschrieben.

5.1.1 Experimentierumgebung und Implementierung

Alle drei Algorithmen wurden in der Programmiersprache Java implementiert und bzgl. Laufzeit- und Speicherbedarf optimiert. Die Algorithmen sind unter der Prämisse implementiert, dass der gesamte Datensatz im Hauptspeicher analysiert werden kann. Insbesondere greifen die Algorithmen während ihrer Laufzeit nicht auf externe Speichermedien zu (d.h. sie selbst erzeugen keine I/O-Last).

Als Experimentierumgebung kamen drei identische SUN Fire X2100 mit 2.2 GHz und 4 Gigabyte Hauptspeicher zum Einsatz.[1] Die während der Experimente ermittelten Laufzeiten beziehen sich immer auf die tatsächlich notwendige Zeitdauer zum Auffinden aller häufigen temporalen Muster.[2] Das Laden der Eingabedaten in den Hauptspeicher oder das Speichern möglicher Resultate sind nicht Teil der Laufzeitmessungen. In der gemessenen Laufzeit sind aber eventuelle systeminterne Vorgänge (z.B. Auslagern von Hauptspeicherseiten) eingeschlossen.

Die Messung des Speicherbedarfs erfolgt durch regelmäßige Abfrage des von der Java Virtual Machine allokierten Speichers.[3] Der Spitzenwert dieser Folge ergibt den Messwert für den

[1] Die Rechner wurden unter dem Betriebssystem Solaris Version 10 betrieben. Die Java Virtual Machine stand in Version 1.5.0_06-b05 zur Verfügung. Für alle Experimente wurden der Virtual Machine maximal 3 000 Megabyte Hauptspeicher zugewiesen (Parameter -Xmx3000m, vgl. [Sun Microsystems, 2007b]).

[2] Die Zeitdauer wird mit Hilfe zweier Abfragen der Funktion *System.currentTimeMillis()* ermittelt (siehe [Sun Microsystems, 2007a]).

[3] Die Funktion *Runtime.totalMemory()* zeigt den aktuell durch die Java Virtual Machine belegten Speicher an (siehe [Sun Microsystems, 2007a]) und wird im Abstand von 500 ms abgerufen.

Speicherbedarf. Da die regelmäßige Abfrage des Speicherbedarfs die Laufzeit der Algorithmen beeinflusst, wird jedes Experiment zweimal durchgeführt. Der erste Durchlauf dient der Ermittlung der Laufzeit, während der zweite Durchlauf den Speicherbedarf feststellt.

5.1.2 Synthetische Daten

Um die Verfahren auf einer großen Anzahl verschiedener Datensätze testen zu können, wurde ein Datensatzgenerator für künstliche Daten implementiert. Dieser wird nachfolgend näher erläutert.

Der Generator erzeugt unter der Vorgabe gewünschter Datensatzeigenschaften eine Menge von Intervallsequenzen. Zu den Parametern des Datensatzgenerators gehören die Anzahl der Intervallsequenzen (S), die Anzahl der temporalen Intervalle pro Intervallsequenz (I) und die Anzahl der verschiedenen Labels (L) im Datensatz. Der Generator erzeugt solange einzelne Intervallsequenzen, bis die vorgegebene Anzahl S erreicht ist. Dazu legt er zunächst für jedes Label eine binäre Variable an, deren initialer Zustand *falsch* ist. Anschließend wird zu den diskreten Zeitpunkten $\{1, 2, 3, \ldots, \infty\}$ der Zustand der binären Variablen mit einer vorgegebenen Wahrscheinlichkeit[4] p verändert (d.h. aus *falsch* wird *wahr* und umgekehrt). Die Menge der Zeitpunkte, in denen eine Variable durchgehend mit dem Wert *wahr* belegt ist, bildet ein temporales Intervall mit dem entsprechenden Start-, Endzeitpunkt und Label. Die Bearbeitung der Zeitpunkte wird abgebrochen, wenn die vorgegebene Anzahl der temporalen Intervalle pro Sequenz (I) erreicht ist.

Die Parameter mit denen die verschiedenen Datensätze der nachfolgenden Experimente erzeugt wurden, dienen als Nomenklatur für die Bezeichnung der Datensätze. So enthält z.B. der Datensatz S5kI20L15 5 000 Intervallsequenzen mit je 20 temporalen Intervallen von 15 verschiedenen Labels. Tabelle 5.1 gibt einen Überblick über die verwendeten synthetischen Datensätze, sowie den mit ihnen verfolgten Experimentierzielen.

Datensatz	Verwendung
S500I30L10	Laufzeitverhalten, Speicherbedarf, Auswirkung zeitlicher Randbedingungen
S30kI10L5	Laufzeitverhalten, Speicherbedarf
S5kI20L15	Laufzeitverhalten, Speicherbedarf
S1kI20L5	Laufzeitverhalten, Speicherbedarf
S500InL10	Laufzeitverhalten, Speicherbedarf, Skalierbarkeit mit der Datensatzgröße (I) $n = \{20, 25, 30, 35, 40, 45, 50, 55\}$
SnI30L10	Laufzeitverhalten, Speicherbedarf, Skalierbarkeit mit der Datensatzgröße (S) $n = \{100, 200, 400, \ldots 204\,800\}$

Tabelle 5.1: Übersicht über die künstlichen Datensätze in den Experimenten

[4] In Experimenten dieser Arbeit wurde mit einem Wert von $p = 0.3$ gearbeitet.

5.1.3 Anwendungsdaten

Neben den synthetischen Daten wurden auch Daten von drei realen Anwendungen zur Evaluation der Algorithmen herangezogen. Die nachfolgende Übersicht gibt einen Einblick in die Eigenschaften der Datensätze. Eine detaillierte Darstellung der Anwendungen zusammen mit ihren Anforderungen folgt in Kapitel 7.

Datensatz PRODUKTBEWÄHRUNG: Dieser Datensatz beschreibt die Ausstattungsmerkmale und Werkstattaufenthalte von ca. 100 000 Fahrzeugen. Jede Intervallsequenz entspricht einem Fahrzeug und besteht aus 14 bis 48 temporalen Intervallen. Temporale Muster, die für die Anwendung relevant sind, beschreiben häufig auftretende Abfolgen von Reparaturen, welche bei Fahrzeugen mit bestimmten Ausstattungen auftreten. In Abschnitt 7.1 wird näher auf die Anwendung eingegangen.

Datensatz CAN-BUS: Dieser Datensatz besteht aus 85 Aufzeichnungen eines Datenloggers, die während der Durchführung von Erprobungsfahrten erhoben wurden. Zu jeder Aufzeichnung existiert eine Intervallsequenz mit bis zu 139 temporalen Intervallen. Die temporalen Intervalle beschreiben sowohl den Fahrzustand (eingelegter Gang, Drehzahlbereiche, etc.) als auch Fehlermeldungen von elektronischen Steuergeräten. Ziel der Anwendung ist es, Fahrsituationen zu identifizieren, die einen Steuergerätfehler verursachen. Weitere Ausführungen zur Anwendung sind in Abschnitt 7.2 zu finden.

Datensatz CRM: Dieser Datensatz entstammt einer Anwendung des analytischen Customer Relationship Managements. Er enthält Informationen zu über 400 000 Kunden. Zu jedem Kunden gibt es eine Intervallsequenz im Datensatz, die zum einen seine bisherigen Käufe zum anderen aber auch allgemeine Informationen des Kunden (Alter, Geschlecht, Kundenwert, etc.) enthält. In der Anwendung werden häufige temporale Muster gesucht, die das Wiederkaufsverhalten von loyalen Kunden beschreiben. Weitergehende Erläuterungen zu dieser Anwendung liefert Abschnitt 7.3.

Tabelle 5.2 fasst die Eigenschaften der Datensätze zusammen und zeigt an, welche Experimentierziele mit ihnen verfolgt wurden.

Datensatz	S	I	L	Verwendung
PRODUKTBEWÄHRUNG	101 250	14 – 48	345	Laufzeitverhalten, Speicherbedarf
CAN-BUS	85	24 – 139	93	Laufzeitverhalten, Speicherbedarf
CRM	416 393	13 – 103	466	Laufzeitverhalten, Speicherbedarf

Tabelle 5.2: Übersicht über die realen Datensätze in den Experimenten

5.2 Experimente

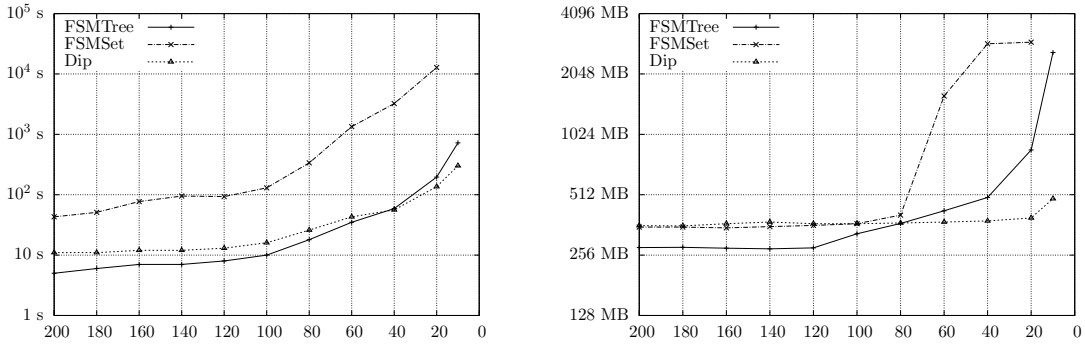
In den nachfolgenden Experimenten werden zunächst das Laufzeitverhalten und der Speicherbedarf der Algorithmen *FSMSet*, *FSMTree* und *Dip* miteinander verglichen. Anschließend folgt

eine Untersuchung der Skalierbarkeit der Algorithmen in Abhängigkeit von der Datensatzgröße. Zum Abschluss wird die Auswirkung zeitlicher Randbedingungen auf das Laufzeitverhalten und den Speicherbedarf analysiert.

5.2.1 Laufzeitverhalten und Speicherbedarf

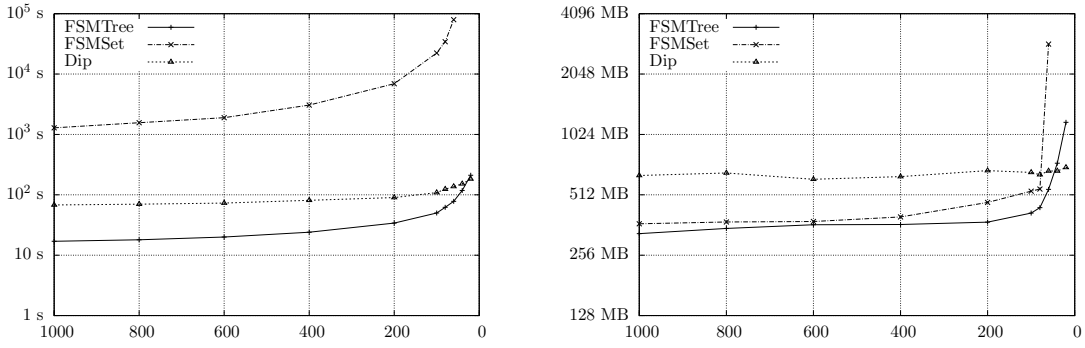
In den ersten Experimenten werden die Algorithmen auf die vier künstlichen Datensätze S500I30L10, S30kI10L5, S5kI20L15 und S1kI20L15 sowie den drei realen Datensätzen PRODUKTBEWÄHRUNG, CAN-BUS und CRM angewendet. Für jeden Datensatz erfolgen mehrere Durchläufe der Verfahren mit variierenden minimalen Supportschranken.

Die Ergebnisse auf den synthetischen Daten sind in den Abbildungen 5.1 bis 5.4 dargestellt.



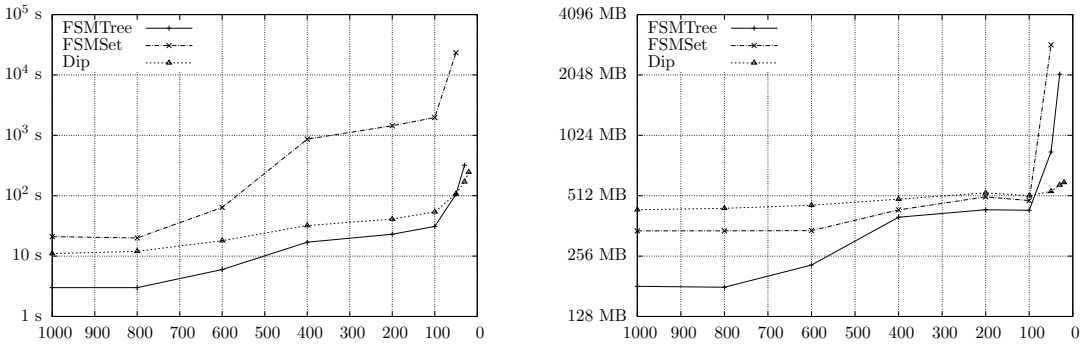
(a) Laufzeit in Sekunden (b) Speicherbedarf in MB

Abbildung 5.1: Effizienz der Verfahren auf S500I30L10 bei variierendem MinSup



(a) Laufzeit in Sekunden (b) Speicherbedarf in MB

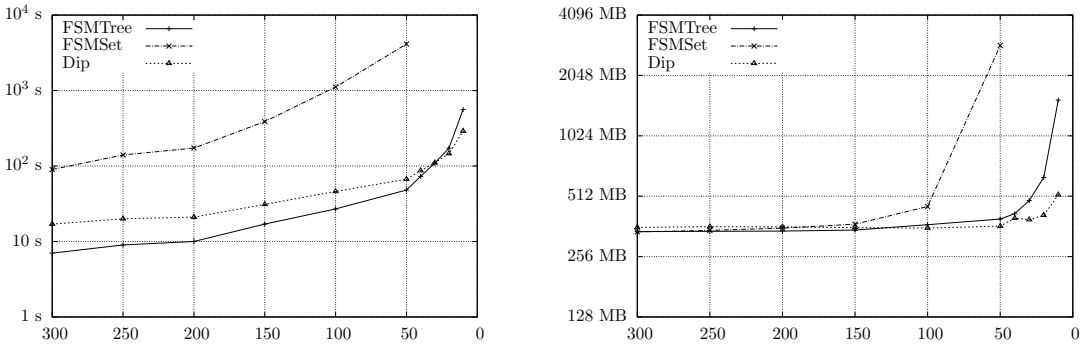
Abbildung 5.2: Effizienz der Verfahren auf S30kI10L5 bei variierendem MinSup



(a) Laufzeit in Sekunden

(b) Speicherbedarf in MB

Abbildung 5.3: Effizienz der Verfahren auf S5kI20L15 bei variierendem MinSup



(a) Laufzeit in Sekunden

(b) Speicherbedarf in MB

Abbildung 5.4: Effizienz der Verfahren auf S1kI20L5 bei variierendem MinSup

Ergebnisse auf synthetischen Daten

Aus den Abbildungen 5.1 bis 5.4 ist ersichtlich, dass für alle drei Algorithmen sowohl die benötigte Laufzeit als auch der Speicherbedarf mit sinkender Supportschwelle stark ansteigt. Dieses Verhalten war zu erwarten, da die Anzahl der häufigen Muster mit sinkender Supportschwelle exponentiell wächst (vgl. Abschnitt 4.2.2).

Des Weiteren bestätigen die Experimente die Erwartungen bzgl. der Algorithmen *FSMSet* und *FSMTree*. *FSMTree* ist eine Weiterentwicklung von *FSMSet*, in der die aus den Kandidaten abgeleiteten endlichen Automaten in einem Präfixbaum zusammengefasst werden. In Abschnitt 4.3.2 wurde angenommen, dass diese effiziente Datenstruktur die Laufzeit von *FSMSet* verbessern müsste, da die Anzahl der benötigten endlichen Automaten verringert wird. Jedes der Experimente auf synthetischen Daten bestätigt diese Annahme. Darüber hinaus ist auch der Speicherbedarf von *FSMSet* im Verlaufe jedes Experiments größer als der von *FSMTree*. Die Laufzeitdifferenzen in den Experimenten zwischen *FSMTree* und *FSMSet* nehmen mit sinkender Supportschwelle zu. Dieser Effekt ist auf die steigende Anzahl häufiger Muster zurückzuführen. Da die Anzahl häufiger Muster mit sinkender Supportschwelle steigt, nimmt ebenfalls die Anzahl der benötigten Kandidaten zu. Je mehr Kandidaten generiert werden, desto größer ist die Anzahl der Kandidaten, die ein gemeinsames Präfix aufweisen. *FSMTree* minimiert den (kumulativen) Aufwand für Kandidaten mit gemeinsamen Präfixen und kann daher bei niedrigen Supportschwellen noch effizienter arbeiten als *FSMSet*.

Das Laufzeitverhalten und der Speicherbedarf von *Dip* unterscheidet sich von *FSMSet* und *FSMTree*. Im Vergleich zu *FSMSet* weist *Dip* immer geringere Laufzeiten auf. Allerdings ist der Speicherbedarf von *Dip* für hohe Supportschranken auf den Datensätzen S30kI10L5n und S5kI20L15 höher als der von *FSMSet*. Bei sehr niedrigen Supportschranken ist der Speicherbedarf von *Dip* aber der geringste von allen Algorithmen. Der Vergleich des Speicherbedarfs von *Dip* mit *FSMTree* zeigt auf, dass *Dip* für hohe Supportschranken zunächst höhere Werte aufweist.[5] Erst der geringere Anstieg des Speicherbedarfs mit sinkender Supportschwelle ermöglichen *Dip* die Werte von *FSMTree* zu unterbieten. Auch das Laufzeitverhalten gleicht diesem Schema. Für hohe Supportschwellen weist *FSMTree* immer die niedrigsten Laufzeiten auf. In allen Experimenten ist aber *Dip* das schnellste Verfahren bei den kleinsten Supportschwellen.

Die Resultate zum Laufzeitverhalten und Speicherbedarf beider Verfahren lassen sich durch die unterschiedlichen algorithmischen Ansätze erklären. Mit sinkender Supportschwelle nimmt die Anzahl der häufigen Muster zu. Dieser Effekt gilt für beide Algorithmen, jedoch muss *FSMTree* zusätzlich die gestiegene Anzahl an Kandidatenmustern verarbeiten. Die Anzahl der Kandidatenmuster wächst ebenfalls exponentiell mit sinkender Supportschwelle, da die Kandidaten eine Obermenge der häufigen Muster sind. *Dip* hingegen verwendet keine Kandidatenmuster und benötigt somit für sie keinen zusätzlichen Hauptspeicher. Die Menge der Kandidaten sind daher als Ursache für den stärkeren Anstieg des Speicherbedarfs von *FSMTree* zu sehen. Die Kandidatenmuster haben nicht nur einen direkten Einfluss auf den Speicherbedarf, sie beeinflussen auch die Laufzeit von *FSMTree*. Für jeden Kandidaten legt *FSMTree* einen endlichen Automaten an, welcher die temporalen Intervalle der Intervallsequenzen verarbeitet. Auch wenn sich ein Muster als nicht-häufig erweist, so hat sein Automat algorithmischen Aufwand verursacht. Für die kleinsten Supportschwellen in den obigen Experimenten übertrifft schließ-

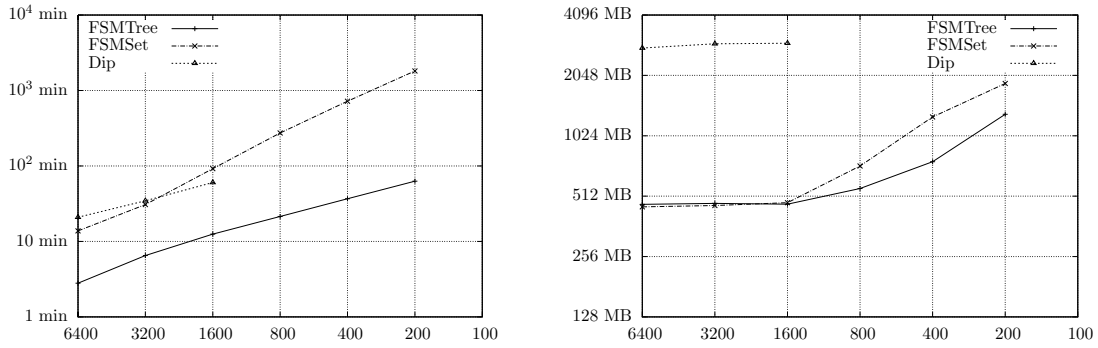
[5] Eine Ausnahme bildet der Datensatz S1kI20L5, bei dem alle Algorithmen anfänglich einen ähnlichen Speicherbedarf aufweisen

lich der erhöhte Aufwand für die Verarbeitung der Kandidaten den algorithmischen Aufwand von *Dip*.

Die Abbildungen 5.5 bis 5.7 zeigen die Resultate der Experimente auf den Anwendungsdatensätzen.

Ergebnisse auf den Anwendungsdaten

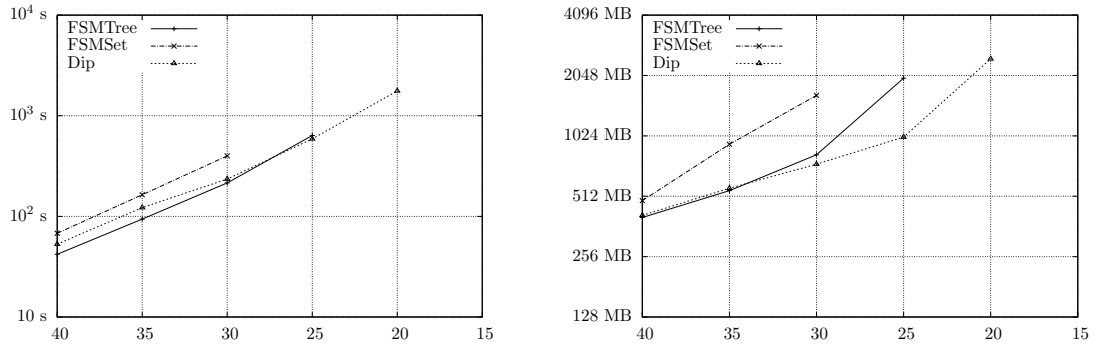
In Bezug auf die Algorithmen *FSMSet* und *FSMTree* bestätigen die Experimente auf den realen Daten die Resultate der synthetischen Daten. In jedem Experiment und bei jeder Supportschwelle weist *FSMTree* einen geringeren Speicherbedarf und eine geringere Laufzeit auf als *FSMSet*. *FSMTree* dominiert *FSMSet* vollständig.



(a) Laufzeit in Minuten

(b) Speicherbedarf in MB

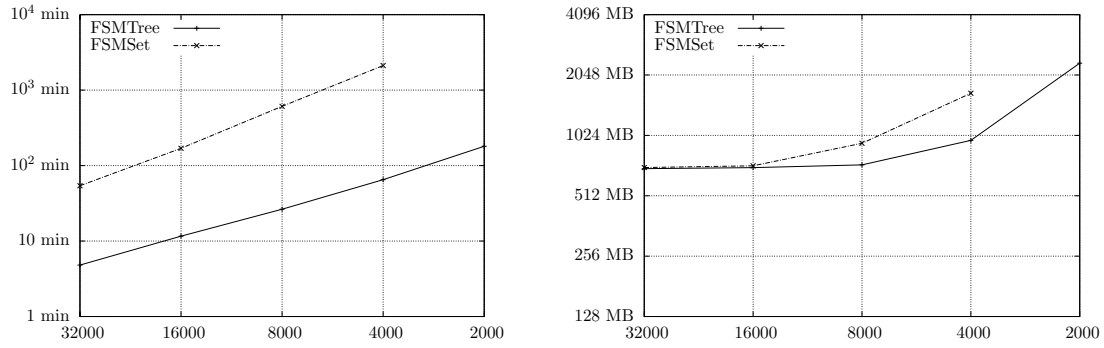
Abbildung 5.5: Effizienz der Verfahren auf PRODUKTBEWÄHRUNG bei variierendem MinSup



(a) Laufzeit in Sekunden

(b) Speicherbedarf in MB

Abbildung 5.6: Effizienz der Verfahren auf CAN-BUS bei variierendem MinSup



(a) Laufzeit in Minuten

(b) Speicherbedarf in MB

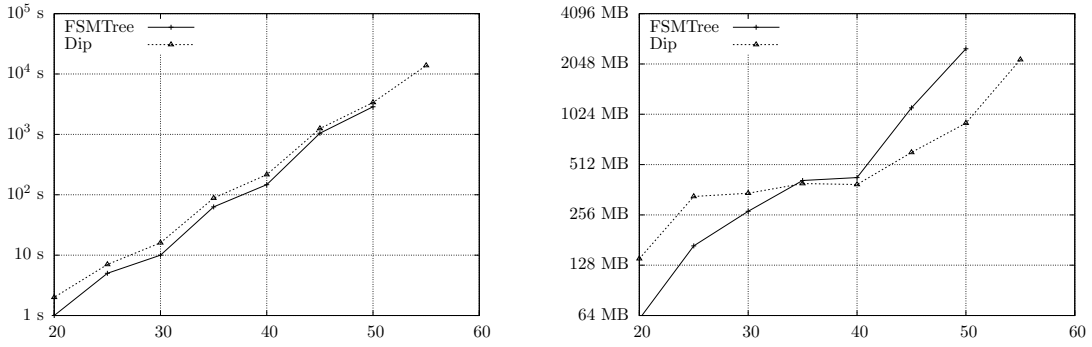
Abbildung 5.7: Effizienz der Verfahren auf CRM bei variierendem MinSup

Im Einklang mit den vorherigen Resultaten stehen auch die Ergebnisse der Experimente zum Datensatz CAN-BUS (Abbildung 5.6). Für hohe Supportschwellen benötigt *FSMTree* die geringsten Laufzeiten. Erst bei geringeren Supportschwellen kann *Dip* die Laufzeiten von *FSMTree* unterbieten. Auch der geringere Speicherbedarf von *Dip* bei niedrigen Supportschranken wird in diesem Experiment bestätigt. Während *FSMTree* mit dem gegebenen Speicher nur Supportschranken bis 25 (*FSMSet* sogar nur bis 30) bearbeiten kann, findet *Dip* noch für die Schranke 20 alle häufigen Muster.

Die Begrenzung des verfügbaren Hauptspeichers ist maßgeblich für die Experimente auf den Datensätzen PRODUKTBEWÄHRUNG und CRM. Wie aus Abbildung 5.5 hervorgeht, kann *Dip* nur für die ersten drei Supportschranken die häufigen Muster identifizieren. Dabei arbeitet *Dip* bereits an der Schranke des für die Java Virtual Machine bereitgestellten Speichers von 3 000 Megabyte. Es ist daher anzunehmen, dass die dabei erzielten Laufzeiten bereits stark durch die Arbeit des Java Garbage Collectors beeinflusst sind [Sun Microsystems, 2007c]. Dennoch unterbietet *Dip* die Laufzeit von *FSMSet* für die Supportschwelle 1 600. Auf dem Datensatz CRM konnte *Dip* keine Ergebnisse erzielen, da der zur Verfügung stehende Hauptspeicher nicht ausreichte. Die Datensätze PRODUKTBEWÄHRUNG und CRM sind die beiden größten Datensätze der bisherigen Experimente ($S > 100\,000$). Dieser Fakt legt die Vermutung nahe, dass die Algorithmen unterschiedlich gut mit zunehmender Datensatzgröße skalieren. Im folgenden Abschnitt werden die Skalierungseigenschaften der Algorithmen gesondert untersucht.

5.2.2 Skalierung mit der Datensatzgröße

Die Größe eines Datensatzes hat zwei Dimensionen. Die erste Dimension besteht aus der (durchschnittlichen) Anzahl von temporalen Intervallen pro Intervallsequenz. Die zweite Dimension betrifft die Anzahl der Intervallsequenzen im Datensatz. Für beide Dimensionen werden im Folgenden Experimente durchgeführt, die das Skalierungsverhalten der Algorithmen aufzeigen sollen. Da aus den bisherigen Experimenten hervorging, dass *FSMTree FSMSet* bzgl. der Laufzeit und des Speicherbedarfs dominiert, werden nur die Algorithmen *FSMTree* und *Dip* berücksichtigt.



(a) Laufzeit in Sekunden

(b) Speicherbedarf in MB

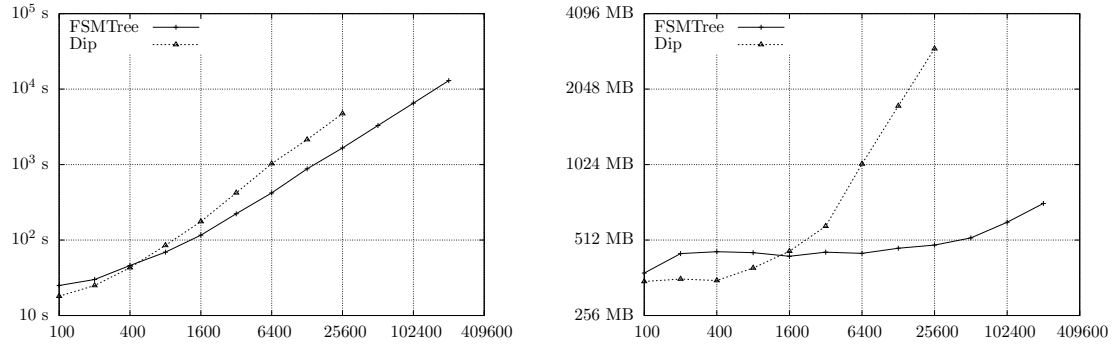
Abbildung 5.8: Effizienz der Verfahren bei steigender Anzahl von temporalen Intervallen pro Intervallsequenz auf S500InL10

Steigende Anzahl temporaler Intervalle

Im ersten Experiment soll das Skalierungsverhalten mit steigender Anzahl von temporalen Intervallen pro Intervallsequenz (I) untersucht werden. Dazu wurden mit Hilfe des Datensatzgenerators verschiedene Datensätze nach dem Schema S500InL10 erzeugt. Die Werte für die Anzahl der temporalen Intervalle sind $n = \{20, 25, 30, \ldots, 55\}$. Die Ergebnisse der Experimente für den minimalen Support 100 sind in Abbildung 5.8 dargestellt.

Beide Verfahren weisen einen exponentiellen Anstieg der Laufzeit mit zunehmender Anzahl temporaler Intervalle auf. Die Laufzeiten von *FSMTree* liegen geringfügig unter den Laufzeiten von *Dip*. Bei geringem I benötigt *FSMTree* weniger Speicher als *Dip*. Mit steigender Anzahl temporaler Intervalle ist der Speicherzuwachs, den *FSMTree* benötigt, deutlich größer als der von *Dip*. Für den Wert $n = 50$ benötigt *FSMTree* sogar fast den gesamten verfügbaren Hauptspeicher, während *Dip* mit weniger als einem Drittel davon auskommt.

Es sind zwei Ursachen für den rapiden Anstieg des Speicherbedarfs von *FSMTree* zu nennen. Zum einen bewirkt die feste Supportschwelle, dass mit steigender Anzahl der temporalen Intervalle auch die Anzahl der häufigen Muster zunimmt. Aus den Experimenten des vorherigen Abschnitts ist bereits bekannt, dass *Dip* mit steigender Anzahl häufiger Muster (bzw. sinkender Supportschwelle) weniger zusätzlichen Speicher benötigt als *FSMTree*. Die zweite Ursache ist in der Vervielfältigung der endlichen Automaten von *FSMTree* zu sehen. Immer wenn der endliche Automat eines Kandidatenmusters ein temporales Intervall akzeptieren kann, wird zunächst eine Kopie des Automaten angelegt. Nur einem der beiden Automaten wird anschließend das temporale Intervall zum Akzeptieren übergeben (vgl. Seite 81). Zwar garantiert diese Vorgehensweise, dass alle Vorkommen eines temporalen Musters gefunden werden, jedoch stellt sie große Ansprüche an den verfügbaren Hauptspeicher. Mit jedem temporalen Intervall verdoppelt sich die Anzahl der Automaten, die es akzeptieren können. Da alle Automaten auch für spätere temporale Intervalle der Intervallsequenz verwendet werden, ist ein exponentieller Anstieg des Speicherbedarfs mit wachsender Anzahl temporaler Intervalle pro Intervallsequenz die Folge. Im Vergleich dazu skaliert *Dip* besser mit steigender Anzahl temporaler Intervalle.



(a) Laufzeit in Sekunden

(b) Speicherbedarf in MB

Abbildung 5.9: Effizienz der Verfahren bei steigender Anzahl von Intervallsequenzen auf SnI30L10 (MinSup = 10% der Anzahl der Intervallsequenzen)

Zwar benötigt auch *Dip* mehr Hauptspeicher, jedoch beschränkt sich dieser auf mehr Einträge in den Instanzlisten.

Steigende Anzahl von Intervallsequenzen

Das nächste Experiment betrachtet die Skalierbarkeit der Algorithmen bei steigender Anzahl der Intervallsequenzen im Datensatz. Dazu wurden mit Hilfe des Datensatzgenerators mehrere Datensätze des Schemas SnI30L10 generiert. Die Anzahl der Intervallsequenzen für die verschiedenen Datensätze beträgt $n = \{100,\ 200,\ 400,\ 800, 1\,600,\ 3\,200,\ 6\,400,\ 12\,800,\ 25\,600,\ 51\,200, 102\,400, 204\,800\}$. Aus den Experimenten in Abschnitt 5.2.1 ist bereits bekannt, dass *Dip* bei steigender Anzahl häufiger Muster (bzw. sinkendem MinSup) Vorteile gegenüber *FSMTree* hat. Damit dieser bekannte Effekt die Untersuchung der Skalierbarkeit in S nicht beeinträchtigt, wurde die minimale Supportschwelle (MinSup) für jeden Datensatz auf 10% der Anzahl seiner Intervallsequenzen festgelegt. Durch die relative Bestimmung von MinSup bleibt die Anzahl der häufigen Muster über die einzelnen Datensätze hinweg konstant.[6] Die Ergebnisse des Experiments sind in Abbildung 5.9 dargestellt.

Beide Algorithmen skalieren bzgl. der Laufzeit linear mit der Anzahl der Intervallsequenzen. Wobei *Dip* nur auf den ersten beiden Datensätzen die Laufzeit von *FSMTree* unterbieten kann. Beim Speicherbedarf unterscheiden sich beide Verfahren deutlich. *Dip* benötigt auf den ersten vier Datensätzen weniger Speicher als *FSMTree*. Ab einer Datensatzgröße von 1 600 Intervallsequenzen zeigt sich ein linearer Zusammenhang zum benötigten Speicher. Bei 25 600 Intervallsequenzen ist der zur Verfügung stehende Speicher verbraucht. Im Gegensatz dazu zeigt *FSMTree* bis zu 25 600 Intervallsequenzen einen nahezu konstanten Speicherbedarf von ca. 480 Megabyte. Erst danach steigt der Speicherbedarf an. Mit einem Bedarf von ca. 720 Megabyte für 204 800 Intervallsequenzen ist der Anstieg aber deutlich geringer als bei *Dip*.

[6] Mit konstant ist hier konstant innerhalb einer Epsilonumgebung gemeint. Kleine Schwankungen in der Anzahl der häufigen Muster sind durch den Zufallsprozess, mit dem die Datensätze generiert werden, nicht zu vermeiden.

Die höhere Speichereffizienz von *FSMTree* lässt sich mit Hilfe des Apriori-Ansatzes erklären. *FSMTree* benötigt für die Kandidatengenerierung und die Supportevaluation Speicher. Für die Kandidaten (und die Kandidatengenerierung) benötigt *FSMTree* für alle Datensätze gleich viel Speicher, da sowohl die Anzahl der Kandidaten als auch die Anzahl der häufigen Muster konstant bleibt (durch den relativ zur Sequenzanzahl festgelegten Supportschwellwert). Auch der Speicherbedarf für die Supportevaluation ist unabhängig von der Anzahl der Intervallsequenzen. Für die Supportevaluation verbraucht *FSMTree* Speicher, um alle benötigten endlichen Automaten verwalten zu können. Die Menge der endlichen Automaten (bzw. Knoten aus dem Präfixbaum vgl. Abschnitt 4.3.2) wächst, je mehr temporale Intervalle eine Intervallsequenz hat. Nachdem eine Intervallsequenz abgearbeitet ist, kann der durch die Automaten belegte Speicher für die nächste Intervallsequenz wiederverwendet werden. Da alle Intervallsequenzen im Experiment gleich lang sind, kann demzufolge der Speicherbedarf für die Supportevaluation als konstant angesehen werden. Der Speicherbedarf von *FSMTree* ist daher unabhängig von der Anzahl der Intervallsequenzen im Datensatz. Der beobachtete Anstieg im Speicherbedarf für viele Intervallsequenzen erklärt sich durch den höheren Speicherbedarf der Daten selbst. Für die Experimente werden die Daten komplett im Hauptspeicher geladen und analysiert. Die von der Speichermessmethode gelieferten Werte, beinhalten daher auch die Größe des Datensatzes. Im Gegensatz zu *FSMTree* benötigt *Dip* mit steigender Sequenzanzahl mehr Speicher. Die Erklärung hierfür liefern die von *Dip* verwendeten Instanzlisten. Zwar bleibt die Anzahl der häufigen Muster über die Datensätze hinweg konstant, jedoch steigt die Anzahl der Instanzen, die ein häufiges Muster hat, mit jeder weiteren Intervallsequenz. Daraus folgt, dass die Instanzlisten mit jedem Datensatz länger werden und somit mehr Speicher benötigen.

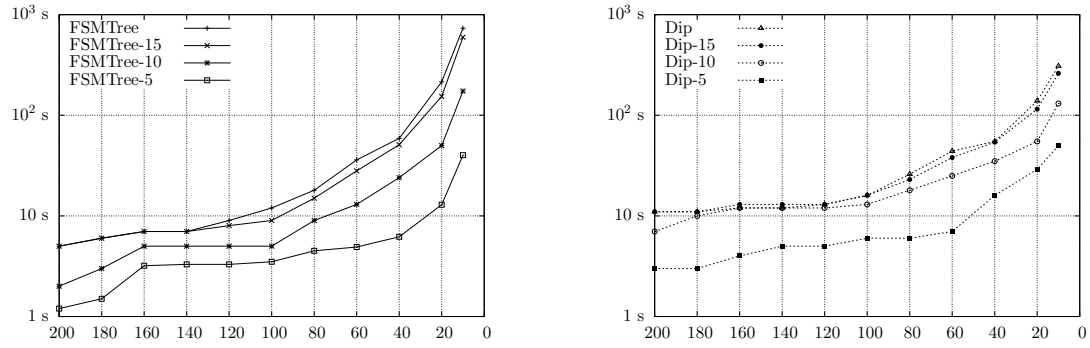
Zusammenfassend ist festzustellen, dass beide Verfahren bzgl. der Laufzeit linear mit der Anzahl der Intervallsequenzen skalieren. Im Bezug auf den Speicherbedarf ist jedoch der Algorithmus *FSMTree* im Vorteil, da sein Speicherbedarf unabhängig von der Sequenzanzahl ist. Des Weiteren erklärt dieses Ergebnis, warum *Dip* auf den Datensätzen PRODUKTBEWÄHRUNG und CRM in Folge zu hohen Speicherbedarfs keine (oder keine vollständigen) Resultate erzielen konnte (siehe Abschnitt 5.2.1).

5.2.3 Beachtung zeitlicher Randbedingungen

In diesem Abschnitt wird die Auswirkung zeitlicher Randbedingungen auf die Laufzeit und den Speicherbedarf der Algorithmen untersucht. Eine zeitliche Randbedingung verlangt, dass nur die Instanzen zum Support eines Musters beitragen, deren minimale Vorkommen eine vorgegebene Zeitdauer nicht überschreiten (vgl. Abschnitt 4.3.1 auf Seite 84).

Die Algorithmen *FSMTree* und *Dip* gehen unterschiedlich mit einer gegebenen zeitlichen Bedingung um. *FSMTree* nutzt die zeitlichen Randbedingungen, um frühzeitig die Menge der endlichen Automaten während der Supportevaluation einzuschränken. Dazu überprüft jeder Automat, ob das aktuelle temporale Intervall der Intervallsequenz akzeptiert werden kann, ohne dass die Zeitschranke verletzt wird. Ist die zeitliche Bedingung verletzt, so kann der Automat entfernt werden, da spätere Intervalle der Sequenz nur längere minimale Vorkommen erzeugen können (zeitliche Sortierung der Intervallsequenzen). Auf Grund der reduzierten Anzahl benötigter Automaten sollte diese Vorgehensweise dazu führen, dass *FSMTree* bei Vorgabe zeitlicher Randbedingungen sowohl kürzere Laufzeiten erzielt als auch weniger Speicher benötigt.

Auch *Dip* kann von zeitlichen Randbedingungen profitieren. Nachdem die Instanzliste eines



(a) Laufzeit in Sekunden

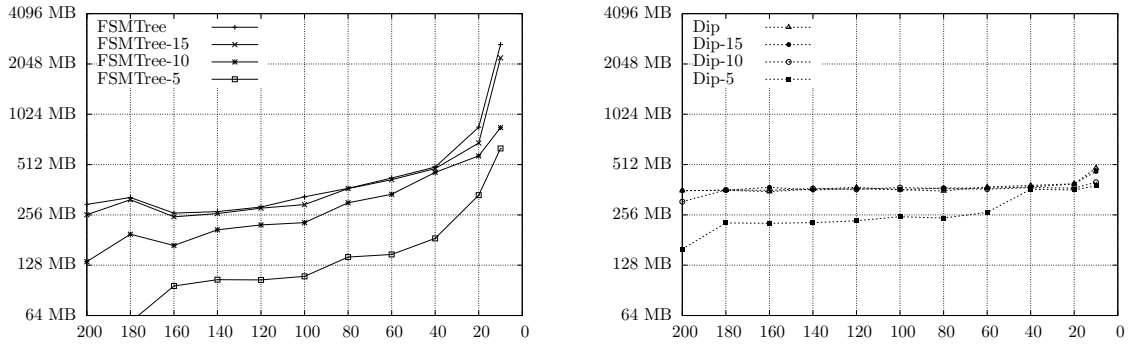
(b) Laufzeit in Sekunden

Abbildung 5.10: Effizienz der Verfahren bei unterschiedlichen zeitlichen Randbedingungen und variierendem MinSup. Abbildung (a) für *FSMTree* und (b) für *Dip*.

temporalen Musters erzeugt wurde, entfernt *Dip* alle Instanzen aus der Liste, deren minimale Vorkommen der zeitlichen Bedingung nicht genügen. Dieser Filterschritt erzeugt zunächst zusätzlichen algorithmischen Aufwand. Das Ergebnis ist aber eine (potentiell) kürzere Instanzliste, die zum einen weniger Speicher verbraucht und zum anderen weniger Aufwand bei der Verknüpfung mit anderen Instanzlisten verursacht (zur Erzeugung längerer temporaler Muster, vgl. Abschnitt 4.3.3).

Im nachfolgenden Experiment wurde wiederum mit dem Datensatz S500I30L10 und variierendem minimalen Support gearbeitet. Für jede Supportschwelle wurden die Verfahren jeweils viermal gestartet. Der erste Durchlauf enthält keine zeitlichen Randbedingungen. In den folgenden Durchläufen ist die maximale Zeitdauer, die eine Instanz aufweisen darf, auf 15, 10 und 5 Zeiteinheiten gesetzt. Die Abbildungen 5.10 und 5.11 veranschaulichen die Ergebnisse bzgl. Laufzeit und Speicherbedarf.

Beide Algorithmen erzielen kürzere Laufzeiten mit kleiner werdender Zeitschranke. Das Experiment bestätigt somit die Annahme, dass beide Algorithmen von der Existenz zeitlicher Randbedingungen profitieren können. Abbildung 5.10 zeigt jedoch auch, dass *FSMTree* eine stärkere Reduktion der Laufzeiten insbesondere bei niedrigen Supportschranken aufweist. Ohne zeitliche Bedingung erzielt *Dip* auf den drei kleinsten Supportschranken die kürzeren Laufzeiten (vgl. auch Abschnitt 5.2.1 Abbildung 5.1). Für die stärkste zeitliche Randbedingung (maximale Zeitdauer der minimalen Vorkommen = 5) aber ist *FSMTree* das schnellste Verfahren für alle Supportschwellen. Auch im Bezug auf den Hauptspeicher können beide Verfahren ihren Bedarf mit stärkeren zeitlichen Randbedingungen reduzieren. Wie jedoch Abbildung 5.11 zeigt, ist die Reduktion für *Dip* bei den Zeitschranken 15 und 10 vernachlässigbar. Erst bei der Schranke 5 ist ein deutlicher Rückgang des Speicherbedarfs zu erkennen (Supportschwellen 200 – 60). Im Gegensatz dazu kann wiederum *FSMTree* mehr von den zeitlichen Randbedingungen profitieren. Bereits für die Zeitschranke 10 ist eine deutliche Reduktion des Speicherbedarfs erkennbar. Des Weiteren übertrifft der Speicherbedarf von *FSMTree* für die Schranke 5 nur auf der kleinsten Supportschwelle den Speicherbedarf von *Dip*. Ohne zeitliche Randbedingungen ist dies für die Supportschwellen 60, 40, 20 und 10 der Fall gewesen.



(a) Speicherbedarf in MB

(b) Speicherbedarf in MB

Abbildung 5.11: Effizienz der Verfahren bei unterschiedlichen zeitlichen Randbedingungen und variierendem MinSup. Abbildung (a) für *FSMTree* und (b) für *Dip*.

5.3 Zusammenfassung

In diesem Kapitel wurden die Eigenschaften der Algorithmen *FSMSet*, *FSMTree* und *Dip* mit Hilfe verschiedener Datensätze untersucht. Für die beiden Apriori basierten Verfahren zeigte sich, dass *FSMTree* sowohl im Speicherbedarf als auch im Bezug auf die Laufzeit *FSMSet* überlegen ist. Die Ursache für die Dominanz *FSMTrees* ist die effizientere Organisation der endlichen Automaten mit Hilfe eines Präfixbaumes.

Ein differenzierteres Bild ergibt sich für die Algorithmen *FSMTree* und *Dip*. Die Experimente zeigten auf, dass in Abhängigkeit vom gewählten Datensatz und der Supportschwelle sowohl *FSMTree* als auch *Dip* Vorteile bzgl. Speicherbedarf und Laufzeit haben können. Auf den meisten Datensätzen war *FSMTree* bei hohen Supportschwellen *Dip* überlegen (Laufzeit und Speicher). Für sehr kleine Supportschwellen hingegen war *Dip* das schnellere und weniger speicherbedürftige Verfahren. Die verschiedenen Lösungsstrategien der beiden Algorithmen, erklären die beobachteten Ergebnisse. Je geringer die gewählte Supportschranke ist, desto größer ist die Anzahl der Kandidatenmuster, die durch den Apriori-Ansatz von *FSMTree* erzeugt werden. Kandidaten benötigen nicht nur zusätzlichen Speicher, sie verursachen auch zusätzlichen Aufwand während der Supportevaluation. Die Tiefensuche von *Dip* hingegen kennt keine Kandidatenmuster und orientiert sich an den Instanzlisten der kürzeren häufigen Muster. Mit sinkender Supportschwelle besitzt daher der datenorientierte Ansatz von *Dip* Vorteile.

Die Experimente mit verschieden großen Datensätzen wiesen nach, dass die Tiefensuche mit Hilfe von Instanzlisten schlechter mit zunehmender Anzahl an Intervallsequenzen im Datensatz skaliert als der Apriori-Ansatz von *FSMTree*. Die Laufzeiten beider Verfahren skalieren linear mit der Anzahl der Intervallsequenzen. Im Gegensatz zu *Dip* besitzt jedoch *FSMTree* einen nahezu konstanten Speicherbedarf. Jede neue Intervallsequenz erzeugt zusätzliche Einträge in den Instanzlisten von *Dip* und verursacht somit einen höheren Speicherbedarf. *FSMTree* hingegen benötigt Speicher zur Kandidatengenerierung und Supportevaluation. Dieser Speicherbedarf ist unabhängig von der Sequenzanzahl und erklärt, warum für sehr große Datensätze nur *FSMTree* einsetzbar ist.

Zuletzt wurde die Auswirkung zeitlicher Randbedingungen auf die Verfahren untersucht. Beide Algorithmen verwenden die Randbedingungen, um frühzeitig die Menge der Kandidaten bzw. die Instanzlisten zu beschneiden. Sowohl *FSMTree* als auch *Dip* können daher von der Existenz zeitlicher Randbedingungen profitieren und erzielen kürzere Laufzeiten und benötigen weniger Speicher.

Kapitel 6

Ergänzende Verfahren zu temporalen Mustern

Mit den in Kapitel 4 beschriebenen Algorithmen können die häufigen temporalen Muster aus einem Datensatz extrahiert werden. Zur Lösung eines gegebenen Anwendungsproblems sind oft weiterführende Verarbeitungsschritte notwendig. In diesem Kapitel werden zum einen die Ableitung temporaler Regeln und ihre Bewertung (Abschnitt 6.1) und zum anderen eine geeignete Visualisierung temporaler Muster und Regeln (Abschnitt 6.2) diskutiert. Darüber hinaus zeigt Abschnitt 6.3, wie temporale und statische (nicht zeitbezogene) Daten in einer gemeinsamen Analyse kombiniert werden können.

6.1 Temporale Regeln

Die Entdeckung häufiger temporaler Muster ist zumeist nur der erste Schritt, um in einem Datensatz zeitliche Zusammenhänge zu finden. In Analogie zur Warenkorbanalyse kann aus den häufigen temporalen Mustern eine Menge von Regeln abgeleitet werden (vgl. Abschnitt 2.2.1). Die gefundenen Regeln erlauben eine quantitative Bewertung der temporalen Zusammenhänge und ermöglichen die einfache Adressierung von Data Mining-Aufgaben wie Klassifikation, Prognose, Abhängigkeitserkennung oder Abweichungsanalyse. Nachfolgend werden *temporale Regeln* formal definiert. Die verwendeten Begriffe sind an die Terminologie aus [Höppner, 2002a, Seite 48] angelehnt.

Definition 6.1 *(temporale Regeln) Ist X ein echtes Teilmuster des temporalen Musters Y ($X \subset Y$), so wird die Regel $X \Rightarrow Y$ als temporale Regel bezeichnet. Das temporale Muster X heißt Prämissenmuster, während Y Regelmuster genannt wird. Durch Entfernung des Prämissenmusters aus dem Regelmuster entsteht das Konklusionsmuster.*

Abbildung 6.1 zeigt ein Beispiel für eine temporale Regel. Das Prämissenmuster *A overlaps B* ist ein Teilmuster des Regelmusters. Das Konklusionsmuster besteht lediglich aus dem Label *C*. Wie bei Assoziationsregeln erschließt sich die Semantik temporaler Regeln durch eine *Wenn-Dann* Aussage. Das Beispiel aus Abbildung 6.1 kann daher wie folgt formuliert werden: *Wenn*

	A	B
A	e	o
B	io	e

$\Longrightarrow$

	A	B	C
A	e	o	b
B	io	e	m
C	a	im	e

Abbildung 6.1: Beispiel einer temporalen Regel

A overlaps B gilt, *dann* existiert ein *C*, das zu *B* in der Relation *is-met-by* und zu *A* in der Relation *after* steht.

Die Erzeugung temporaler Regeln orientiert sich an der Gewinnung von Assoziationsregeln aus häufigen Warenkörben (vgl. Abschnitt 2.2.1 und [Agrawal u. a., 1996]). Aus jedem häufigen temporalen Muster kann eine Menge von Regeln abgeleitet werden. Dazu wird jedes Teilmuster des häufigen Musters genutzt, um eine Regel nach Definition 6.1 zu bilden. Das häufige Muster entspricht dann dem Regelmuster, während das Teilmuster das Prämissenmuster bildet. Im Kontext einer praktischen Anwendung kann die Menge der Regeln oft eingeschränkt werden. Relevante Einschränkungen bestehen z.B. darin, dass nur Regeln mit einem bestimmten Konklusionsmuster oder Regeln mit einer Aussage für zukünftige Ereignisse betrachtet werden müssen (siehe Anwendungsbeispiele in Kapitel 7).

Trotz ihrer Gemeinsamkeiten weisen Assoziationsregeln und temporale Regeln entscheidende Unterschiede auf. In einer Assoziationsregel $A \Rightarrow B$ kennzeichnen A und B disjunkte Warenkörbe ($A \cap B = \emptyset$). Die Schnittmenge zwischen A und B ist leer, da die Semantik von Assoziationsregeln nur eine mögliche Relation zwischen den Items aus A und B erlaubt — alle Items sind in derselben Transaktion.[1] In einer temporalen Regel hingegen können die Labels des Konklusionsmusters in verschiedenen Relationen zu den Labels des Prämissenmusters stehen. Würde im Beispiel aus Abbildung 6.1 die Konsequenz der temporalen Regel allein aus dem Label *C* bestehen, so könnte nicht aufgelöst werden, ob *B* zu *C* in der Relation *meets*, *overlaps*, *finishes*, usw. steht. Daher muss in einer temporalen Regel die Prämisse ein Teilmuster der Konsequenz sein, um die Intervallrelationen zwischen den Labels des Konklusionsmusters und des Prämissenmusters vollständig erklären zu können.

6.1.1 Bewertende Gütemaße

Für Assoziationsregeln existieren eine Vielzahl von bewertenden Gütemaßen, mit denen starke Abhängigkeiten zwischen einzelnen Items identifiziert werden können. Zu den häufig verwendeten Maßen gehören z.B. Lift, Gini-Index oder J-measure. Eine detaillierte Auflistung verschiedener Gütemaße, sowie eine Untersuchung ihrer Eigenschaften ist in [Tan u. a., 2002] zu finden.[2] Die einfachsten Maße für Assoziationsregeln sind der Support und die Konfidenz (vgl. Abschnitt 2.2.1). Den etablierten Pfaden der Assoziationsregeln folgend, können Support und Konfidenz direkt auf temporale Regeln übertragen werden. Für eine temporale Regel $X \Rightarrow Y$ beschreibt der Support, wie oft sie auf den Ausgangsdaten korrekt anwendbar ist.

$$\mathrm{Sup}(X \Rightarrow Y) = \mathrm{Sup}(Y)$$

Die Konfidenz hingegen stellt das Verhältnis, wie oft die Regel korrekt anwendbar ist, zur Gesamtanzahl ihrer Anwendbarkeit dar (nur die Prämisse ist gültig).

$$\mathrm{Conf}(X \Rightarrow Y) = \frac{\mathrm{Sup}(Y)}{\mathrm{Sup}(X)}$$

[1] Die Assoziationsregel $A \Rightarrow B$ kann formuliert werden als: *Wenn* eine Transaktion die Items aus A enthält, *dann* enthält sie auch die Items aus B.

[2] Weitere Untersuchungen zu Gütemaßen für Assoziationsregeln geben z.B. [Sahar, 1999; Shah u. a., 1999; Tan u. Kumar, 2000; McGarry, 2005].

Trotz der Analogie zu Assoziationsregeln unterscheidet sich die Semantik von Support und Konfidenz bei temporalen Regeln. Der Support einer Assoziationsregel ist relativ zur Größe der Transaktionsdatenbank definiert. Er kann als die Wahrscheinlichkeit interpretiert werden, dass die Assoziationsregel für eine zufällig gezogene Transaktion der Datenbank gültig ist. Bei temporalen Regeln hingegen ist diese wahrscheinlichkeitsbasierte Interpretation nicht durchführbar, da der Support die Anzahl der minimalen Vorkommen des Regelmusters angibt. Die wahrscheinlichkeitsbasierte Interpretation lässt sich auch nicht durch das Verhältnis des Supports zur Anzahl der Intervallsequenzen in den Ausgangsdaten ($\frac{\mathrm{Sup}(Y)}{N}$) wiederherstellen. Durch die gewählte Supportdefinition werden alle minimalen Vorkommen eines Musters innerhalb einer Intervallsequenz berücksichtigt. Das Verhältnis zur Anzahl der Intervallsequenzen liefert daher die durchschnittliche Anzahl eines temporalen Musters pro Intervallsequenz. Dieser Mittelwert kann Werte größer 1 annehmen und ist nicht als eine Wahrscheinlichkeit zu verstehen.[3]

Die Konfidenz einer Assoziationsregel $A \Rightarrow B$ kann ebenfalls als Wahrscheinlichkeit gedeutet werden. Sie gibt an, mit welcher Wahrscheinlichkeit eine Transaktion neben den Items aus A auch die Items aus B enthält. Für eine temporale Regel $X \Rightarrow Y$ beschreibt die Konfidenz entsprechend die Wahrscheinlichkeit, dass eine Instanz von X in den Ausgangsdaten zu einer Instanz von Y erweitert werden kann. Diese Interpretation der Konfidenz für temporale Regeln ist nur gerechtfertigt, wenn der Wertebereich der Konfidenz auf das Intervall $[0, 1]$ begrenzt ist. Aus der Definition der Konfidenz folgt, dass Werte größer 1 möglich sind, falls $\mathrm{Sup}(Y) > \mathrm{Sup}(X)$ gilt. Da X ein Teilmuster von Y ist, kann dieser Fall nur auftreten, wenn Y ein Contains-Muster und X das 1-Präfix von Y ist (vgl. Satz 4.36 auf Seite 77). Für Contains-Muster ist die wahrscheinlichkeitsbasierte Interpretation daher nicht gültig. Die Konfidenz zeigt in diesem Fall an, wie oft im Durchschnitt eine Instanz von X in den Ausgangsdaten zu einer Instanz von Y erweitert werden kann.

Komplexere Maße als Support und Konfidenz für Assoziationsregeln werden allerdings nicht allein auf Basis des Supports von Prämisse und Konsequenz berechnet. Vielmehr wird die Information berücksichtigt, wie oft die Prämisse (oder die Konsequenz) nicht in einer Transaktion vorkommt. Der Ausgangspunkt für alle Gütemaße einer Assoziationsregel $A \Rightarrow B$ ist eine 2×2-Kontingenztafel. Sie enthält alle Häufigkeiten von A und B sowie die Häufigkeit ihres Nichtauftretens $\overline{A}$ und $\overline{B}$ (allein und in jeder Kombination AB, $A\overline{B}$, $\overline{A}B$ und $\overline{A}\overline{B}$). Abbildung 6.2 zeigt den Aufbau der Kontingenztafel. Da der Support eines Warenkorbs relativ zur Größe der Transaktionsdatenbank definiert ist (Definition 2.3), kann $|\overline{A}|$ als Differenz $1 - \mathrm{Sup}(A)$ angegeben werden. Analog ergeben sich die anderen Einträge der Kontingenztafel durch Differenzenbildung. Insgesamt sind lediglich die Werte für $\mathrm{Sup}(A)$, $\mathrm{Sup}(B)$ und $\mathrm{Sup}(A \cup B)$ notwendig, um die gesamte Kontingenztafel auszufüllen.

Es ist erforderlich Kontingenztafeln für temporale Regeln aufzustellen, um beliebige Gütemaße von Assoziationsregeln übertragen zu können. Abbildung 6.3 zeigt die Kontingenztafel der temporalen Regel $X \Rightarrow Y$. Die Spalten der Kontingenztafel beschreiben das von der temporalen Regel vorhergesagte Muster. Im Gegensatz zu Assoziationsregeln muss das Konklusionsmuster erst gewonnen werden, indem das Prämissenmuster aus dem Regelmuster entfernt wird (angegeben durch $Y \backslash X$). Die letzten Einträge der ersten Zeile und ersten Spalte geben die Häufigkeit

[3] Bei Assoziationsregeln bleiben mehrfache Vorkommen eines Warenkorbs in einer Transaktion unberücksichtigt, so dass sich ein Wertebereich des Supports von $[0, 1]$ ergibt und eine wahrscheinlichkeitsbasierte Interpretation ermöglicht wird.

	B	$\overline{B}$	
A	$\lvert AB\rvert$	$\lvert A\overline{B}\rvert$	$\lvert A\rvert$
$\overline{A}$	$\lvert \overline{A}B\rvert$	$\lvert \overline{A}\overline{B}\rvert$	$\lvert \overline{A}\rvert$
	$\lvert B\rvert$	$\lvert \overline{B}\rvert$	$\sum = 1$

$$|A| = \mathrm{Sup}(A) \qquad |AB| = \mathrm{Sup}(A \cup B)$$
$$|B| = \mathrm{Sup}(B) \qquad |A\overline{B}| = \mathrm{Sup}(A) - \mathrm{Sup}(A \cup B)$$
$$|\overline{A}| = 1 - \mathrm{Sup}(A) \qquad |\overline{A}B| = \mathrm{Sup}(B) - \mathrm{Sup}(A \cup B)$$
$$|\overline{B}| = 1 - \mathrm{Sup}(B) \qquad |\overline{A}\overline{B}| = |\overline{B}| - |A\overline{B}| = |\overline{A}| - |\overline{A}B|$$

Abbildung 6.2: Kontingenztafel für die Assoziationsregel $A \Rightarrow B$

	$Y \backslash X$	$\overline{Y \backslash X}$	
X	$\mathrm{Sup}(Y)$	$\mathrm{Sup}(X) - \mathrm{Sup}(Y)$	$\mathrm{Sup}(X)$
$\overline{X}$	$\mathrm{Sup}(Y \backslash X) - \mathrm{Sup}(Y)$	-	-
	$\mathrm{Sup}(Y \backslash X)$	-	-

Abbildung 6.3: Kontingenztafel der temporalen Regel $X \Rightarrow Y$

des Prämissenmusters und des Konklusionsmusters an ($\mathrm{Sup}(X)$ und $\mathrm{Sup}(Y \backslash X)$). Der erste Eintrag in der Kontingenztafel enthält die Häufigkeit, wie oft Prämissen- und Konklusionsmuster zusammen auftraten. Diese Information ist durch $\mathrm{Sup}(Y)$ gegeben, da X ein Teilmuster von Y ist. Der zweite Eintrag in der ersten Zeile (Spalte) beschreibt, wie oft das Prämissenmuster (das Konklusionsmuster) vorkam, ohne ein Teil des Regelmusters zu sein. Beide Einträge können aus der Differenz zwischen $\mathrm{Sup}(X)$ und $\mathrm{Sup}(Y)$ bzw. zwischen $\mathrm{Sup}(Y \backslash X)$ und $\mathrm{Sup}(Y)$ abgeleitet werden.[4] Die restlichen Einträge der Kontingenztafel sind nicht ausfüllbar, ohne semantische Widersprüche einzuführen. Für Assoziationsregeln gibt $\mathrm{Sup}(\overline{A})$ den Anteil der Transaktionen der Datenbasis an, die X nicht enthalten. Wie bereits angeführt, gilt $\mathrm{Sup}(\overline{A}) = 1 - \mathrm{Sup}(A)$, da ein Warenkorb maximal einmal in einer Transaktion vorkommen kann. Die Übertragung dieser Berechnungsvorschrift auf temporale Regeln ergibt $\mathrm{Sup}(\overline{X}) = N - \mathrm{Sup}(X)$, wobei N der Anzahl der Intervallsequenzen entspricht. Auf Grund der Berücksichtigung mehrfacher Vorkommen eines Musters innerhalb einer Intervallsequenz, kann diese Berechnungsvorschrift zu negativen Werten für $\mathrm{Sup}(\overline{X})$ führen. Dazu muss lediglich das Muster X in jeder Intervallsequenz des Datensatzes mehrfach vorkommen.[5] Der Support muss jedoch eine nichtnegative Funktion sein, so dass eine direkte Übertragung der Berechnungsvorschrift von Assoziationsregeln nicht möglich ist. Auch die Idee $\mathrm{Sup}(\overline{X})$ als die Anzahl der Intervallsequenzen in denen X nicht vorkommt zu definieren, erzeugt semantische Inkonsistenzen. Zum einen würde $\overline{X}$ nur maximal einmal pro Intervallsequenz gezählt, obwohl X hingegen mehrfach in einer Intervallsequenz berücksichtigt wird. Zum anderen kann keine Kontingenztafel aufgestellt werden, da im Allgemeinen nicht mehr $\mathrm{Sup}(X) + \mathrm{Sup}(\overline{X}) = \mathrm{Sup}(Y \backslash X) + \mathrm{Sup}(\overline{Y \backslash X})$ gilt.

Auf Grund der dargelegten Probleme konnte keine semantisch konsistente Kontingenztafel für temporale Regeln aufgestellt werden, die den Ansprüchen der Supportdefinition minimaler

[4] Die Berechnung $\mathrm{Sup}(X) - \mathrm{Sup}(Y)$ ist wiederum nur möglich, wenn Y kein Contains-Muster und X nicht das 1-Präfix von Y ist. Ansonsten können auf Grund von $\mathrm{Sup}(X) < \mathrm{Sup}(Y)$ negative Einträge in der Kontingenztafel entstehen ($\mathrm{Sup}(Y \backslash X) - \mathrm{Sup}(Y)$ analog).

[5] Es folgt $\mathrm{Sup}(X) > N$ und daher $\mathrm{Sup}(\overline{X}) = N - \mathrm{Sup}(X) < 0$.

Vorkommen genügt. Daher sind z.B. folgende Gütemaße für temporale Regeln nicht ableitbar:[6]

- ϕ-Koeffizient,
- Goodman-Kruskal (λ),
- Odds ratio (α),
- Yule's Q,
- Yule's Y,
- Kappa (κ),
- J-measure (J),
- Gini index (G),
- Conviction (V),
- Collective strength (S).

Die Suche nach einem geeigneten Gütemaß für temporale Regeln wird daher im Folgenden auf die bekannten Werte für $\mathrm{Sup}(X)$, $\mathrm{Sup}(Y)$ und $\mathrm{Sup}(Y \backslash X)$ beschränkt.

Die Konfidenz ist als alleiniger Indikator für die Güte einer Regel oft ungeeignet. Eine hohe Konfidenz bedeutet nicht, dass es eine starke Abhängigkeit zwischen dem Prämissenmuster und dem Konklusionsmuster gibt. Die hohe Konfidenz kann auch durch einen hohen Support des Konklusionsmusters begründet werden. Zur Bewertung einer Regel ist es daher sinnvoll die Konfidenz ins Verhältnis zum Support des Konklusionsmusters zu setzen.[7] Sei N die Anzahl der Intervallsequenzen in den Ausgangsdaten. Dann liefert

$$\mathrm{Avg}(Y \backslash X) = \frac{\mathrm{Sup}(Y \backslash X)}{N}$$

die durchschnittliche Anzahl des Konklusionsmusters pro Intervallsequenz. Für das Verhältnis aus Konfidenz und $\mathrm{Avg}(Y \backslash X)$ folgt

$$\mathrm{Lift}(X \Rightarrow Y) = \frac{\mathrm{Conf}(X \Rightarrow Y)}{\mathrm{Avg}(Y \backslash X)} = \frac{\mathrm{Sup}(Y)N}{\mathrm{Sup}(X)\mathrm{Sup}(Y \backslash X)}. \tag{6.1}$$

Das Vorbild für das Gütemaß Lift aus Gleichung 6.1 ist das gleichnamige Gütemaß für Assoziationsregeln ($\mathrm{Lift}(A \Rightarrow B) = \frac{\mathrm{Conf}(A \Rightarrow B)}{\mathrm{Sup}(B)}$). Für Assoziationsregeln beschreibt der Lift den Faktor, um den sich die Wahrscheinlichkeit erhöht, dass eine Transaktion die Items aus B enthält, wenn bekannt ist, dass die Transaktion bereits die Items aus A beinhaltet. Im Allgemeinen kann diese Interpretation nicht auf temporale Regeln übertragen werden, da $\mathrm{Avg}(Y \backslash X)$ keine Wahrscheinlichkeit darstellt. Dennoch drücken temporale Regeln mit hohen Liftwerten eher Abhängigkeiten zwischen Prämissen- und Konklusionsmuster aus als Regeln mit niedrigen Liftwerten. Daher eignet sich der Lift, um eine Rangfolge für eine Menge von temporalen Regeln zu erstellen. Die Erstellung einer Rangfolge hilft dem Anwender schneller die für die Anwendung wichtigen Regeln zu identifizieren (vgl. [Blumenstock u. a., 2006]).

[6] Zur Berechnung fehlen die Werte für $\mathrm{Sup}(\overline{X})$, $\mathrm{Sup}(\overline{Y \backslash X})$, $\mathrm{Sup}(\overline{X} \wedge Y \backslash X)$, $\mathrm{Sup}(\overline{Y \backslash X} \wedge X)$ oder $\mathrm{Sup}(\overline{X} \wedge \overline{Y \backslash X})$. Die Berechnungsvorschriften der einzelnen Maße sind in [Tan u. a., 2002] zu finden.

[7] Häufig auftretende Konklusionsmuster werden dadurch abgewertet.

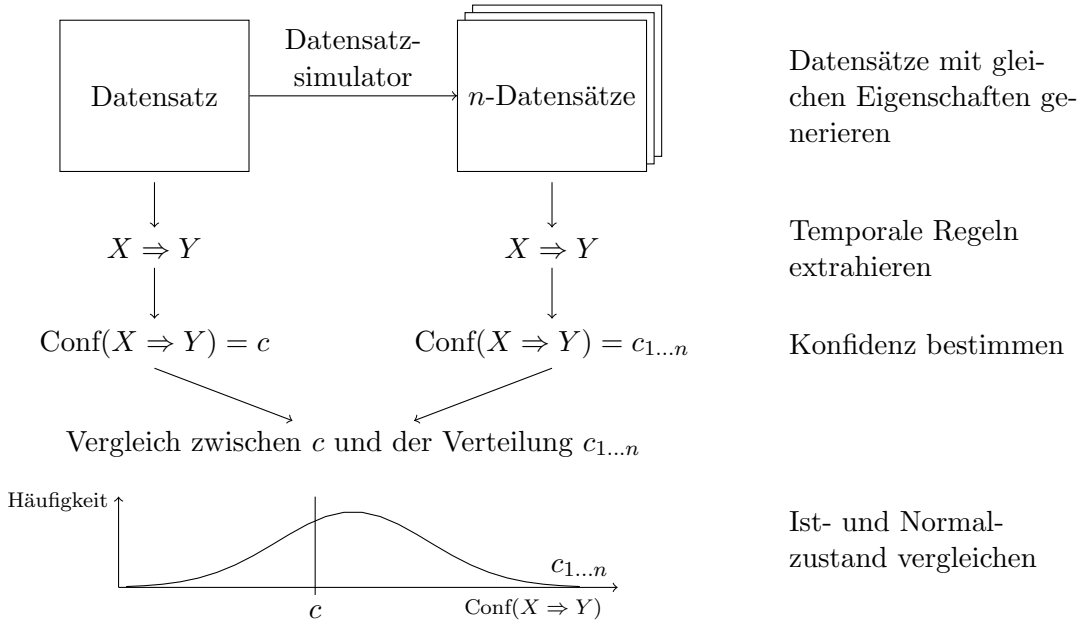


Abbildung 6.4: Simulationsansatz zur Bewertung temporaler Regeln

6.1.2 Bewertung durch Simulation

Eine grundlegende Vorgehensweise der Gütemaße für Assoziationsregeln besteht darin, die (statistische) Abhängigkeit zwischen Prämisse und Konklusion zu quantifizieren. Dazu wird zunächst die Unabhängigkeit zwischen Prämisse und Konklusion angenommen und anschließend die beobachtete Abweichung von der Unabhängigkeitsannahme anhand der Kontingenztafel bewertet. Die Idee, einen Normalzustand (Unabhängigkeitsannahme) mit dem Istzustand (Kontingenztafel) zu vergleichen, ist mit Hilfe des im Folgenden beschriebenen Simulationsansatzes auf temporale Regeln übertragbar.

Wie bereits angeführt, kann die Konfidenz einer temporalen Regel als die Wahrscheinlichkeit verstanden werden, dass eine Instanz des Prämissenmusters zu einer Instanz des Regelmusters erweiterbar ist. Die Konfidenz bewertet somit die Korrektheit der Regel. Eine hohe Konfidenz kann jedoch das Ergebnis von Eigenschaften der Ausgangsdaten sein, ohne dass eine Abhängigkeit zwischen Prämisse und Konklusion besteht.[8] Um zu bewerten, ob eine beobachtete Konfidenz durch eine Abhängigkeit und nicht durch Eigenschaften des Datensatzes entstanden ist, muss sie mit einem Normalzustand verglichen werden. Abbildung 6.4 verdeutlicht, wie der Normalzustand anhand eines Simulationsansatzes gewonnen werden kann. Mit Hilfe eines Datensatzsimulators wird eine Vielzahl von Datensätzen erzeugt, die dieselben Eigenschaften aufweisen wie der ursprüngliche Datensatz. Die zu berücksichtigenden Eigenschaften betreffen dabei die Anzahl und Länge der Intervallsequenzen sowie die Anzahl, Länge und Labels der einzelnen temporalen Intervalle im Ausgangsdatensatz. Zu jedem der generierten Datensätze

[8] Z.B. hat die temporale Regel $A \Rightarrow A$ *before* B bereits dann eine hohe Konfidenz, wenn B oft in den Ausgangsdaten vorkommt.

können die häufigen Muster identifiziert und die temporalen Regeln extrahiert werden. Die Verteilung der Konfidenzwerte einer temporalen Regel $X \Rightarrow Y$ auf den generierten Datensätzen lässt sich anschließend mit der beobachteten Konfidenz der Regel auf dem Ausgangsdatensatz vergleichen. Sei n die Anzahl der generierten Datensätze und $c_{1...n}$ die in den Datensätzen beobachten Konfidenzwerte der Regel $X \Rightarrow Y$, so liefert

$$\overline{c} = \frac{1}{n} \sum_{i=1}^{n} c_i$$

den Mittelwert und

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (c_i - \overline{c})^2}$$

die Standardabweichung der Konfidenzwerte $c_{1...n}$. Mit Hilfe des Mittelwertes und der Standardabweichung kann die Abweichung der Konfidenz der Regel $X \Rightarrow Y$ auf dem Ausgangsdatensatz durch

$$A(c, c_{1...n}) = \frac{|c - \overline{c}|}{s}$$

bewertet werden. Offensichtlich gibt $A(c, c_{1...n})$ die Abweichung einer Konfidenz vom Mittelwert als Vielfaches der Standardabweichung an.

Experiment

Die Eignung des Simulationsansatzes zur Bewertung temporaler Regeln soll durch das nachfolgende Experiment untersucht werden. Zunächst wurde dazu ein synthetischer Datensatz mit Hilfe des Datensatzgenerators (vgl. Abschnitt 5.1.2) erzeugt. Dieser Ausgangsdatensatz enthält 50 Intervallsequenzen zu je 30 temporalen Intervallen. Insgesamt gibt es 10 verschiedene Labels im Datensatz. Des Weiteren wurde die Erzeugung des Datensatzes so abgeändert, dass vier verschiedene temporale Regeln bei der Erzeugung berücksichtigt sind. Immer wenn der Datensatzgenerator die Prämisse einer der vier Regeln zufällig erzeugte, so wurde mit einer der Regel zugeordneten Wahrscheinlichkeit auch das Konklusionsmuster eingefügt. Abbildung 6.5 zeigt die implementierten Regeln und ihre Konfidenzwerte.

Der Ausgangsdatensatz diente als Grundlage für den in Abbildung 6.4 dargestellten Simulationsansatz. Mit Hilfe eines Datensatzsimulators wurden 100 weitere Datensätze erzeugt, die alle dieselben Eigenschaften wie der Ausgangsdatensatz aufweisen.[9] Anschließend wurden aus dem Ausgangsdatensatz und den 100 Simulationsdatensätzen alle häufigen Muster extrahiert (MinSup = 10). Die Regelgewinnung aus den gefundenen Mustern beschränkte sich auf die $(k-1)$-Präfixe. Das häufige Muster entspricht dabei dem Regelmuster, während das $(k-1)$-Präfix das Prämissenmuster bildet. Aus jedem häufigen k-Muster wurde daher genau eine temporale Regel abgeleitet. Bei einer maximalen Regelgröße von 3 konnten auf diese Weise zu jedem der Datensätze ca. 3 000 temporale Regeln identifiziert werden.

Die Konfidenz einer temporalen Regel des Ausgangsdatensatzes lässt sich mit den Konfidenzwerten der gleichen Regel auf den simulierten Regeln vergleichen ($\mathrm{A}(c, c_{1...n})$). Tabelle 6.1

[9] Der Datensatzsimulator erzeugt einen neuen Datensatz, indem er jedes temporale Intervall des Ausgangsdatensatz zufällig in einer neuen Menge von Intervallsequenzen verteilt. Die Anzahl und Länge der einzelnen Intervallsequenzen des Ausgangsdatensatzes werden dabei berücksichtigt.

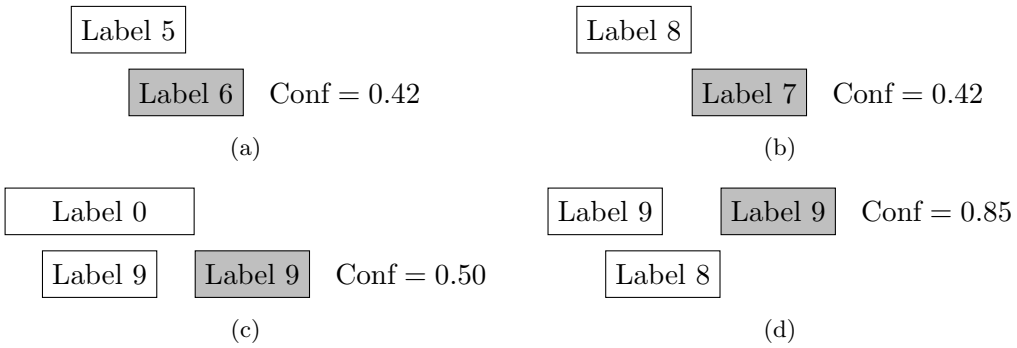


Abbildung 6.5: Im Ausgangsdatensatz wurden vier temporale Regeln künstlich eingefügt. Das Konklusionsmuster jeder Regel ist grau hinterlegt.

	Abweichung	Rang	Rang / Anzahl aller Regeln
Regel a	13.6	15	0.005
Regel b	10.3	21	0.007
Regel c	6.6	33	0.011
Regel d	2.4	170	0.056

Tabelle 6.1: Die Tabelle zeigt die Abweichung und Rang der eingefügten temporalen Regeln. Die Rangfolge bezieht sich auf die 3 009 temporalen Regeln des Ausgangsdatensatzes, die auch bei mindestens 5 simulierten Datensätzen auftraten.

zeigt die Abweichungen der künstlich eingefügten Regeln aus Abbildung 6.5. Des Weiteren gibt Tabelle 6.1 an, welchen Rang die Regeln haben, wenn die Abweichung dazu verwendet wird eine Rangfolge aller temporalen Regeln des Ausgangsdatensatz zu erstellen.

Eine Rangfolge unterstützt einen Anwendungsexperten, wenn es die Anzahl der zu überprüfenden Regeln deutlich reduziert. Der Simulationsansatz und die Bewertung der temporalen Regeln mit Hilfe der Abweichung sind zu diesem Zweck gut geeignet. Unter den besten 1.1 Prozent aller Regeln befinden sich bereits drei der vier künstlichen Regeln. Die vierte Regel befindet sich unter den besten 5.6 Prozent.

Bewertung

Obwohl der Simulationsansatz im obigen Experiment ein viel versprechendes Ergebnis lieferte, ist seine Anwendbarkeit in der Praxis zu hinterfragen. Zum einen muss eine große Anzahl von simulierten Datensätzen erstellt werden, um den Normalzustand hinreichend genau zu beschreiben. Zum anderen muss ein Datensatzsimulator existieren. Wie Kapitel 5 gezeigt hat, kann die Laufzeit der Algorithmen auf großen Datensätzen mehrere Stunden betragen. Diese Laufzeit vervielfacht sich im Simulationsansatz, da auf jedem der simulierten Datensätze alle häufigen Muster gefunden werden müssen. Im Kontext der Anwendung können sich die langen Laufzeiten des Simulationsansatzes als nicht durchführbar erweisen. Darüber hinaus existieren

in realen Daten oft bereits bekannte Abhängigkeiten zwischen einzelnen Labels. Diese Abhängigkeiten sollten durch den Datensatzsimulator berücksichtigt werden, damit die simulierten Datensätze möglichst den Gegebenheiten der Anwendung entsprechen und nur bisher unbekannte Abhängigkeiten hohe Abweichungen erzeugen. Je nach Anwendung kann die Erstellung des Datensatzsimulators daher einen eigenen zeitintensiven Arbeitsschritt darstellen.

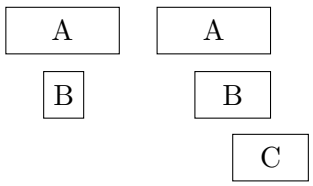
Der Simulationsansatz ist somit vor allem für einfache Anwendungen mit kleinen Datensätzen geeignet. Für komplexe Anwendungen oder große Datensätze kann der Anwender nur durch die Gütemaße Support, Konfidenz und Lift unterstützt werden.

6.2 Visualisierung von Mustern und Regeln

Ein Kritikpunkt gegenüber der Hypothesensprache temporaler Muster besteht in ihrer schweren Verständlichkeit (vgl. z.B. [Mörchen, 2006a]). Mit zunehmender Größe eines temporalen Musters wächst die Anzahl der im Muster aufgeführten Intervallrelationen quadratisch (siehe Definition 4.4). Eine tabellarische Darstellung von temporalen Mustern, ist daher bereits ab der Größe 4 für einen Anwender schwer zu interpretieren. Des Weiteren muss ein Benutzer bei der Lösung eines Anwendungsproblems, mit einer Vielzahl von Mustern arbeiten, so dass eine tabellarische oder textuelle Darstellung von temporalen Mustern nicht sinnvoll erscheint. Jedoch ist es möglich, anstelle einer tabellarischen eine graphische Repräsentation temporaler Muster zu nutzen. Graphische Darstellungen haben den Vorteil, dass sie intuitiver von einem Benutzer verstanden werden können. Abbildung 6.6 verdeutlicht den Unterschied zwischen der tabellarischen und graphischen Darstellung an einem 5-Muster.

	A	B	A	B	C
A	e	c	b	b	b
B	d	e	b	b	b
A	a	a	e	if	o
B	a	a	f	e	o
C	a	a	io	io	e

(a)



(b)

Abbildung 6.6: Ein temporales 5-Muster in tabellarischer (a) und graphischer (b) Darstellung.

Die Algorithmen *FSMSet*, *FSMTree* und *Dip* liefern als Ergebnis eine Menge von temporalen Mustern auf Basis ihrer Relationsmatrizen (d.h. in tabellarischer Form). Im Folgenden wird die Aufgabe betrachtet, aus einer gegebenen tabellarischen Darstellung eines Musters eine wohlgeformte graphische Repräsentation zu erzeugen.

Die graphische Darstellung eines Musters lässt sich einfach generieren, wenn eine Instanz des Musters gegeben ist. In diesem Fall kann jedes beteiligte temporale Intervall anhand seines Anfangs- und Endzeitpunkts an einer Zeitachse ausgerichtet werden. Ein weiterer Vorteil von Instanzen zur graphischen Repräsentation besteht darin, dass sie unabhängig von der konkreten Darstellung temporaler Intervalle sind. So können temporale Intervalle je nach Bedarf durch Rechtecke (Abbildung 6.6b), Linien (Abbildung 6.7) oder anderen Darstellungsformen abgebildet werden.

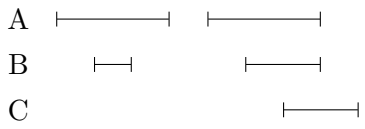


Abbildung 6.7: Das 5-Muster aus Abbildung 6.6 in alternativer graphischer Darstellung.

Algorithmus 6.1 Generierung einer Instanz zu einem gegeben temporalen Muster $P = (s, R)$

```
1:  START, LENGTH: Konstanten zur Initialisierung
2:
3:  procedure GenerateInstance(P = (s, R))
4:      IntervalSequence instance
5:      instance.add((START, START + LENGTH, s(1)))
6:      for i := 2 ... dim(P) do
7:          bconst := [⊥, ⊥]
8:          econst := [⊥, ⊥]
9:          for j := 1 ... (i − 1) do
10:             t_j := j-th element of instance
11:             updateConstraints(t_j, R[j, i], bconst, econst)
12:         end for
13:         b := solveConstraint(bconst)
14:         econst = join([b, ⊥], econst)
15:         e := solveConstraint(econst)
16:         instance.add((b, e, s(i)))
17:     end for
18:     return instance
19: end procedure
```

Die Instanz eines temporalen Musters kann auf den Ausgangsdaten mit Hilfe von endlichen Automaten gefunden werden (siehe Abschnitt 4.3.1). Eine weitere Möglichkeit besteht darin, die Algorithmen aus Kapitel 4 so zu verändern, dass sie neben den Mustern auch jeweils eine Instanz des Musters speichern. Beide Ansätze haben Nachteile. Zum einen kann die Suche nach einer Instanz je nach Größe der Daten zeitintensiv sein. Zum anderen verdoppelt das Speichern von Instanzen zu den temporalen Mustern die benötigte Speicherplatzgröße, da das temporale Muster bereits alle Informationen über Labels und Relationen enthält. Wie jedoch Algorithmus 6.1 zeigt, kann aus einem temporalen Muster auch direkt eine Instanz abgeleitet werden.

Algorithmus 6.1 bekommt ein temporales Muster P bestehend aus der Enumeration s der Labels und der Relationsmatrix R übergeben. Algorithmus 6.1 verfolgt die Idee, zunächst eine Instanz des 1-Präfixes von P zu erzeugen und diese anschließend durch Hinzufügen weiterer temporaler Intervalle schrittweise zu einer Instanz von P zu erweitern. Zur Initialisierung wird der Intervallsequenz *instance* das erste temporale Intervall hinzugefügt (Zeile 5). Dieses erste temporale Intervall besteht aus dem ersten Label des Musters $s(1)$, sowie Anfangs- und End-

$[b, e]$	*bconst* $[b_\mathrm{b}, e_\mathrm{b}]$	*econst* $[b_\mathrm{e}, e_\mathrm{e}]$
before	join($[e, \bot]$, $[b_\mathrm{b}, e_\mathrm{b}]$)	join($[e, \bot]$, $[b_\mathrm{e}, e_\mathrm{e}]$)
meets	$[e, e]$	join($[e, \bot]$, $[b_\mathrm{e}, e_\mathrm{e}]$)
overlaps	join($[b, e]$, $[b_\mathrm{b}, e_\mathrm{b}]$)	join($[e, \bot]$, $[b_\mathrm{e}, e_\mathrm{e}]$)
is-finished-by	join($[b, e]$, $[b_\mathrm{b}, e_\mathrm{b}]$)	$[e, e]$
contains	join($[b, e]$, $[b_\mathrm{b}, e_\mathrm{b}]$)	join($[b, e]$, $[b_\mathrm{e}, e_\mathrm{e}]$)
starts	$[b, b]$	join($[e, \bot]$, $[b_\mathrm{e}, e_\mathrm{e}]$)
equals	$[b, b]$	$[e, e]$

Tabelle 6.2: Die Funktion *updateConstraints* verfeinert bestehende Randbedingungen für *bconst* $[b_\mathrm{b}, e_\mathrm{b}]$ und *econst* $[b_\mathrm{e}, e_\mathrm{e}]$ mit Hilfe eines Zeitintervalls $[b, e]$ und einer gegebenen Intervallrelation. Die Einträge in der zweiten und dritten Spalte geben die neuen Randbedingungen für *bconst* und *econst* in Abhängigkeit von der Intervallrelation wieder (entweder direkt oder über einen Berechnungsschritt durch die Funktion *join*).

zeitpunkten, die durch die Konstanten *START* und *LENGTH* bestimmt sind. Offensichtlich ist das so gebildete temporale Intervall eine Instanz des 1-Präfixes von P. Algorithmus 6.1 erweitert *instance* anschließend schrittweise zu einer Instanz des i-Präfixes von P (Schleife über die Zeilen 6 – 17). Im Gegensatz zum ersten temporalen Intervall können die Zeitpunkte der folgenden temporalen Intervalle weniger frei gewählt werden, da sie den Einträgen in der Relationsmatrix R genügen müssen. Um geeignete Zeitpunkte für ein neues temporales Intervall zu finden, speichert Algorithmus 6.1 bekannte Randbedingungen über den Beginn und das Ende des temporalen Intervalls in den zweielementigen Feldern[10] *bconst* und *econst* (Zeilen 7 und 8). Zunächst sind keine Randbedingungen bekannt und die Einträge in den Feldern sind undefiniert ($\bot$). Anschließend werden nacheinander alle bereits ermittelten temporalen Intervalle und die Relationsmatrix R genutzt, um die Randbedingungen zu verfeinern (Schleife über die Zeilen 9 – 12). Das heißt, um das i-te temporale Intervall eines Musters zu bestimmen, wird die innere Schleife $(i-1)$-mal durchlaufen. Bei jedem Durchlauf aktualisiert Algorithmus 6.1 die Randbedingungen mit Hilfe des j-ten temporalen Intervalls ($j < i$) und der benötigten Intervallrelation zwischen dem j-ten und i-ten Label ($R[j, i]$) durch die Hilfsfunktion *updateConstraints* (Zeile 11). Die Funktion *updateConstraints* realisiert eine einfache Fallunterscheidung über alle 7 Intervallrelationen. Anhand der Intervallrelation und des übergebenen temporalen Intervalls können die Randbedingungen für *bconst* und *econst* verfeinert werden. Tabelle 6.2 listet alle auftretenden Fälle auf. Nachdem alle temporalen Intervalle bei der Bestimmung der Randbedingungen berücksichtigt wurden, berechnet Algorithmus 6.1 einen Beginn für das neue temporale Intervall mit Hilfe der Funktion *solveConstraint* (Zeile 13). *SolveConstraint* ermittelt den Zeitpunkt für eine Randbedingung $[b, e]$ durch das arithmetische Mittel $\frac{b+e}{2}$, falls b und e definiert sind. Modelliert $[b, e]$ ein offenes Intervall (entweder b oder e ist undefiniert), so ergibt sich der gesuchte Zeitpunkt aus $b +$ LENGTH bzw. $e -$ LENGTH.

[10] In den Feldern werden die minimalen und maximalen Zeitpunkte für den Beginn und das Ende eines temporalen Intervalls gespeichert. Ein Eintrag $[b, e]$ zeigt an, dass der Zeitpunkt zwischen b und e liegen muss. Undefinierte Einträge kennzeichnen offene Intervalle. Das heißt, bei $[b, \bot]$ muss ein Zeitpunkt größer b gewählt werden. Umgekehrt zeigt $[\bot, e]$ an, dass der Zeitpunkt vor e liegen muss.

i	j	temporales Intervall	Relation	*bconst*	*econst*	neues temporales Intervall
1	-	-	-	-	-	$(1, 5, A)$
2	1	$(1, 5, A)$	contains	$[1, 5]$	$[1, 5] \rightarrow [3, 5]$	$(3, 4, B)$
3	1	$(1, 5, A)$	before	$[5, \perp]$	$[5, \perp]$	
3	2	$(3, 4, B)$	before	$[5, \perp]$	$[5, \perp] \rightarrow [9, \perp]$	$(9, 13, A)$
4	1	$(1, 5, A)$	before	$[5, \perp]$	$[5, \perp]$	
4	2	$(3, 4, B)$	before	$[5, \perp]$	$[5, \perp]$	
4	3	$(9, 13, A)$	is-finished-by	$[9, 13]$	$[13, 13] \rightarrow [13, 13]$	$(11, 13, B)$
5	1	$(1, 5, A)$	before	$[5, \perp]$	$[5, \perp]$	
5	2	$(3, 4, B)$	before	$[5, \perp]$	$[5, \perp]$	
5	3	$(9, 13, A)$	overlaps	$[9, 13]$	$[13, \perp]$	
5	4	$(11, 13, B)$	overlaps	$[11, 13]$	$[13, \perp] \rightarrow [13, \perp]$	$(12, 17, C)$

Tabelle 6.3: Berechnung neuer temporaler Intervalle durch Algorithmus 6.1 am Beispiel des temporalen Musters aus Abbildung 6.6a (START = 1 und LENGTH = 4).

Aus dem Startzeitpunkt b für das neue temporale Intervall folgt, dass der Endzeitpunkt nicht vor b liegen kann. Diese Randbedingung $[b, \perp]$ verwendet Algorithmus 6.1, um *econst* erneut zu verfeinern (Hilfsfunktion *join* in Zeile 14). Zuletzt wird aus *econst* ein Endzeitpunkt abgeleitet (Zeile 15) und das so ermittelte temporale Intervall *instance* hinzugefügt (Zeile 16). Die Funktion *join* wird zum Verfeinern von Randbedingungen verwendet (siehe Tabelle 6.2 und Zeile 14). Für zwei Randbedingungen $[b_1, e_1]$ und $[b_2, e_2]$ berechnet *join* den Rückgabewert als $[\text{Max}(b_1, b_2), \text{Min}(e_1, e_2)]$. Für das Minimum (bzw. Maximum) von undefinierten Einträgen gilt $\text{Min}(b, \perp) = b$, $\text{Min}(\perp, b) = b$ und $\text{Min}(\perp, \perp) = \perp$ (Maximum analog).

Die Arbeitsweise von Algorithmus 6.1 soll am Beispiel des temporalen Musters aus Abbildung 6.6a (siehe Seite 119) verdeutlicht werden. Die Werte der Konstanten sind im Folgenden START = 1 und LENGTH = 4. Das erste temporale Intervall wird direkt aus den Konstanten und dem ersten Label gebildet: $(1, 1 + 4, A) = (1, 5, A)$. Für das zweite temporale Intervall werden erstmals die Randbedingungen *bconst* und *econst* benötigt. Nach der Relationsmatrix muss das temporale Intervall $(1, 5, A)$ zum zweiten temporalen Intervall in der Relation *contains* stehen. Daher müssen die neuen Anfangs- und Endzeitpunkte zwischen 1 und 5 liegen — $bconst = econst = [1, 5]$. Aus *bconst* ergibt sich der Anfangszeitpunkt des zweiten temporalen Intervalls als $\frac{1+5}{2} = 3$. Mit dem nun bekannten Anfangszeitpunkt muss die Randbedingung für den Endzeitpunkt aktualisiert werden $econst = join([3, \perp], [1, 5]) = [3, 5]$. Durch Auflösen von *econst* ($\frac{3+5}{2} = 4$) kann das zweite temporale Intervall gebildet werden $(3, 4, B)$. Die nachfolgenden temporalen Intervalle berechnen sich nach demselben Schema. Tabelle 6.3 enthält eine Übersicht über alle Zwischenergebnisse für das vorliegende Beispiel.

Im oberen Teil von Abbildung 6.8 ist die durch Algorithmus 6.1 gefundene Instanz dargestellt. Die Instanz soll der graphischen Repräsentation des Musters aus Abbildung 6.6a gegenüber einem Anwender dienen. Aus Abbildung 6.8 ist ersichtlich, dass die von Algorithmus 6.1 gebildete Instanz einige nachteilige Eigenschaften besitzt. So sind die einzelnen temporalen Intervalle unterschiedlich lang und die Zeitabstände zwischen einzelnen Intervallen wirken willkürlich. Des

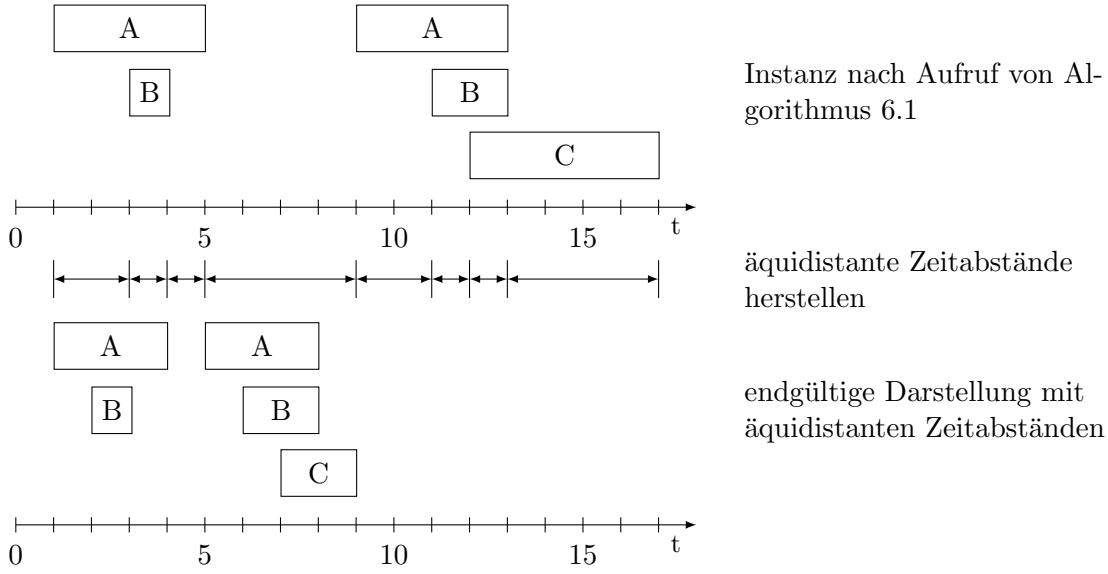


Abbildung 6.8: Die von Algorithmus 6.1 gefundenen Instanzen besitzen nicht äquidistante Zeitabstände und wirken daher bei ihrer Darstellung weniger wohlgeformt. Durch einen Nachbearbeitungsschritt werden äquidistante Zeitabstände erstellt.

Weiteren ist das erste temporale Intervall mit dem Label B nicht zentriert zum ersten temporalen Intervall mit dem Label A angeordnet. Ursache für diese Eigenschaften sind die nicht äquidistanten Zeitabschnitte zwischen den einzelnen Grenzen der beteiligten temporalen Intervalle (siehe Mitte der Abbildung 6.8). Aus diesem Grund werden die von Algorithmus 6.1 gefunden Instanzen einem Nachbearbeitungsschritt unterzogen. Während der Nachbearbeitung werden alle verschiedenen Anfangs- und Endzeitpunkte der Instanz ermittelt und auf Zeitpunkte mit einem Einheitsabstand abgebildet. Das Resultat der Nachbearbeitung für das obige Beispiel ist im unteren Teil von Abbildung 6.8 wiedergeben.

Die beschriebene Vorgehensweise zur Visualisierung temporaler Muster lässt sich auch zur Darstellung von temporalen Regeln verwenden. Dazu muss lediglich das gesamte Regelmuster visualisiert und die temporalen Intervalle des Konklusionsmusters durch Schraffuren oder Einfärbung kenntlich gemacht werden. Im Rahmen dieser Arbeit wurde eine Anwendung zur Visualisierung temporaler Muster und Regeln erstellt, die auf den beschriebenen Techniken basiert. Zusätzlich erlaubt die Anwendung Regeln nach den in Abschnitt 6.1 vorgestellten Gütemaßen zu Sortieren, zu Filtern oder beliebige Labels durch Einfärbung zu kennzeichnen. Die Software wurde bei der Analyse von Anwendungsproblemen (siehe Kapitel 7) eingesetzt. Abbildung 6.9 vermittelt einen Eindruck der Anwendung.

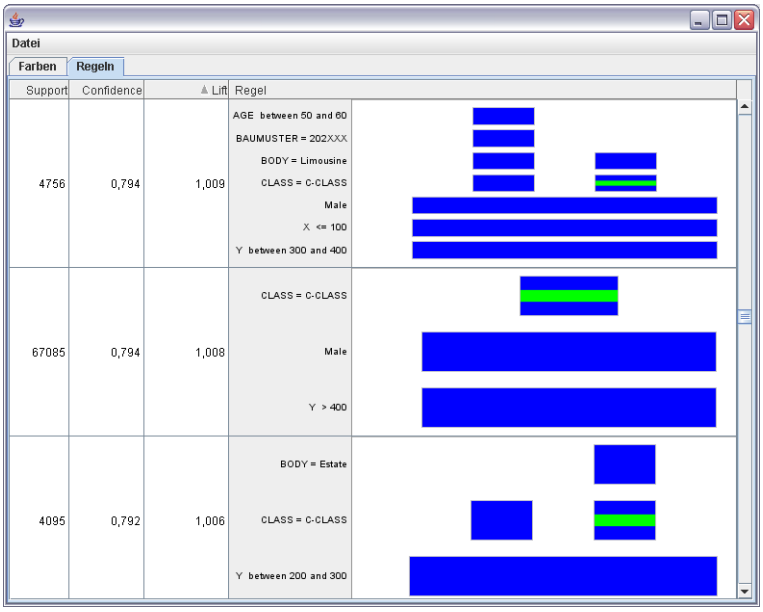


Abbildung 6.9: Eine Anwendung zur Visualisierung temporaler Muster und Regeln.

6.3 Kombination von statischen und temporalen Daten

Bisher beschränkte sich die Suche nach häufigen Mustern ausschließlich auf zeitbezogene Daten. In vielen Anwendungen werden Objekte der realen Welt nicht nur durch temporale Daten sondern auch durch statische Eigenschaften beschrieben. Beispiele für statische Eigenschaften sind das Geschlecht eines Menschen oder die Baureihe (A-Klasse, B-Klasse, etc.) eines Fahrzeugs. Statische Eigenschaften verändern sich über die Zeit nicht.

Im Folgenden wird die Aufgabe betrachtet, statische Eigenschaften in die Mustersuche zu integrieren. Ein Beispiel für die Kombination von temporalen und statischen Daten findet sich in [Eichinger u. a., 2006]. Eichinger u. a. führen die Kombination auf Modellebene durch. Das heißt, es wird ein Modell für die statischen Daten (in diesem Fall Entscheidungsbäume) und ein Modell für temporale Daten (Sequenzen) gebildet. Erst ein Metamodell verknüpft die Ergebnisse aus den statischen und temporalen Daten. Im Gegensatz zu Eichinger u. a. soll jedoch die Integration statischer Eigenschaften auf Ebene der Hypothesensprache erfolgen, um ein einheitliches Modell für temporale und statische Daten zu erzeugen.

Codierung statischer Eigenschaften durch temporale Intervalle

Ein direkter Ansatz zur Verknüpfung von statischen und temporalen Daten besteht darin, alle statischen Eigenschaften mit Hilfe temporaler Intervalle auszudrücken. In Abbildung 6.10 ist diese Idee beispielhaft dargestellt. Zunächst enthält die Intervallsequenz I_1 aus Abbildung 6.10 alle zeitbezogenen Daten in Form von temporalen Intervallen (Labels $T_1, \ldots, T_4$). Des Weiteren besitzt das durch I_1 beschriebene Objekt die statischen Eigenschaften S_1, S_2 und S_3. Sollen temporale Intervalle die statischen Eigenschaften codieren, so müssen Anfangs- und

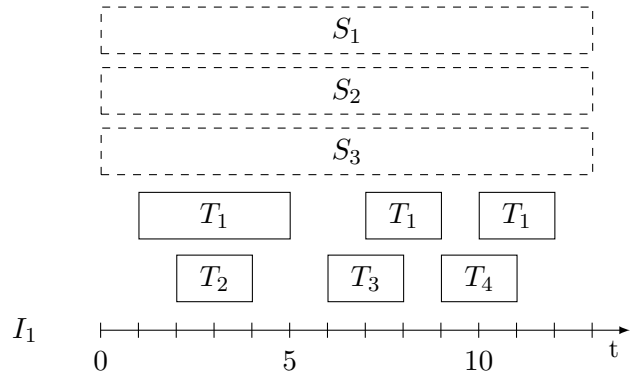


Abbildung 6.10: Temporale Intervalle können genutzt werden, um statische Eigenschaften zu beschreiben.

Endzeitpunkte für S_1, S_2 und S_3 angegeben werden, obwohl diese Eigenschaften zeitunabhängig sind. Für eine gegebene Intervallsequenz ist dieses Problem leicht zu lösen, indem als Anfang ein Zeitpunkt vor und als Ende ein Zeitpunkt nach allen anderen temporalen Intervallen gewählt wird (vgl. gestrichelte temporale Intervalle in Abbildung 6.10). Die Semantik der statischen Eigenschaften bleibt durch diese Vorgehensweise unberührt. S_1, S_2 und S_3 sind zu jedem Zeitpunkt der Intervallsequenz gültig. Als Ergebnis dieser Codierung stehen statische Eigenschaften zueinander immer in der Intervallrelation *equals*, während sie zu den übrigen temporalen Intervallen in der Relation *contains* stehen.

Nachdem die statischen Eigenschaften in Form von temporalen Intervallen in einer gegebenen Menge von Intervallsequenzen verfügbar gemacht wurden, können die häufigen temporalen Muster identifiziert werden. Die Labels eines Musters können sowohl statischen als auch zeitbezogenen Ursprungs sein. Allerdings haben die statischen Eigenschaften Auswirkungen auf den Support. Abbildung 6.11 zeigt ein temporales Muster (P), das aus den statischen Eigenschaften S_1 und S_2 und dem zeitbezogenen Label T_1 besteht. Mit der bisherigen Supportdefinition er-

P	S_1	S_2	T_1
S_1	e	e	c
S_2	e	e	c
T_1	d	d	e

Abbildung 6.11: Ein Muster bestehend aus temporalen und statischen Labels.

gibt sich der Support des Musters für die Intervallsequenz I_1 (Abbildung 6.10) als $\mathrm{Sup}(P) = 1$, da alle Instanzen von P dasselbe minimale Vorkommen erzeugen: $[0, 13]$. Aus jedem kleineren Zeitintervall ist die Intervallrelation *equals* zwischen S_1 und S_2 nicht ableitbar. Intuitiv sollte der Support aber die Frage beantworten, wie oft T_1 bei Intervallsequenzen mit den statischen Eigenschaften S_1 und S_2 vorkommt. Das heißt, für I_1 muss $\mathrm{Sup}(P) = 3$ gelten.

Wie das obige Beispiel zeigt, ist die Codierung statischer Eigenschaften durch temporale Intervalle allein nicht ausreichend, um den Support von Mustern mit statischen und zeitbe-

zogenen Labels korrekt angeben zu können. Für Muster mit beiden Typen von temporalen Intervallen dürfen die minimalen Vorkommen nur aus den temporalen Intervallen abgeleitet werden, die sich auf zeitbezogene (nicht-statische) Labels beziehen. Im Folgenden werden zwei Lösungswege vorgestellt, mit denen sich die in Kapitel 4 eingeführten Algorithmen für den Einsatz von statischen Eigenschaften erweitern lassen.

1. Anpassung der Supportberechnung

Der erste Lösungsweg steht unter der Prämisse, dass der bisherige Parametersatz der Algorithmen unverändert bleibt. Das heißt, die Eingabeparameter der Algorithmen bestehen weiterhin aus einer Menge von Intervallsequenzen und einer minimalen Supportschranke. Die Intervallsequenzen können zwar statische Eigenschaften in Form temporaler Intervalle enthalten, aber die Zuordnung, ob ein Label eine statische Eigenschaft oder eine zeitbezogene Information darstellt, wird nicht angegeben.

Auf Grund dieser Voraussetzung muss für ein gegebenes temporales Muster entschieden werden, welches Teilmuster für die Supportberechnung bestimmend ist.[11] Nur die temporalen Intervalle des Teilmusters werden anschließend zur Ableitung der minimalen Vorkommen benötigt. Durch die Codierung statischer Eigenschaften als temporale Intervalle besitzen Muster mit beiden Intervalltypen die Eigenschaft, dass ein statisches Intervall zu einem zeitbezogenen Intervall nur in der Relation *contains* stehen kann. Für ein gegebenes Muster P lässt sich die Existenz dieser Eigenschaft überprüfen, indem aus P folgendermaßen ein Graph abgeleitet wird.

1. Der Graph enthält einen Knoten für jedes Label in P.
2. Für jeden Eintrag in der oberen Dreiecksmatrix der Relationstabelle von P, der nicht *contains* ist, wird eine Kante zwischen den entsprechenden Knoten des Graphen eingefügt.

Ist der resultierende Graph *unverbunden*, so enthält P ein Teilmuster dessen Labels zu allen anderen Labels von P in der Relation *during* stehen. Dieses Teilmuster entspricht einem (verbundenen) Teilgraphen. Infolge der Sortierung der Labels in einem gültigen temporalen Muster muss das Teilmuster ein Suffix von P sein (vgl. Definition 4.7).

Der Test, ob der Graph eines temporalen Musters verbunden oder unverbunden ist, lässt sich leicht in die Supportberechnung integrieren. Für verbundene Muster bleibt die Supportberechnung unverändert. Für unverbundene Muster hingegen liefert der Test das für die Supportbestimmung relevante Teilmuster und nur aus den temporalen Intervallen dieses Teilmusters werden anschließend die minimalen Vorkommen abgeleitet.

Bei der Bestimmung des relevanten Teilmusters müssen Fehler entstehen, da letztendlich die Information, ob ein Label eine statische Eigenschaft oder ein zeitbezogenes Ereignis darstellt, nicht zur Verfügung steht. Abbildung 6.12a zeigt ein Beispiel für ein temporales Muster, bei dem der Test fehlschlägt. Wie der zugehörige Graph in Abbildung 6.12b veranschaulicht, ist das temporale Muster unverbunden. Das Teilmuster, dessen Labels zu allen übrigen Labels in der Relation *during* steht, ist das 2-Suffix (nur die Labels T_3 und T_4). In diesem Beispiel sollen aber T_1 und T_2 keine statischen Eigenschaften repräsentieren sondern zeitbezogene Ereignisse.

[11] Das heißt, welche Labels den nicht-statischen Teil des Musters repräsentieren.

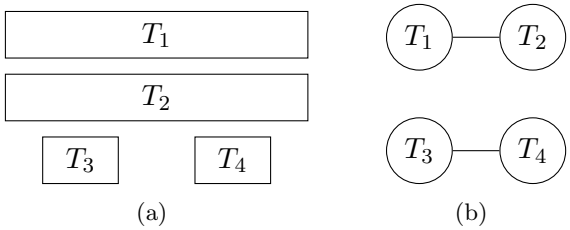


Abbildung 6.12: Der Test auf Verbundenheit kann fehlerhafte Ergebnisse liefern.

Daher müssen alle temporalen Intervalle einer Instanz des Musters für die Supportberechnung berücksichtigt werden. Es lässt sich eine Vielzahl weiterer Muster angeben, bei denen der Test unzulängliche Ergebnisse liefert.[12] Bei Experimenten mit den in Kapitel 7 präsentierten Anwendungen zeigte sich jedoch, dass die fehlerbehaftete Berechnung des Supports dieser Muster ohne Auswirkung ist. In zwei der Anwendungen (Produktbewährung und Customer Relationship Management) tritt dieser Mustertyp nicht auf und bei einer Anwendung (Diagnoseunterstützung) ist er für die Anwendung ohne Bedeutung.

Der Vorteil des ersten Lösungswegs besteht darin, dass die Algorithmen dasselbe Eingabe- und Ausgabeverhalten besitzen. Obwohl die Intervallsequenzen nun auch statische Eigenschaften enthalten, können allein durch Angabe der Daten und einer minimalen Supportschwelle alle häufigen Muster gefunden werden. Der Nachteil ist, dass für einen Teil der Muster ein falscher Support berechnet werden kann. Falls diese Muster für die Anwendung eine Rolle spielen, muss der nachfolgend beschriebene zweite Lösungsweg genutzt werden.

2. Erweiterung der Hypothesensprache

Der zweite Lösungsweg erweitert den bestehenden Parametersatz der Algorithmen um eine Zuordnung der Labels zu statischen Eigenschaften bzw. zeitbezogenen Ereignissen. Durch die Zuordnung kann für jedes temporale Muster korrekt entschieden werden, welches Teilmuster zeitbezogene Ereignisse beschreibt.

Um die Information über statische Eigenschaften auf Ebene der temporalen Muster auszudrücken, wird die Menge von Allens Intervallrelationen um die Relation *static* ergänzt. Die Relation *static* gilt immer dann, wenn mindestens eines der beteiligten Labels eine statische Eigenschaft repräsentiert. Abbildung 6.13 veranschaulicht den Einsatz der neuen Intervallrelation. Abbildung 6.13a zeigt ein temporales Muster bei dem die Labels T_1 und T_2 zeitbezogene Ereignisse beschreiben (vgl. graphische Darstellung in Abbildung 6.12). Codieren aber T_1 und T_2 statische Eigenschaften, so entsteht das temporale Muster aus Abbildung 6.13b.

Auf Grund der vorhandenen Zuordnung von Labels zu statischen Eigenschaften kann aus jedem temporalen Muster das Teilmuster extrahiert werden, welches zeitbezogene Labels enthält. Bei Mustern mit beiden Labeltypen dürfen wiederum nur die temporalen Intervalle der zeitbezogenen Labels zur Supportbestimmung herangezogen werden.

Im Gegensatz zum ersten Lösungsweg ist die korrekte Berechnung des Supports für alle

[12] Der Test erkennt z.B. auch dann das Muster aus Abbildung 6.12 als unverbunden, wenn T_1 zu T_2 in der Relation *overlaps* steht.

	T_1	T_2	T_3	T_4
T_1	*equals*	*equals*	*contains*	*contains*
T_2	*equals*	*equals*	*contains*	*contains*
T_3	*during*	*during*	*equals*	*before*
T_4	*during*	*during*	*after*	*equals*

(a)

	T_1	T_2	T_3	T_4
T_1	*static*	*static*	*static*	*static*
T_2	*static*	*static*	*static*	*static*
T_3	*static*	*static*	*equals*	*before*
T_4	*static*	*static*	*after*	*equals*

(b)

Abbildung 6.13: Die Erweiterung von Allens Intervallrelationen um die Relation *static* ermöglicht es, verschiedene Varianten eines Musters eindeutig zu unterscheiden. Links beschreiben T_1 und T_2 zeitbezogene Ereignisse, rechts codieren sie statische Eigenschaften (vgl. Abbildung 6.12a).

temporalen Muster möglich. Allerdings ist zusätzlich die Information erforderlich, welches Label eine statische Eigenschaft beschreibt.

Bei beiden Lösungswegen ist garantiert, dass alle häufigen Muster gefunden werden. Wie bereits erläutert, müssen die temporalen Intervalle von statischen Eigenschaften zu allen zeitbezogenen Intervallen in der Relation *contains* stehen. *Contains* impliziert „beginnt vor“ und „endet nach“. Die Sortierung der Labels in einem gültigen temporalen Muster erzwingt daher, dass alle statischen Eigenschaften am Anfang und alle zeitbezogenen Ereignisse am Ende des Musters stehen müssen. Das heißt, alle statischen Eigenschaften bilden ein Präfix und alle zeitbezogenen Ereignisse ein Suffix des Musters. Das partielle Apriori-Kriterium (siehe Satz 4.31) bleibt unverletzt, da die Supportberechnung weiterhin alle zeitbezogenen Ereignisse berücksichtigt.

Die beiden vorgeschlagenen Lösungswege erlauben es daher, zeitbezogene Ereignisse und statische Eigenschaften in einem gemeinsamen Modell zu vereinen und mit Hilfe der in Kapitel 4 vorgestellten Algorithmen die häufigen Muster zu identifizieren.

6.4 Zusammenfassung

In diesem Kapitel wurde der Fokus auf ergänzende Verfahren zu häufigen temporalen Mustern gelegt, die einen Benutzer bei der Lösung seines Anwendungsproblems unterstützen sollen. Dazu wurden im ersten Teil die Ableitung von temporalen Regeln aus den häufigen Mustern vorgestellt. Temporale Regeln ermöglichen es Data Mining-Aufgaben wie z.B. Klassifikation, Prognose, Abhängigkeitserkennung oder Abweichungsanalyse zu adressieren. Die Gewinnung temporaler Regeln aus den häufigen Mustern orientiert sich dabei an der Ableitung von Assoziationsregeln aus häufigen Warenkörben. Trotz der starken Analogie zwischen temporalen Regeln und Assoziationsregeln erweisen sich etablierte Gütemaße zur Bewertung von Assoziationsregeln als ungeeignet für temporale Regeln. Die Ursache hierfür ist die gewählte Supportdefinition. Sie erlaubt es zwar, mehrfache Vorkommen eines temporalen Musters innerhalb einer Intervallsequenz zu berücksichtigen, dabei geht aber die Eigenschaft verloren, neben dem Support eines Musters X ($\mathrm{Sup}(X)$) auch die Häufigkeit des Nichtauftretens von X ($\mathrm{Sup}(\overline{X})$) semantisch konsistent angeben zu können. Viele Gütemaße für Assoziationsregeln benötigen jedoch zur Bewertung einer Regel $X \Rightarrow Y$ den Wert $\mathrm{Sup}(\overline{X})$. Um dennoch die gefundenen

Regeln eines Datensatzes bewerten zu können, wurden zwei Alternativen vorgeschlagen. Zum einen bietet das Gütemaß *Lift* für temporale Regeln die Möglichkeit, eine Regel anhand des Supports des Regel-, Prämissen- und Konklusionsmusters zu bewerten. Zum anderen wurde ein Simulationsansatz vorgeschlagen. Beim Simulationsansatz werden mehrere Datensätze mit denselben Eigenschaften zufällig erzeugt, die auch der Originaldatensatz besitzt. Die Konfidenzwerte einer temporalen Regel auf den zufälligen Datensätzen werden anschließend zur Bewertung der Regel verwendet.

Der zweite Abschnitt widmete sich einer geeigneten Darstellung von temporalen Mustern und Regeln. Hier wurde ein Algorithmus angegeben, der zu einem beliebigen temporalen Muster eine wohlgeformte Instanz bildet. Die Abstände zwischen Anfangs- und Endzeitpunkten der einzelnen temporalen Intervalle in der Instanz sind äquidistant. Die Instanzen sind daher gut geeignet, um temporale Muster und Regeln zu visualisieren.

Der letzte Abschnitt beschäftigte sich mit der Integration von statischen Eigenschaften in die Mustersuche. Statische Eigenschaften beinhalten Informationen über die Objekte der realen Welt, die nicht zeitbezogen sind (z.B. das Geschlecht eines Kunden). Es konnten zwei Wege aufgezeigt werden, die es erlauben, statische Eigenschaften und zeitbezogene Ereignisse mit Hilfe temporaler Intervalle in einem gemeinsamen Modell zu vereinen. Die erste Möglichkeit verzichtet auf eine explizite Angabe, welche Labels statische Eigenschaften bzw. zeitbezogene Ereignisse beschreiben. Mit Hilfe eines (heuristischen) Tests wird das zeitbezogene Teilmuster eines Musters bestimmt und zur Supportevaluation herangezogen. Die zweite Möglichkeit erweitert den Parametersatz der Algorithmen aus Kapitel 4 um eine Zuordnung der Labels zu statischen Eigenschaften bzw. zeitbezogenen Ereignissen. Anhand der Zuordnung können demzufolge auf Ebene der Hypothesensprache statische Eigenschaften durch die neue Intervallrelation *static* kenntlich gemacht werden.

Kapitel 7

Anwendungen in der Automobilindustrie

Dieses Kapitel beschreibt drei Anwendungen für die Suche nach temporalen Mustern und Regeln in zeitbezogenen Daten in der Automobilindustrie. Die Anwendungen dienten als Motivation und Orientierungshilfe für viele der notwendigen Designentscheidungen beim Entwurf der Algorithmen (z.B. Mustersprache, Supportdefinition) in Kapitel 4. Im Folgenden wird jede Anwendung anhand des CRISP-DM Prozesses (vgl. Abschnitt 2.1.1) näher erläutert.

7.1 Qualitätsüberwachung von Fahrzeugflotten

In der ersten Anwendung wird die Qualität produzierter Fahrzeuge überwacht.

Anwendungsanalyse

Ein wichtiges Merkmal für jeden Automobilhersteller ist die Qualität seiner produzierten Fahrzeuge. Zum einen ist Qualität ein verkaufsentscheidendes Kriterium für Neufahrzeugkäufer und wirkt daher umsatzsteigernd. Zum anderen besteht eine direkte Verbindung zwischen der produzierten Qualität und dem Unternehmensgewinn. Hersteller wie z.B. Daimler geben auf ihre Fahrzeuge eine zweijährige Neufahrzeuggarantie sowie einen modellabhängigen Kulanzzeitraum. Eine geringere Qualität führt zu einer höheren Anzahl an Reparaturen im Garantie- und Kulanzzeitraum und damit zu höheren Kosten.

Qualitätssicherung und Qualitätsüberwachung werden daher zu jedem Zeitpunkt des Fahrzeuglebenszyklus durchgeführt. Bereits Forschungs- und Entwicklungsprozesse orientieren sich an Qualitätsstandards und müssen Meilensteine (Quality Gates) erreichen. Auch während der Produktion werden Produktionsparameter und ihr Einfluss auf Maße und Toleranzen kontrolliert. Der dritte und für diese Anwendung maßgebliche Zeitraum schließt sich nach dem Verkauf eines Fahrzeuges an (Aftersales). In diesem Zeitraum wird die Qualität der Fahrzeuge durch die *Produktbewährung* überwacht.

Die Aufgabe der Produktbewährung ist es, Qualitätsmängel der Fahrzeuge im Feld (d.h. beim Kunden) frühzeitig zu erkennen, einzugrenzen und mögliche Ursachen zu identifizieren. Aus den Erkenntnissen der Produktbewährung werden anschließend Maßnahmen zur Verbesserung der Qualität abgeleitet. Solche Maßnahmen können z.B. die Entwicklung neuer Bauteile, Änderungen in der laufenden Produktion oder Reparatur- und Wartungsarbeiten in Werkstätten betreffen.

Die Produktbewährung nutzt vor allem die durch Vertragswerkstätten gemeldeten Reparatur- und Wartungsarbeiten als Grundlage für die Überwachung der Qualität im Feld. Des Weiteren

werden Informationen über die Konfiguration der Fahrzeuge (z.B. verbaute Sonderausstattungen) aus der Produktion genutzt. Bei Daimler werden diese Datenquellen mit Hilfe von Verfahren der intelligenten Datenanalyse überwacht und ausgewertet. Bei diesen Auswertungen blieb bisher die zeitliche Abfolge der Reparaturen an Fahrzeugen unberücksichtigt bzw. wurde nur in Einzelfällen einbezogen. Das systematische Entdecken von temporalen Mustern und Regeln liefert der Produktbewährung Hinweise dafür, ob das Eintreten eines Schadens durch vorherige Reparatur- oder Wartungsarbeiten bedingt ist. Gründe für solche Reparaturabfolgen können zeitlich verzögerte Folgeschäden oder unsachgemäße Reparatur- und Wartungsarbeiten sein.

Temporale Muster und Regeln können keine bestehenden Analysen der Produktbewährung ablösen. Vielmehr ergänzen sie die bestehenden Analysemöglichkeiten um zeitliche Aspekte. In diesem Zusammenhang ist auch die Erfolgsbewertung des Einsatzes von temporalen Mustern und Regeln zu sehen. Das Erfolgskriterium besteht darin, ob ein Produktbewährer in seiner Arbeit durch die zusätzlichen Informationen der Muster und Regeln unterstützt wird.

Datenverständnis

Wie bereits oben angeführt, entstammen die Daten aus zwei Quellen: den Werkstätten und der Produktion. Die Werkstattdaten enthalten die Informationen, welche Arbeiten an einem Fahrzeug durchgeführt wurden. Dazu gehören unter anderem die Fahrzeugidentifikationsnummer (FIN), Werkstattkennung, Laufleistung, Reparaturdatum, identifizierter Schaden codiert in einem Schadensschlüssel (SSL), Arbeitsleistungen, Materialleistungen und Reparaturkosten. Aus der Produktion stammt die so genannte Fahrzeugdokumentation. Sie beschreibt zu jeder FIN die Bauart des Fahrzeuges. Einige relevante Informationen sind hier z.B. die Baureihe, Baumuster, Motor- und Getriebereihe sowie alle codierten Ausstattungsmerkmale (z.B. Typ der Klimaanlage, Schiebedach, Radio, Navigation, etc.).

Beide Datenquellen besitzen das gemeinsame Schlüsselattribut FIN und können daher einfach verknüpft werden. Auf Grund ihrer zentralen Bedeutung liegen in der Produktbewährung beide Datenquellen bereits verknüpft in einer Datenbank vor und brauchen für die Anwendung lediglich eingebunden werden. Abbildung 7.1 veranschaulicht die verschiedenen Datenquellen.

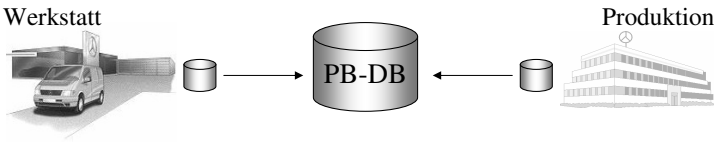


Abbildung 7.1: Der Produktbewährung (PB-DB) liegen sowohl Informationen aus den Werkstätten als auch der Produktion vor.

Datenaufbereitung

Die Aufbereitung der Daten muss sich an der Arbeitsweise der Produktbewährer orientieren. Die Produktbewährung ist in einer Matrixform organisiert. Das heißt, es gibt auf der einen Seite Produktbewährer, die die Qualität einer gesamten Baureihe (z.B. A-Klasse, B-Klasse, etc.) überwachen. Dem gegenüber stehen Spezialisten für bestimmte Bauteile (z.B. Klimaanlage, Motor, Getriebe, etc.), welche die Qualität dieser Bauteile über verschiedene Baureihen

hinweg kontrollieren. Analysen werden bevorzugt innerhalb einer Baureihe und für bestimmte (vergleichbare) Produktionszeiträume durchgeführt.

Für die Datenaufbereitung bedeutet dies, dass die Suche nach temporalen Mustern und Regeln nicht auf der gesamten Datenmenge vollzogen wird. Vielmehr werden Teilmengen für die gewünschten Baureihen und Produktionszeiträume verwendet. Ein weiteres Filterkriterium betrifft Standardwartungsarbeiten an Verschleißteilen (z.B. Ölwechsel, Wischerblätter). Diese sind in großer Zahl in den Daten aus den Werkstätten vorhanden, besitzen für den Großteil der Analysen jedoch keine Bedeutung. Um den Datenumfang (und auch Ergebnisumfang) möglichst klein zu halten, werden solche Standardwartungsarbeiten bei der Datenaufbereitung herausgefiltert.

Das Ergebnis der Datenaufbereitung ist eine Menge von Intervallsequenzen. Jede Intervallsequenz beschreibt die bekannten Informationen zu einem individuellen Fahrzeug (d.h. zu einer FIN). Eine Intervallsequenz enthält entsprechend den Datenquellen zwei verschiedene Arten von Informationen. Die aus der Produktion stammenden Daten über die Fahrzeugkonfiguration sind statisch, das heißt, sie unterliegen keinen Änderungen nachdem das Fahrzeug verkauft wurde.[1] Erst die Daten aus den Werkstätten besitzen einen Zeitbezug. Die tagesgenauen Angaben über durchgeführte Reparaturen (codiert als Schadensschlüssel) ergänzen die statischen Fahrzeugkonfigurationen. Abbildung 7.2 zeigt ein Beispiel für eine mögliche Intervallsequenz. Das Fahrzeug aus der Baureihe C-Klasse mit Dieselmotor und silber-metallic Lackierung hatte bisher drei Reparaturen (Schadensschlüssel SSL1, SSL2 und SSL3).

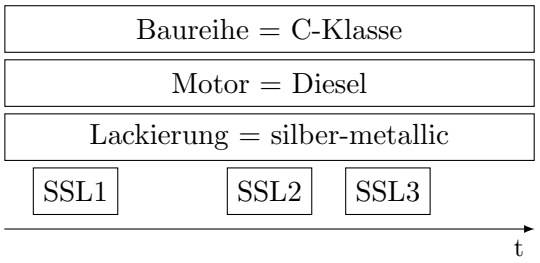


Abbildung 7.2: Beispiel für eine Intervallsequenz aus der Qualitätsüberwachung

Die beschriebene Datenaufbereitung wurde auch für den Datensatz *PRODUKTBEWÄHRUNG* für die Evaluation der Algorithmen in Kapitel 5 verwendet. Der Datensatz enthält Informationen zu einer Jahresproduktion von Chrysler-Fahrzeugen und besteht aus 101 250 Intervallsequenzen. Jede Intervallsequenz besteht wiederum aus 14 bis 48 temporalen Intervallen. Insgesamt gibt es 345 verschiedene Labels im Datensatz.

Data Mining

Für den eigentlichen Schritt des Data Minings werden die aus Kapitel 4 bekannten Algorithmen verwendet.

Verwandte Verfahren (vgl. Abschnitte 3.1 und 3.2) erweisen sich als ungeeignet, da die Anwendung bestimmte Anforderungen an die Algorithmen stellt. Zu den Anforderungen gehören:

[1] Prinzipiell kann ein Fahrzeug auch umgerüstet werden. Der Anteil dieser Fahrzeuge ist jedoch zu gering, als dass sie eine Rolle für die Produktbewährung spielen würden.

1. Der Support eines temporalen Musters muss durch die Anzahl seiner Vorkommen in den Daten definiert sein.
2. Mehrfache Vorkommen eines Musters innerhalb einer Intervallsequenz müssen berücksichtigt werden.
3. Es müssen zeitliche Bedingungen für die Gültigkeit einer Instanz angegeben werden können.

Die erste Anforderung ergibt sich aus praktischen Erwägungen. Die Produktbewährung ist ein stark interdisziplinäres Fachgebiet. Die jeweiligen Experten kommen z.B. aus den Gebieten der Elektrotechnik, der Physik, des Maschinenbaus, der Fertigungstechnik oder der Informatik. Den Support eines Musters über die Anzahl seiner Vorkommen zu definieren, entspricht der natürlichen Fragestellung der Produktbewährer, wie oft eine Abfolge von Schäden auftritt. Im Gegensatz dazu erschließt sich die Bedeutung der Supportdefinition *frequency* (vgl. [Mannila u. a., 1997] und Abschnitt 3.2.1, Seite 18ff.) erst durch eine wahrscheinlichkeitsbasierte Interpretation. Jedoch ist nicht jeder Produktbewährer gleichermaßen zu dieser Interpretation im Stande. In Folge dessen würden temporale Muster mit der Supportdefinition *frequency* keine Akzeptanz finden.

Verfahren der Sequenzanalyse (siehe Abschnitt 3.1.2) oder die in Abschnitt 3.2.2 vorgestellten Verfahren vernachlässigen mehrfache Vorkommen eines Musters innerhalb einer Eingabesequenz. Für diese Anwendung müssen alle Vorkommen eines Musters berücksichtigt werden, da ein temporales Muster potentiell einen Qualitätsmangel in Form von Abfolgen von Schadensschlüsseln beschreibt. Die Vernachlässigung sich wiederholender Schadensabfolgen würde dazu führen, dass ein Qualitätsmangel durch den Support als weniger schwerwiegend dargestellt wird.

Die letzte Anforderung verlangt die Möglichkeit zeitliche Bedingungen für die Instanzen eines Musters angeben zu können. In der Anwendung verbirgt sich dahinter die Überlegung, dass zwei Schäden, die im Abstand von einem Jahr aufeinander folgen, weniger wahrscheinlich in einem kausalen Zusammenhang stehen, als zwei Schäden, die in einem Monat auftreten. Durch die Angabe einer Obergrenze für die zeitliche Ausdehnung einer Instanz, kann die Gültigkeit der Instanz gesteuert werden. Nur gültige Instanzen werden anschließend bei der Supportevaluation berücksichtigt.

Bewertung

Durch die gefundenen temporalen Muster und Regeln sind die Produktbewährer in der Lage zu bewerten, ob ein zu untersuchendes Qualitätsproblem mit früheren Reparatur- und Wartungsarbeiten in Zusammenhang steht. Eine vergleichbare systematische Analyse war bisher nicht im Analyseportfolio der Produktbewährung enthalten. In Folge dessen ist eine Unterstützung der Produktbewährung durch die Suche nach temporalen Mustern und Regeln gegeben und sinnvoll. Bestehende Auswerteverfahren bleiben durch temporale Muster jedoch unberührt (z.B. zur Fehlerfrüherkennung oder zur Ursachenanalyse). Eine erwähnenswerte Eigenschaft der Anwendung ist, dass eine Unterstützung der Produktbewährung selbst dann stattfindet, wenn keine auffälligen Zusammenhänge zu früheren Reparaturen erkennbar sind. Für die Produktbewährer kann auch die Abwesenheit solcher Zusammenhänge eine nützliche Information darstellen.

Umsetzung

Die Suche nach temporalen Mustern und Regeln kann die Produktbewährung am besten unterstützen, wenn die Experten sie selbstständig bei der Untersuchung eines Qualitätsproblems anwenden. Diese Erkenntnis basiert auf Erfahrungen mit vergleichbaren Anwendungen (siehe z.B. [Blumenstock u. a., 2006]). Im Rahmen einer Umsetzung müssen die Data Mining Algorithmen daher in einer integrierten Anwendung[2] den Produktbewährern zur Verfügung gestellt werden. Sowohl für die Chrysler Group als auch die Mercedes Car Group wird derzeit überprüft, ob dieser Schritt mit Hilfe einer Intranetanwendung realisierbar ist.

7.2 Diagnoseunterstützung (CAN-Bus)

In der zweiten Anwendung wird die Diagnose von Steuergerätfehlern unterstützt.

Anwendungsanalyse

Eine maßgebliche Entwicklung der letzten Jahrzehnte im Automobilbau war die zunehmende Anzahl elektronischer Steuergeräte im Fahrzeug. Premiumfahrzeuge können über 70 Steuergeräte besitzen. Daher sind Steuergeräte in praktisch jedem Bereich des Fahrzeuges anzutreffen (z.B. Motor, Getriebe, Bremsen, Türen oder Kombiinstrument vgl. Abbildung 7.3) [Huber, 2004]. Die meisten Steuergeräte sind über Bussysteme miteinander vernetzt, um systemweit Informationen austauschen zu können. Bei der Erprobung neuer Fahrzeuge und Steuergeräte ist die Überwachung der Bussysteme von zentraler Bedeutung. Anhand der Kommunikation über das Bussystem kann ein Experte nachvollziehen, in welchen Zuständen sich ein Steuergerät befand und eine Diagnose im Falle eines Steuergerätfehlers durchführen.

Die für diese Anwendung relevanten Daten entstammen einer Erprobungsflotte der Chrysler Group. Jedes Fahrzeug der Flotte ist mit einem Datenrekorder ausgerüstet, welches die Kommunikation der Steuergeräte am zentralen Bussystem (dem so genannten CAN-Bus – *C*ontroller *A*rea *N*etwork [Lawrenz, 2007]) protokolliert. Tritt während einer Erprobungsfahrt ein Steuergerätfehler auf, so speichert der Datenrekorder die Kommunikation auf dem CAN-Bus aus dem Zeitbereich 20 Sekunden vor und 10 Sekunden nach dem Eintreten des Fehlers in einer Diagnosedatei. Daten die außerhalb dieses Zeitbereiches anfallen, können auf Grund technischer Einschränkungen vom Datenrekorder nicht persistent abgelegt werden.

Zur Diagnose eines Steuergerätfehlers nutzt der Experte die Aufzeichnungen des Datenrekorders. Die Arbeit mit den Diagnosedateien muss an dieser Stelle als manuell bezeichnet werden. Das heißt, der Experte überprüft die Diagnosedateien einzeln und bildet aus seinen Beobachtungen eine Hypothese über die mögliche Ursache des Steuergerätfehlers. Oft werden bei dieser Vorgehensweise nicht alle verfügbaren Diagnosedateien herangezogen. Die Suche nach temporalen Mustern und Regeln soll dem Experten Hinweise dafür geben, ob Zusammenhänge zwischen der aufgetretenen Buskommunikation und dem Steuergerätfehler bestehen. Des Weiteren können durch den algorithmischen Ansatz alle zur Verfügung stehenden Diagnosedateien berücksichtigt werden. Das Erfolgskriterium dieser Anwendung besteht darin, ob die gefundenen temporalen Muster und Regeln den Experten bei seiner Hypothesenfindung unterstützen.

[2] Sowohl die Datenanbindung, die Datenaufbereitung als auch die Präsentation der Analyseergebnisse müssen von der Anwendung geeignet unterstützt werden.

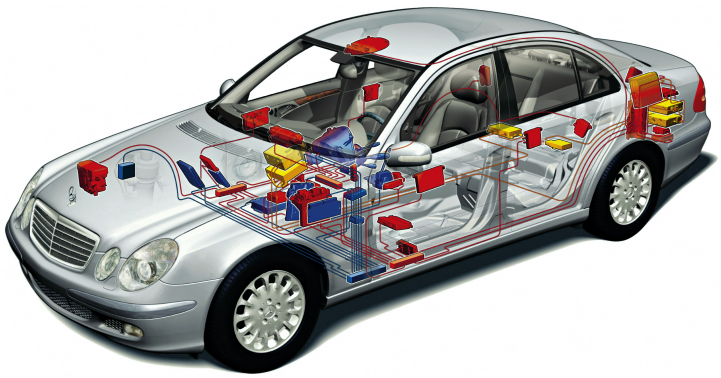


Abbildung 7.3: Schematische Darstellung von elektronischen Steuergeräten, ihrer Position im Fahrzeug sowie ihrer Vernetzung mit Hilfe von Bussystemen anhand einer E-Klasse.

Datenverständnis

Der CAN-Bus dient den Steuergeräten zum Austausch systemweiter Informationen. Zu diesen Informationen gehören vor allem die Messwerte der verschiedenen Sensoren im Fahrzeug (bzw. die aus ihnen abgeleiteten Größen). Vom Motorsteuergerät können z.B. die Werte für Drehzahl, Drehmoment und Motortemperatur stammen, während das Getriebesteuergerät den gegenwärtig eingelegten Gang liefert. Im Fahrzeug gibt es mehr als 100 solcher Signale, die vom Datenrekorder im Fehlerfall aufgezeichnet werden können. Der Datenrekorder wird aber in der Regel durch die Experten so konfiguriert, dass nur die für die Diagnose relevanten Signale aufgezeichnet werden. Mit einer Frequenz von 10 Hz überprüft der Datenrekorder den CAN-Bus auf die programmierten Signale und speichert die Werte im Fehlerfall.

In den entstehenden Diagnosedateien können zwei Arten von Signalen unterschieden werden. Zum einen gibt es Signale mit einem kontinuierlichem Wertebereich (z.B. Drücke, Temperaturen, Drehzahlen, Batteriespannung, etc.). Zum anderen gibt es einige Signale, die einen diskreten Wertebereich (z.B. eingelegter Gang, Kupplung offen oder geschlossen, Stand-, Ab- oder Aufblendlicht eingeschaltet, etc.) aufweisen. Auf Grund der Abtastrate von 10 Hz ergeben sich für jedes aufgezeichnete Signal 300 Werte pro aufgetretenem Steuergerätfehler. Abbildung 7.4 gibt jeweils ein Beispiel für die aufgezeichneten Werte eines diskreten und kontinuierlichen Signals.

Datenaufbereitung

Im ersten Schritt der Datenaufbereitung wurde eine sinnvolle Teilmenge der Daten selektiert. Der Datenrekorder wird in unterschiedlichen Konfigurationen in verschiedenen Fahrzeugtypen eingesetzt. Für eine Analyse muss die Vergleichbarkeit der ausgewählten Aufzeichnungen des Datenrekorders sichergestellt sein.[3] Im vorliegenden Fall wurden dazu 85 Aufzeichnungen des Datenrekorders zu einem Steuergerätfehler von 29 verschiedenen Fahrzeugen ausgewählt.

[3] Die Fehler eines Motorsteuergerätes können z.B. nur dann sinnvoll miteinander verglichen werden, wenn die betroffenen Fahrzeuge den gleichen Motor besitzen.

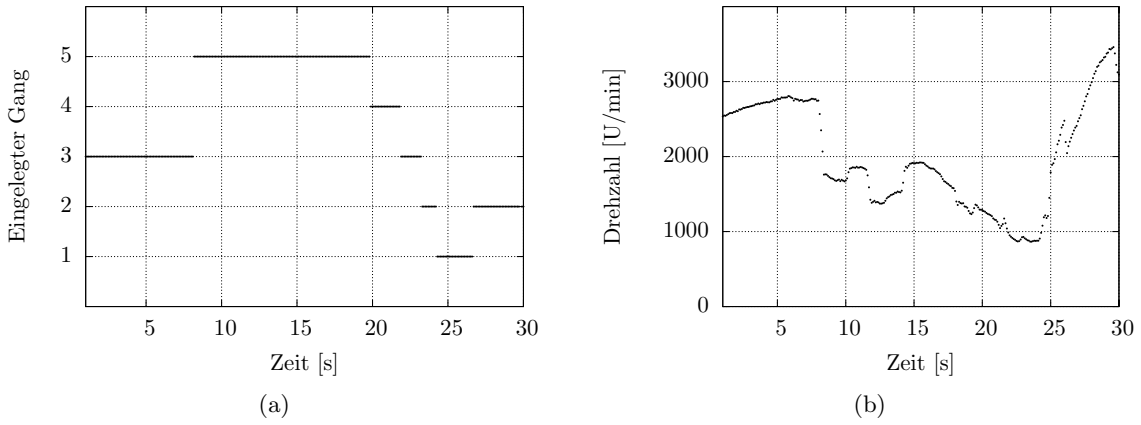


Abbildung 7.4: Beispiele für zwei aufgezeichnete Signale: (a) das diskrete Signal des eingelegten Gangs und (b) das kontinuierliche Signal für die gegenwärtige Motordrehzahl.

Als nachteilig erwies sich die freie Konfigurierbarkeit des Datenrekorders. Obwohl bei jedem Steuergerätfehler mehrere Dutzend Signale aufgezeichnet werden können, waren lediglich 15 Signale in jedem der 85 Aufzeichnungen vertreten. Zu diesen Signalen gehörten unter anderem der eingelegte Gang, die Motordrehzahl, die Turboladerdrehzahl, die Katalysatortemperatur, der Sauerstoffgehalt des Abgases, die Batteriespannung sowie Signale für die Zündung, Klimaanlage, Bremsen und Kupplung.

Im zweiten Schritt der Datenaufbereitung mussten alle aufgezeichneten Signale in Intervallsequenzen konvertiert werden. Jede Intervallsequenz entspricht hierbei einer der 85 Aufzeichnungen des Datenrekorders. Für jedes der kontinuierlichen Signale wurden Schwellwerte ermittelt, die einen bestimmten Wertebereich definieren. Als Beispiel diene die Aufbereitung des Motordrehzahlsignals mit Hilfe der Schwellwerte 1 000, 2 000 und 3 000 Umdrehungen pro Minute. Jedes temporale Intervall des Motordrehzahlsignals gibt an, über welchen Zeitraum ein Fahrzeug innerhalb eines Drehzahlbereiches betrieben wurde. Für das Motordrehzahlsignal aus Abbildung 7.4b ergibt sich anhand der obigen Schwellwerte die Intervallsequenz: (0.0, 8.3, Motordrehzahl zwischen 2 000 und 3 000), (8.4, 21.8, Motordrehzahl zwischen 1 000 und 2 000), (21.9, 24.3, Motordrehzahl kleiner 1 000), (24.4, 25.3, Motordrehzahl zwischen 1 000 und 2 000), (25.4, 28.0, Motordrehzahl zwischen 2 000 und 3 000), (28.1, 30.0, Motordrehzahl größer 3 000). Analog erfolgte die Verarbeitung diskreter Signale. Anstelle von Schwellwerten konnten jedoch die verschiedenen Werte des diskreten Signals verwendet werden. Für das Signal des eingelegten Gangs aus Abbildung 7.4a liefert die Datenaufbereitung daher die folgenden temporalen Intervalle: (0.0, 0.9, Gang 2), (1.0, 8.1, Gang 3), (8.2, 19.8, Gang 5), (19.9, 21.8, Gang 4), (21.9, 23.2, Gang 3), (23.3, 24.2, Gang 2), (24.3, 26.6, Gang 1), (26.7, 30.0, Gang 2).

Der auf diese Art entstandene Datensatz CAN-BUS wurde ebenfalls für die Evaluation der Algorithmen in Kapitel 5 verwendet. CAN-BUS enthält 85 Intervallsequenzen. Jede Intervallsequenz besteht aus 24 bis 139 temporalen Intervallen. Insgesamt exisiteren 93 verschiedene Labels im Datensatz.

Data Mining

Zur Suche nach häufigen temporalen Mustern wurden wiederum die in Kapitel 4 vorgestellten Algorithmen verwendet.

Wie bereits bei der Qualitätsüberwachung von Fahrzeugflotten, können Muster mehrmals innerhalb einer Intervallsequenz auftreten (z.B. wiederholte Gangwechsel) und müssen bei der Auswertung berücksichtigt werden. Durch diese Anforderung sind die in Abschnitt 3.2.2 vorgestellten Verfahren nicht einsetzbar. Im Gegensatz zur Qualitätsüberwachung stellt die Anwendung aber keine zeitlichen Bedingungen an die Instanzen eines Musters, da diese indirekt bereits durch die 30 Sekunden dauernden Aufzeichnungen des Datenrekorders integriert sind.

Bewertung

Die Analyse des Datensatzes CAN-BUS erzeugte eine Vielzahl von temporalen Mustern und Regeln. Für die Anwendung ist jedoch nur die Teilmenge von Regeln relevant, die einen Zusammenhang zwischen den 15 aufgezeichneten Signalen und dem Steuergerätfehler aufzeigen. Diese Teilmenge lässt sich leicht aus der Menge aller Regeln gewinnen, indem geeignete Filterkriterien angewendet werden. Bei der Bewertung mit Anwendungsexperten zeigte sich jedoch, dass die gefundenen Regeln keine Hinweise auf einen kausalen Zusammenhang zwischen den 15 Signalen und dem Steuergerätfehler liefern. Obwohl die Suche nach temporalen Mustern und Regeln in diesem Fall keine neuen Hinweise für die Anwendung erbrachte, bestätigten die Anwender, dass temporale Regeln das Potential besitzen die Diagnose von Steuergerätfehlern zu unterstützen. Der Vorteil der Algorithmen besteht darin, alle vorhandenen Aufzeichnungen zu einem Steuergerätfehler nutzen zu können, während ein Experte nur einige Stichproben zu untersuchen vermag. Des Weiteren verbessert sich die Aussagekraft der Regeln, je mehr Signale des Datenrekorders berücksichtigt werden.

Umsetzung

Die Unterstützung der Diagnose von Steuergerätfehlern wurde studienartig an einem Beispiel durchgeführt (Datensatz CAN-BUS). Eine wichtige Erkenntnis dieser Studie war, dass die Konfiguration der Datenrekorder (d.h. die Auswahl der aufzuzeichnenden Signale) vereinheitlicht werden muss. Die Konfigurationsvielfalt entsteht, da der Datenrekorder nicht allein der Aufzeichnung von CAN-Bus Zeitreihen dient. Darüber hinaus werden z.B. auch aggregierte Belastungsinformationen (Lastkollektive) gespeichert. Auf Grund der verschiedenen Zielsetzungen, die mit dem Datenrekorder verfolgt werden können, wird er mit unterschiedlichen Konfigurationen in Erprobungsfahrzeugen verbaut. In der gegenwärtigen Situation gibt es dadurch Signale des CAN-Busses, die nur bei einem bzw. wenigen Fahrzeugen aufgezeichnet werden. Im Hinblick auf den Einsatz der beschriebenen Data Mining Verfahren ist es zielführend, wenn alle relevanten Signale bei allen Fahrzeugen aufgezeichnet werden. In diesem Idealfall liefern temporale Muster und Regeln nicht nur Aussagen für ein spezielles Fahrzeug, sondern für die gesamte Flotte. Eine erneute studienartige Analyse der CAN-Bus Signale ist geplant, sobald die Datenaufzeichnung vereinheitlicht wurde.

Die Durchführung der Analyse des Datensatzes CAN-BUS zeigte weiterhin auf, wie die Suche nach temporalen Mustern in den Arbeitsprozess der Experten einzubetten ist. Die Experten können nur einige Aufzeichnungen zu einem Steuergerätfehler detailliert untersuchen. Wird

die Suche nach temporalen Mustern und Regeln vor Beginn der Arbeit des Experten durchgeführt, so können die gefundenen Ergebnisse dem Experten als Arbeitsgrundlage dienen. Die Unterstützung ist dadurch gegeben, dass der Experte bereits am Anfang seiner Arbeit Hinweise erhält, welche Signale, Muster und Aufzeichnungen für seine Diagnose maßgeblich sein können.

7.3 Customer Relationship Management

In der dritten Anwendung wird das Kaufverhalten aktiver Kunden untersucht.

Anwendungsanalyse

In den letzten Jahren führten die zunehmende Globalisierung und die gestiegene Produktvielfalt zu einem erhöhten Wettbewerbsdruck in der Automobilindustrie. In diesem Zusammenhang wird die Stärkung bestehender Kundenbeziehungen immer wichtiger, um sich einen Wettbewerbsvorteil zu erarbeiten. Die Bindung von Kunden an ein Unternehmen ist eines der Hauptziele des Customer Relationship Managements (CRM).

Unter Customer Relationship Management werden die Handlungen eines Unternehmens verstanden, die dazu dienen neue Kunden zu akquirieren, vorhandene Kunden weiter an das Unternehmen zu binden und Kunden zurückzugewinnen, die zu einem Wettbewerber gewechselt sind [Berry, 1983]. CRM lässt sich in zwei Bereiche unterteilen. Das operative CRM (oCRM) beinhaltet alle Aktivitäten, in denen das Unternehmen in direkten Kontakt mit Kunden tritt. Dazu gehören z.B. Werbekampagnen, Servicetelefone (Hotlines) oder Kundenclubs. Das analytische CRM (aCRM) hingegen beschäftigt sich mit der Analyse vorhandener Kundendaten, um operative CRM-Maßnahmen zielgerichtet durchführen zu können (vgl. z.B. [Arndt u. Gersten, 2001]).

Analytisches CRM ist häufig am Kundenlebenszyklus ausgerichtet (vgl. [Arndt u. Gersten, 2001] und [Berry u. Linoff, 2000, Seite 72ff.]). Der Kundenlebenszyklus aus Unternehmenssicht beginnt mit der Bedarfsidentifikation des potentiellen Kunden (z.B. durch Werbung). Entschließt sich der Kunde zum Kauf, so wird er zu einem aktiven Kunden. Im Falle von Fahrzeugen wird der Kunde nach einiger Zeit sein Fahrzeug ersetzen wollen. Hier besteht die Möglichkeit, dass der Kunde dem Unternehmen als aktiver Kunde erhalten bleibt (Loyalitätsschleife) oder als ehemaliger Kunde zu einem Wettbewerber abwandert. Die einzelnen Phasen des Kundenlebenszyklus sind in Abbildung 7.5 veranschaulicht. Abbildung 7.5 verdeutlicht darüber hinaus die Zuordnung der CRM Ziele zu den einzelnen Phasen durch das Akquisitions-, Loyalitäts- und Rückgewinnungsprogramm.

In dieser Anwendung soll das (Wieder-)Kaufverhalten aktiver Kunden untersucht werden, womit sich eine Zuordnung zum Loyalitätsprogramm ergibt. Ziel der Anwendung ist es, aus dem Kaufverhalten der Vergangenheit Informationen mit Hilfe temporaler Muster und Regeln zu identifizieren, welche die Unterstützung zukünftiger operativer CRM-Maßnahmen erlauben. Als Beispiel für diese Idee diene ein temporales Muster, das Kunden beschreibt, die sich nach dem Besitz einer C-Klasse für den Kauf einer höherwertigen E-Klasse entschieden haben (Upselling). Operatives CRM könnte diese Information nutzen, um vergleichbare C-Klasse Besitzer (und potentielle E-Klasse Käufer) im Rahmen einer Direktwerbung (Direktmarketing) gezielt anzusprechen. Das Erfolgskriterium dieser Anwendung besteht darin, ob sich aus dem historischen Kaufverhalten anwendbare temporale Muster und Regeln ableiten lassen.

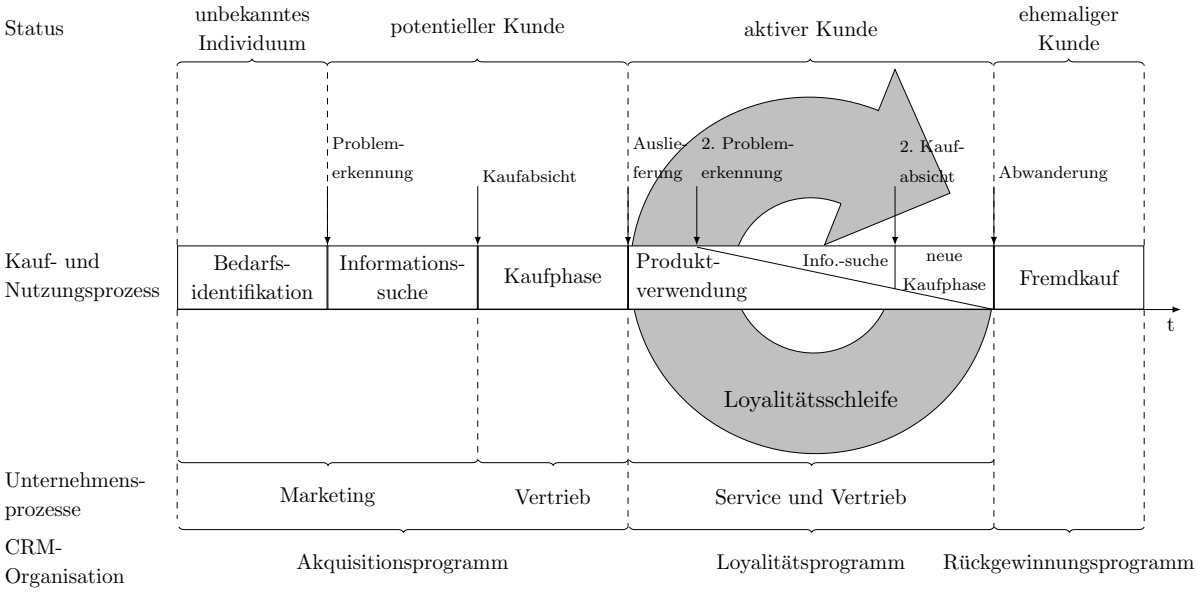


Abbildung 7.5: CRM als Prozess basierend auf dem Kundenlebenszyklus [Arndt u. Gersten, 2001]

Datenverständnis

Einen wesentlichen Bestandteil der Analyse bilden die Datenquellen zum Fahrzeugverkauf. Hier können auf Grund der Art des Verkaufs zwei verschiedene Datenquellen genutzt werden. Die Daten aus dem *Direktverkauf* beinhalten Informationen zum verkauften Fahrzeug (Fahrzeugidentifikationsnummer, Baureihe, Baumuster, Baujahr, etc.) sowie zum Käufer (Art der Bezahlung, Alter, Geschlecht, etc.). Dem Direktverkauf stehen die Daten aus der *Finanzdienstleistung* gegenüber. Hier wird das Fahrzeug nicht direkt verkauft, sondern über eine vertragliche Vereinbarung geleast oder finanziert. Dem entsprechend enthalten die Daten der Finanzdienstleistung neben Informationen zum Fahrzeug und zum Kunden weitere Informationen über die Vertragsart (Laufzeit, finanzierte Summe, etc.).

Die Informationen über die Käufer werden durch weitere Datenquellen noch ergänzt. Zu diesen Datenquellen gehören in regelmäßigen Abständen durchgeführte Umfragen zur Kundenzufriedenheit, Umfragen bei Leasing-Kunden vor dem Ende der Vertragszeit oder in Call-Centern bearbeitete Kundenanfragen.

Die vorhandenen Daten lassen sich in zwei Gruppen einteilen. Die erste Gruppe enthält Informationen über die einzelnen Kunden (Alter, Geschlecht, Umfrageergebnisse, etc.), während die zweite Gruppe, die gekauften Fahrzeuge beschreibt. Zu den Kundendaten existieren bereits Bewertungsmodelle, welche den monetären und nicht-monetären Wert eines Kunden (Customer Value bzw. Kundenwert) bestimmen (siehe auch [Arndt, 2007, Seite 293ff.]). Der monetäre Kundenwert ergibt sich aus dem Gewinn, der mit dem Kunden erzielt werden konnte. Der nicht-monetäre Kundenwert hingegen bewertet „alle positiven und negativen Einflüsse, die zwar die Attraktivität eines Konsumenten für das Unternehmen mitbestimmen, aber nicht

ohne weiteres in monetäre Wertgrößen zu transformieren sind“ [Wittkötter u. Steffen, 2002]. Bei der Bestimmung des nicht-monetären Kundenwertes werden z.B. das Feedback-Verhalten der Kunden bei Umfragen, die Wiederkaufabsicht, die Kaufhäufigkeit, die Kauffrequenz oder die Weiterempfehlungsabsicht berücksichtigt. Beide Kundenwerte müssen in die Analyse mit temporalen Mustern und Regeln aufgenommen werden, da sie wichtige Kennzahlen bei der Planung von operativen CRM-Maßnahmen darstellen.

Datenaufbereitung

Bei der Datenaufbereitung wurde für jeden Kunden der Datenbasis eine Intervallsequenz angelegt. Für jede Intervallsequenz gibt es allgemeine Informationen über den Kunden (Geschlecht, monetärer Kundenwert, nicht-monetärer Kundenwert), die durch temporale Intervalle ausgedrückt werden. Weitere temporale Intervalle beschreiben die Fahrzeugkäufe des Kunden. Hier sind vor allem der Fahrzeugtyp (z.B. Roadster, Limousine, Kombi, Coupé, SUV), die Baureihe (A-Klasse, C-Klasse, E-Klasse, etc.) und das Alter des Kunden beim Kauf von Interesse.

Die numerischen Merkmale (Alter und beide Kundenwerte) wurden anhand von Schwellwerten in definierte Wertebereiche diskretisiert (z.B. „Alter zwischen 40 und 50“). Abbildung 7.6 zeigt ein Beispiel für die Intervallsequenz eines männlichen Kunden mit drei gekauften Fahrzeugen. Nach dem ersten Kauf eines C-Klasse Coupés folgte eine C-Klasse Limousine und ein E-Klasse Kombi.

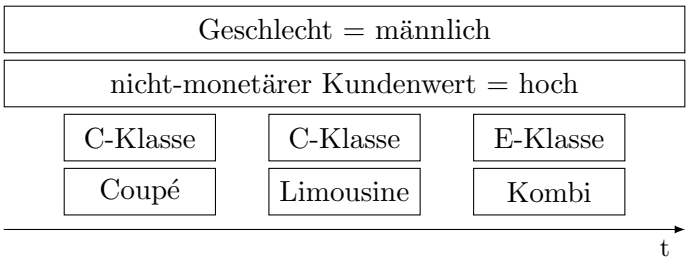


Abbildung 7.6: Beispiel einer Intervallsequenz für einen männlichen Kunden mit drei Fahrzeugen.

Die Datenaufbereitung wurde auch für den Datensatz CRM verwendet, welcher zur Evaluation der Algorithmen in Kapitel 5 diente. Der Datensatz CRM enthält 466 verschiedene Labels und 416 393 Intervallsequenzen. Insgesamt besteht der Datensatz aus ca. 6.3 Millionen temporalen Intervallen.

Data Mining

Die Extraktion der häufigen temporalen Muster erfolgte mit Hilfe der in Kapitel 4 vorgestellten Verfahren.

Wie bereits bei den zuvor beschriebenen Anwendungen können Muster innerhalb einer Intervallsequenz wiederholt auftreten. Ein Beispiel hierfür sind besonders markenloyale Kunden, deren Kaufhistorie zum Teil für die letzten 25 Jahre bekannt ist. Diese Kunden zeichnen sich

dadurch aus, dass sie wiederholt bestimmte Fahrzeugtypen erwerben.[4] Für die Anwendung ist es notwendig, dass mehrfach auftretende Muster in einer Intervallsequenz auch mehrfach berücksichtigt werden. Ursache hierfür ist, dass jedes gefundene Muster nicht nur die Eigenschaften der Käufer (d.h. ein bestimmtes Kundensegment) beschreibt, sondern zusätzlich den Verkauf von Fahrzeugen und somit Unternehmensumsatz repräsentiert. Die Lukrativität eines Kundensegmentes für operative CRM-Maßnahmen wird daher durch den Support der temporalen Muster korrekt wiedergegeben. Im Gegensatz dazu lassen Verfahren, die das Auftreten eines Musters nur einmal pro Intervallsequenz zählen (vgl. Verfahren in Abschnitt 3.2.2), dasselbe Kundensegment weniger attraktiv für CRM-Maßnahmen erscheinen. Daher sind die in Kapitel 4 beschriebenen Verfahren besser für die Untersuchung des Wiederkaufverhaltens geeignet.

Bewertung

Die Analyse des Wiederkaufverhaltens soll die Frage beantworten, welche Kunden bei der Durchführung einer operativen CRM-Maßnahme im Loyalitätsprogramm zu adressieren sind. Durch die häufigen temporalen Muster und Regeln des Datensatzes CRM konnten mehrere Kundensegmente identifiziert werden, die sich vorzugsweise für die Bewerbung bestimmter Baureihen (z.B. B-Klasse und M-Klasse) eignen. Insbesondere bei Direktwerbeaktionen ist die Kenntnis geeigneter Kundensegmente vorteilhaft, da ein effizienterer Einsatz des investierten Werbebudgets als bei zufällig ausgewählten Adressaten angenommen werden kann. Im Sinne der Anwendung ist daher der Einsatz temporaler Muster zur Analyse des Wiederkaufverhaltens von Kunden als Erfolg zu werten.

Umsetzung

Die Ergebnisse der Analysen wurden an die Experten des analytischen CRMs übergeben. Die identifizierten Kundensegmente unterstützen die Planung und Durchführung operativer CRM-Maßnahmen.

7.4 Zusammenfassung

In diesem Kapitel wurden drei Anwendungen für die Suche nach häufigen temporalen Mustern in der Automobilindustrie präsentiert. Obwohl alle drei Anwendungsprobleme aus unterschiedlichen Unternehmensbereichen (Produktbewährung, Elektronikentwicklung und Marketing) stammen, stellen sie dieselben Anforderungen an ein Lösungsverfahren.

- In einer Menge von Intervallsequenzen müssen alle häufigen temporalen Muster gefunden werden.
- Das mehrfache Auftreten eines temporalen Musters in einer Intervallsequenz muss berücksichtigt werden.

[4] Aus Sicht des CRM sind derartig loyale Kunden von besonderer Bedeutung, da sie häufig eine meinungsbildende Wirkung auf ihr Umfeld besitzen.

- Der Support (die Häufigkeit) eines temporalen Musters ist durch die Anzahl seiner Vorkommen in den Daten definiert.

Für die in der Produktbewährung angesiedelte Anwendung mussten zusätzlich zeitliche Bedingungen an die Vorkommen eines temporalen Musters beachtet werden.

Bereits existierende Verfahren zur Suche nach häufigen Mustern in intervallbasierten Daten (vgl. Abschnitt 3.2.2) erweisen sich für die genannten Anwendungen als ungeeignet, da sie nicht die multiplen Vorkommen eines temporalen Musters innerhalb einer Intervallsequenz zählen können. Ihr Einsatz beeinträchtigt die korrekte Interpretation der temporalen Muster im Kontext der jeweiligen Anwendung. Erst mit den in Kapitel 4 erarbeiteten Algorithmen konnten die Anwendungsprobleme zufrieden stellend adressiert werden.

Kapitel 8

Zusammenfassung und Ausblick

Im Folgenden werden die Ergebnisse und der Forschungsbeitrag der vorliegenden Arbeit zusammengefasst und ein Ausblick auf weiterführende Fragestellungen zur Entdeckung häufiger Muster in zeitbezogenen Daten gegeben.

8.1 Zusammenfassung

Den Anfang dieser Arbeit bildete eine Einführung in das Gebiet der Wissensentdeckung in Datenbanken. Dazu wurde der Wissensentdeckungsprozess mit Hilfe des CRISP-DM Prozessmodells (vgl. [Chapman u. a., 1999]) vorgestellt. Nachdem typische Data Mining-Aufgaben und Methoden besprochen wurden, folgte anschließend eine detaillierte Darstellung der Warenkorbanalyse sowie der Assoziationsregeln (siehe [Agrawal u. Srikant, 1994a]). Beide Data Mining-Methoden sind sowohl für diese Arbeit als auch für alle anderen Verfahren, die häufige Muster in zeitbezogenen Daten suchen, von grundlegender Bedeutung. In der Warenkorbanalyse wurde erstmals aufgezeigt, wie mit Hilfe des Apriori-Kriteriums die Suche nach allen häufigen Warenkörben effizient organisiert werden kann. Häufige Warenkörbe sind die Grundlage zum Ableiten von Assoziationsregeln. Mit Hilfe der Assoziationsregeln lassen sich Data Mining-Aufgaben wie z.B. Klassifikation, Konzeptbeschreibung, Abhängigkeitsanalyse oder Abweichungserkennung bearbeiten.

Im zweiten Teil der Grundlagen (Kapitel 3) wurden existierende Verfahren zur Entdeckung häufiger Muster in zeitbezogenen Daten vorgestellt und systematisiert. Die vorgeschlagene Systematik orientiert sich dabei an den Eingabedaten der Verfahren und unterscheidet zwischen ereignisbasierten und intervallbasierten Verfahren sowie zwischen Verfahren, bei denen sich die Eingabedaten aus einer einzelnen Eingabesequenz bzw. einer Vielzahl von Eingabesequenzen zusammensetzt. Die zugrunde liegende Idee besteht darin, dass Verfahren mit gleichen Eingabedaten, d.h. Verfahren derselben Kategorie, auch ein vergleichbares Problem lösen. Für jede der vier Kategorien der Systematik wurde die Arbeitsweise mehrerer Algorithmen zur Suche nach häufigen Mustern aufgezeigt und Unterschiede bzgl. der Hypothesensprachen, der Supportdefinitionen und der algorithmischen Lösungsstrategien besprochen. Die Untersuchung der verschiedenen Verfahren machte deutlich, dass keines der Verfahren für viele Eingabesequenzen in der Lage ist, multiple Vorkommen eines Musters innerhalb einer Eingabesequenz zu berücksichtigen. Das heißt, obwohl ein Muster mehrfach in einer Eingabesequenz vorkommen kann, wird der Supportzähler des Musters nur einmal inkrementiert. Im Gegensatz dazu müssen Verfahren, die nur mit einer Eingabesequenz arbeiten, das mehrfache Auftreten eines Musters beachten. Zu diesem Zweck haben sich die Supportdefinitionen *minimale Vorkommen*, *frequency* und *temporaler Support* etabliert. Eine Übertragung der verwendeten Ideen

zur Häufigkeitsbestimmung von Verfahren für eine Eingabesequenz auf Verfahren für viele Eingabesequenzen fand vor dieser Arbeit nicht statt. Vervollständigt wurde der Grundlagenteil durch eine Übersicht über Verfahren, die zeitbezogene Regeln direkt (d.h. ohne häufige Muster) aus den Eingabedaten extrahieren bzw. auf numerischen Zeitreihen arbeiten.

Kapitel 4 war der Entwicklung neuer Verfahren zur Entdeckung häufiger temporaler Muster gewidmet. Der Bedarf an neuen Verfahren entstand, da keines der existierenden Verfahren in der Lage ist, die folgenden Anforderungen der gegebenen industriellen Anwendungen (vgl. Kapitel 7) zu erfüllen:

- Die Eingabedaten bestehen aus einer Vielzahl von Intervallsequenzen.
- Das mehrfache Vorkommen eines Musters innerhalb einer Intervallsequenz wird bei der Bestimmung des Supports berücksichtigt.
- Der Support eines Musters basiert auf der Anzahl seiner Vorkommen in den Intervallsequenzen.

Kapitel 4 zergliedert sich in drei Teile. Im ersten Teil stand die formale Problemdefinition mit der Wahl einer geeigneten Hypothesensprache und einer Supportdefinition im Fokus. Die Vor- und Nachteile verschiedener Hypothesensprachen und Supportdefinitionen wurden eingehend beleuchtet. *Temporale Muster* auf Basis von Allens Intervallrelationen (siehe [Allen, 1983]) bildeten auf Grund ihrer Ausdrucksstärke die Hypothesensprache. Für die Supportdefinition hingegen wurde die Idee der *minimalen Vorkommen* (vgl. [Mannila u. a., 1995]) von ereignisbasierten auf intervallbasierte Daten übertragen. Minimale Vorkommen sind in besonderem Maße zur Bestimmung des Supports geeignet, da sie sowohl für einen unerfahrenen Anwender intuitiv verständlich sind, als auch mehrfache Vorkommen eines Musters innerhalb einer Intervallsequenz berücksichtigen können.

Im zweiten Teil wurden die Eigenschaften des definierten Problems ausführlich untersucht. Die Untersuchung zeigte auf, dass die Hypothesensprachen der Warenkorbanalyse und der Sequenzanalyse Spezialfälle von temporalen Mustern sind. In diesem Zusammenhang konnte die Größe der Hypothesenräume für die Warenkorb- und Sequenzanalyse sowie eine obere Schranke für die Größe des Hypothesenraumes temporaler Muster in Abhängigkeit von der maximalen Größe eines Musters angegeben werden. Im Mittelpunkt stand jedoch die Erkenntnis, dass durch die Berücksichtigung von mehrfachen Vorkommen eines temporalen Musters innerhalb einer Intervallsequenz das Apriori-Kriterium verletzt ist. Zwar kann mit Hilfe minimaler Vorkommen die Häufigkeit (der Support) eines temporalen Musters berechnet werden, aber es ist nicht mehr garantiert, dass alle Teilmuster eines häufigen Musters selbst häufig sind. Für den vorliegenden Fall konnte jedoch die Gültigkeit des *partiellen Apriori-Kriteriums* bewiesen werden. Das heißt, alle Suffixe eines häufigen Musters müssen selbst häufig sein. Infolgedessen lässt sich die Suche nach allen häufigen temporalen Mustern über die häufigen Suffixe organisieren. Dazu müssen zunächst alle kurzen häufigen Muster identifiziert werden. Anschließend werden nur solche langen Muster auf ihren Support untersucht, die die kürzeren häufigen Muster als Suffixe enthalten. Das partielle Apriori-Kriterium garantiert die Vollständigkeit dieser Strategie.

Im dritten Teil wurden die drei Algorithmen *FSMSet*, *FSMTree* und *Dip* vorgestellt. Alle drei Verfahren nutzen das partielle Apriori-Kriterium, um alle häufigen temporalen Muster zu

finden. *FSMSet* und *FSMTree* verwenden dazu eine Breitensuche anhand eines Schemas aus Kandidatengenerierung und Supportevaluation (Apriori-Ansatz). Die Supportevaluation wird von beiden Verfahren mit Hilfe endlicher Automaten realisiert. *FSMTree* stellt eine Verbesserung von *FSMSet* dar, da durch den Einsatz einer optimierten Datenstruktur (Präfixbaum) der algorithmische Aufwand reduziert werden konnte. Im Gegensatz zu *FSMSet* und *FSMTree* führt *Dip* eine Tiefensuche nach allen häufigen Mustern durch und findet die Instanzen eines temporalen Musters in den Intervallsequenzen nicht durch endliche Automaten, sondern durch die Verknüpfung von Instanzlisten. Des Weiteren besitzen alle drei Algorithmen die Fähigkeit, zeitliche Randbedingungen zu beachten. Durch zeitliche Randbedingungen kann die Gültigkeit der Vorkommen eines temporalen Musters einschränkt werden.

Eine Evaluation der Algorithmen in Bezug auf das Laufzeitverhalten, den Speicherbedarf, die Skalierbarkeit mit der Datensatzgröße und die Auswirkung zeitlicher Randbedingungen erfolgte in Kapitel 5. Die Evaluation nutzte sowohl reale Anwendungsdaten als auch künstliche Daten, die mit Hilfe eines parametrisierbaren Datensatzgenerators erzeugt wurden. Für die beiden Apriori basierten Algorithmen *FSMSet* und *FSMTree* zeigten die Experimente, dass *FSMTree* bzgl. Laufzeit und Speicherverhalten *FSMSet* überlegen ist. Der geringere Laufzeit- und Speicherbedarf von *FSMTree* ist auf den Einsatz effizienterer Datenstrukturen zur Speicherung von Kandidatenmustern zurückzuführen. Der Vergleich der Algorithmen *Dip* und *FSMTree* ergab ein komplexeres Verhalten bzgl. Laufzeit und Speicherbedarf. Je nach Wahl der minimalen Supportschranke kann *FSMSet* oder *Dip* die kürzeste Laufzeit und den geringsten Speicherbedarf aufweisen. Für die meisten Datensätze ist *FSMTree* effizienter (Laufzeit und Speicher), wenn eine hohe minimale Supportschranke gewählt wird. Bei sehr niedrigen Werten für die minimale Supportschranke ist hingegen *Dip* der leistungsfähigere Algorithmus. Das Verhalten der beiden Algorithmen erklärt sich durch die verwendeten Suchstrategien. Die Breitensuche von *FSMTree* erzeugt während der Kandidatengenerierung eine Vielzahl von Kandidatenmustern, die zum einen gespeichert werden müssen und zum anderen algorithmischen Aufwand verursachen. Die Anzahl der Kandidatenmuster steigt, je niedriger die gewählte minimale Supportschranke ist. Die Tiefensuche von *Dip* generiert keine Kandidaten und besitzt daher Vorteile bei niedrigen Supportschranken.

Neben der Effizienz wurde auch das Skalierungsverhalten der Algorithmen *FSMTree* und *Dip* untersucht. Die Experimente belegten, dass *FSMTree* besser mit einer zunehmenden Anzahl an Intervallsequenzen im Datensatz skaliert als *Dip*, wenn die minimale Supportschwelle relativ zur Gesamtanzahl der Intervallsequenzen definiert ist. Durch die relativ definierte Supportschwelle blieb in den Experimenten die Anzahl der häufigen Muster über verschieden große Datensätze hinweg (nahezu) konstant. Unter diesen Bedingungen benötigt *Dip* für größere Datensätze mehr Speicher, da es zu jedem Muster mehr Instanzen gibt, die in den Instanzlisten gehalten werden müssen. Die Verknüpfung längerer Instanzlisten wiederum verursacht längere Laufzeiten. Im Experiment zeigte sich ein positiv linearer Zusammenhang zwischen Datensatzgröße und Laufzeit. *FSMTree* hingegen besitzt einen konstanten Speicherbedarf für alle Datensätze, da die Anzahl der zu speichernden Kandidatenmuster für alle Datensätze gleich groß ist. Wie bereits bei *Dip* konnte auch für *FSMTree* ein linearer Zusammenhang zwischen Datensatzgröße und Laufzeit aufgezeigt werden. Der Zusammenhang erklärt sich durch den steigenden algorithmischen Aufwand für jede zusätzliche Intervallsequenz während der Supportevaluation. Der konstante Speicherbedarf von *FSMTree* ist herauszustellen, da er prinzipiell die Verarbeitung beliebig großer Datensätze erlaubt.

Zuletzt konnte anhand der Experimente nachgewiesen werden, dass *FSMTree* und *Dip* von der Existenz zeitlicher Randbedingungen profitieren und sowohl kürzere Laufzeiten erzielen als auch weniger Speicher benötigen. Beide Algorithmen verwenden die zeitlichen Randbedingungen, um die Menge der Kandidaten bzw. die Instanzlisten zu beschneiden. Je restriktiver die zeitlichen Randbedingungen sind, desto effizienter arbeiten *FSMTree* und *Dip*.

In Kapitel 6 wurden drei ergänzende Verfahren zu temporalen Mustern behandelt, die den Anwender bei der Bearbeitung eines Problems unterstützen. Der erste Teil betrachtete die Ableitung temporaler Regeln aus den häufigen Mustern sowie ihre Bewertung. Hier wurde insbesondere auf die Unterschiede zwischen temporalen Regeln und Assoziationsregeln eingegangen. Der größte Unterschied zeigt sich in der Bewertung der temporalen Regeln. Viele existierende Gütemaße für Assoziationsregeln sind auf Grund der gewählten Supportdefinition zur Bewertung von temporalen Regeln ungeeignet. Die Supportdefinition ermöglicht es zwar den Support eines Musters X ($\mathrm{Sup}(X)$) zu bestimmen, aber auf Grund der Berücksichtigung multipler Vorkommen von X innerhalb einer Intervallsequenz kann die Häufigkeit des Nichtauftretens von X ($\mathrm{Sup}(\overline{X})$) nicht semantisch konsistent angeben werden. Viele Gütemaße für Assoziationsregeln benötigen jedoch zur Bewertung einer Regel $X \Rightarrow Y$ den Wert $\mathrm{Sup}(\overline{X})$. Zur Bewertung von temporalen Regeln wurden daher zwei Wege vorgeschlagen. Der erste Weg bestand aus dem Gütemaß *Lift* für temporale Regeln. Der Lift bewertet eine temporale Regel anhand des Supports des Regel-, Prämissen- und Konklusionsmusters. Der zweite Weg enthält einen Simulationsansatz. Hier wird eine Vielzahl von Datensätzen künstlich erzeugt, die dieselben Eigenschaften aufweisen wie der Ausgangsdatensatz. Zur Bewertung einer temporalen Regel wird schließlich die Konfidenz der Regel auf dem Ausgangsdatensatz mit den Konfidenzwerten auf den künstlichen Datensätzen verglichen.

Im zweiten Teil stand die geeignete Visualisierung temporaler Muster und Regeln im Mittelpunkt. Es wurde ein Algorithmus angegeben, der zu einem beliebigen temporalen Muster eine wohlgeformte Instanz erzeugt. Wohlgeformt bedeutet in diesem Fall, dass die Abstände zwischen den einzelnen Zeitpunkten (Anfang und Ende der temporalen Intervalle) in der Instanz äquidistant sind. Mit Hilfe dieser Instanzen lassen sich temporale Muster und Regeln leicht visualisieren.

Im letzten Teil wurden zwei Möglichkeiten untersucht, wie statische Eigenschaften in die Mustersuche integriert werden können. Statische Eigenschaften beschreiben nicht-zeitbezogene Informationen wie z.B. das Geschlecht eines Kunden. Die beiden Alternativen unterscheiden sich darin, dass in der ersten Variante darauf verzichtet wird, Labels anhand einer Zuordnung in statische Eigenschaften und zeitbezogene Ereignisse zu trennen.

Kapitel 7 präsentierte drei Anwendungen zur Suche nach häufigen Mustern in zeitbezogenen Daten aus der Automobilindustrie. Die erste Anwendung beschrieb die Aufgabe, die Qualität produzierter Fahrzeuge im Feld zu überwachen. Zu diesem Zweck mussten zeitbezogene Informationen zu den einzelnen Fahrzeugen aus den Werkstätten (Reparatur- und Wartungsarbeiten) sowie die Konfiguration der Fahrzeuge (Baureihe, Motortyp, etc.) berücksichtigt werden. Die systematische Suche nach häufigen temporalen Mustern und Regeln unterstützt einen Ingenieur bei der Entscheidung, ob das gehäufte Auftreten eines Schadens durch frühere Schäden oder zurückliegende Reparatur- und Wartungsarbeiten bedingt ist, und kann so Qualitätsmängel aufdecken.

In der zweiten Anwendung sollte die Diagnose von Steuergerätfehlern in Entwicklungsfahrzeugen unterstützt werden. Die Datengrundlage der Anwendung bestand aus einer Menge von

Zeitreihen, die während verschiedener Erprobungsfahrten aufgezeichnet wurden. Die Zeitreihen beinhalteten wichtige Informationen zu den aufgetretenen Fahrsituationen (z.B. zu Geschwindigkeiten, Temperaturen, Drücken, Drehzahlen, etc.). Aus den Zeitreihen wurden im ersten Schritt Intervallsequenzen extrahiert, um nachfolgend die häufigen temporalen Muster zu finden. Das Ziel der Anwendung war es, temporale Zusammenhänge zwischen den Fahrsituationen und den eingetretenen Steuergerätfehlern aufzuzeigen.

In der letzten Anwendung wurde das Kaufverhalten von Kunden analysiert. Als Grundlage der Analyse dienten zum einen Informationen aus dem Fahrzeugverkauf (z.B. Baureihe und Ausstattung des Fahrzeugs) und zum anderen Angaben über den Kunden aus Umfragen (Alter, Geschlecht, etc.). Aus der Kaufhistorie der einzelnen Kunden wurden mit Hilfe temporaler Muster Kundensegmente identifiziert, die sich als Adressaten für Werbekampagnen bestimmter Baureihen eignen.

Obwohl die drei Anwendungen aus unterschiedlichen Geschäftsbereichen der Automobilindustrie stammen, besitzen sie einen gemeinsamen Problemkern: Die Suche nach häufigen temporalen Mustern in einer Menge von Intervallsequenzen. Des Weiteren erfordern die Anwendungen von einem Lösungsverfahren, dass multiple Vorkommen von Mustern innerhalb einer Intervallsequenz berücksichtigt werden und der Support eines Musters auf der Anzahl seiner Vorkommen in den Daten basiert. Bisherige Verfahren zur Mustersuche in zeitbezogenen Daten setzten diese Anforderungen nicht um. Erst mit Hilfe der neu entwickelten Verfahren dieser Arbeit konnten die Anwendungsprobleme sachgemäß gelöst werden.

8.2 Ausblick

Die Forschung auf dem Gebiet des temporalen Data Minings, insbesondere der Entdeckung häufiger Muster in zeitbezogenen Daten, hat sich im letzten Jahrzehnt schnell entwickelt. Daher ist abzusehen, dass die Forschung auch zukünftig eine Vielzahl von neuen Verfahren hervorbringen wird. Im Folgenden soll ein Ausblick auf weiterführende Fragestellungen der Entdeckung temporaler Muster und mögliche Erweiterungen dieser Arbeit gegeben werden.

Die Evaluation der Algorithmen in Kapitel 5 zeigte, dass die Laufzeiten der Algorithmen auf großen Datensätzen mehr als eine Stunde beträgt. Die Laufzeiten der Algorithmen sind Wartezeiten des Analysten. Es existiert daher ein Bedarf an weiteren, effizienteren Algorithmen zur Entdeckung aller häufigen temporalen Muster.

Neben der Entwicklung effizienterer Algorithmen besteht eine Möglichkeit zur Verkürzung der Laufzeiten darin, dass sich die Suche auf maximale oder geschlossene temporale Muster beschränkt. Die Mengen der maximalen und geschlossenen Muster sind jeweils eine Teilmenge aller häufigen Muster. Die Menge der maximalen Muster enthält hierbei nur Muster, die nicht Teilmuster eines anderen Musters der Menge sind. Die Idee maximaler Muster wurde für die Sequenzanalyse bereits in [Agrawal u. Srikant, 1995] verfolgt. Die Menge der geschlossenen Muster besteht aus der Teilmenge aller häufigen Muster, die nicht Teilmuster eines anderen Musters mit gleichem Support sind. Vor allem für geschlossene Muster existieren aus der Warenkorb- und Sequenzanalyse mehrere Algorithmen, die ihre Eigenschaften ausnutzen, um den Suchraum einzuschränken und kürzere Laufzeiten zu erzielen (siehe z.B. [Zaki u. Hsiao, 2002; Wang u. a., 2003; Wang u. Han, 2004; Tzvetkov u. a., 2003]). Der Ansatz geschlossener Muster ist direkt auf temporale Muster übertragbar.

In der vorliegenden Arbeit wurden mit dem Gütemaß Lift und einem Simulationsansatz zwei Wege vorgestellt, um temporale Regeln zu bewerten. Für Assoziationsregeln existieren jedoch eine Vielzahl an Gütemaßen (vgl. [Tan u. a., 2002]), deren semantisch konsistente Übertragung auf temporale Regeln weiterhin ein offenes Problem darstellt. Weitere Untersuchungen zu Gütemaßen für temporale Regeln erscheinen sinnvoll, da hier ein großes Potential zur Unterstützung eines Anwenders bestehet.

Die in der vorliegenden Arbeit vorgestellten neuen Algorithmen zur Suche nach häufigen temporalen Mustern waren durch drei Anwendungsbeispiele der Automobilindustrie motiviert. Voraussichtlich können mit den neuen Algorithmen auch Anwendungsprobleme außerhalb der Automobilbranche adressiert werden. Insbesondere die Möglichkeit, alle Vorkommen eines temporalen Musters bei der Supportbestimmung zu berücksichtigen, erschließt Anwendungsbereiche, die bisherigen Verfahren verschlossen blieben.

Zuletzt sei das aktive Forschungsgebiet der Analyse von Datenströmen (z.B. [Gama u. a., 2007]) genannt. Datenströme sind kontinuierliche Abfolgen von Daten. Die Entdeckung von häufigen temporalen Mustern in Datenströmen erfordert neue algorithmische Suchstrategien, da der Inhalt eines Datenstroms aufgrund begrenzter Ressourcen nur einmal gelesen werden darf. Alle bisherigen Algorithmen zur Suche nach häufigen intervallbasierten Mustern, gehen hingegen von einem uneingeschränkten Zugriff auf die Datengrundlage aus.

Literaturverzeichnis

[Agrawal u. a. 1993a] Agrawal, Rakesh ; Faloutsos, Christos ; Swami, Arun N.: Efficient Similarity Search In Sequence Databases. In: Lomet, D. B. (Hrsg.): *Foundations of Data Organization and Algorithms, 4th International Conference* Bd. 730, Springer, 1993, S. 69–84

[Agrawal u. a. 1993b] Agrawal, Rakesh ; Imielinski, Tomasz ; Swami, Arun: Mining Association Rules between Sets of Items in Large Databases. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1993, S. 207–216

[Agrawal u. a. 1995] Agrawal, Rakesh ; Lin, King-Ip ; Sawhney, Harpreet S. ; Shim, Kyuseok: Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases. In: **[Dayal u. a., 1995]**, S. 490–501

[Agrawal u. a. 1996] Agrawal, Rakesh ; Mannila, Heikki ; Srikant, Ramakrishnan ; Toivonen, Hannu ; Verkamo, Inkeri: Fast Discovery of Association Rules. In: *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996, S. 307–328

[Agrawal u. Srikant 1994a] Agrawal, Rakesh ; Srikant, Ramakrishnan: Fast Algorithms for Mining Association Rules. In: *Proceedings of the 20th International Conference on Very Large Databases*, 1994, S. 487–499

[Agrawal u. Srikant 1994b] Agrawal, Rakesh ; Srikant, Ramakrishnan: Mining Sequential Patterns. / IBM Research Division. 1994 (RJ9910). – Forschungsbericht

[Agrawal u. Srikant 1995] Agrawal, Rakesh ; Srikant, Ramakrishnan: Mining Sequential Patterns. In: Yu, P. S. (Hrsg.) ; Chen, A. L. P. (Hrsg.): *Proceedings of the Eleventh International Conference on Data Engineering*, IEEE Computer Society, 1995, S. 3–14

[Agresti 1990] Agresti, Alan: *Categorical Data Analysis*. Wiley, 1990 (Wiley series in probability and mathematical statistics)

[Aiello u. a. 2002] Aiello, Marco ; Monz, Christof ; Todoran, Leon: Document understanding for a broad class of documents. In: *International Journal on Document Analysis and Recognition* 5 (2002), Nr. 1, S. 1–16

[Allen 1983] Allen, James F.: Maintaining knowledge about temporal intervals. In: *Commun. ACM* 26 (1983), Nr. 11, S. 832–843

[Antunes u. Oliveira 2001] Antunes, Cláudia ; Oliveira, Arlindo: Temporal Data Mining: an overview. In: *KDD 2001 Workshop on Temporal Data Mining*, 2001

[Arndt 2007] Arndt, Dirk: *Customer Information Management. Ein Referenzmodell für die Informationsversorgung im Customer Relationship Management.*, Universität Stuttgart, Diss., 2007

[Arndt u. Gersten 2001] Arndt, Dirk ; Gersten, Wendy: Data Management in Analytical Customer Relationship Management. In: Gersten, W. (Hrsg.) ; Vanhoof, K. (Hrsg.): *Proceedings of the Workshop Data Mining for Marketing Applications at the ECML/PKDD 2001*, 2001, S. 25–38

[ASL] ASL: *American Sign Language Linguistic Research Project.* `http://www.bu.edu/asllrp`, Abruf: 01. Juni 2007. – Online Quelle

[Ayres u. a. 2002] Ayres, Jay ; Flannick, Jason ; Gehrke, Johannes ; Yiu, Tomi: Sequential Pattern mining using a bitmap representation. In: **[Hand u. a., 2002]**, S. 429–435

[Badaloni u. Giacomin 1999] Badaloni, Silvana ; Giacomin, Massimiliano: A Fuzzy Extension of Allen's Interval Algebra. In: Lamma, E. (Hrsg.) ; Mello, P. (Hrsg.): *Advances in Artificial Intelligence, 6th Congress of the Italian Association for Artificial Intelligence* Bd. 1792, Springer, 1999, S. 155–165

[Bellazzi u. a. 2000] Bellazzi, Riccardo ; Larizza, Cristiana ; Magni, Paolo ; Montani, Stefania ; Stefanelli, Mario: Intelligent analysis of clinical time series: an application in the diabetes mellitus domain. In: *Artificial Intelligence in Medicine* 20 (2000), Nr. 1, S. 37–57

[Bengio 1999] Bengio, Yoshua: Markovian models for sequential data. In: *Neural Computing Surveys* 2 (1999)

[Berndt u. Clifford 1996] Berndt, Donald J. ; Clifford, James: Finding Patterns in Time Series: A Dynamic Programming Approach. In: **[Fayyad u. a., 1996d]**, S. 229 – 248

[Berry 1983] Berry, Leonard T.: Relationship Marketing. In: Berry, L. T. (Hrsg.) ; Shostak, G. L. (Hrsg.) ; Upah, G. D. (Hrsg.): *Emerging Perspectives on Services Marketing.* Amer Marketing Assn, 1983, S. 25–28

[Berry u. Linoff 2000] Berry, Michael J.Ã. ; Linoff, Gordon S.: *Mastering Data Mining. The Art and Science of Customer Relationship Management.* Wiley, 2000

[Berthold u. Hand 2003] Berthold, Michael (Hrsg.) ; Hand, David J. (Hrsg.): *Intelligent Data Analysis. An Introduction.* 2. Edition. Springer, 2003

[Blumenstock u. a. 2006] Blumenstock, Axel ; Hipp, Jochen ; Kempe, Steffen ; Lanquillon, Carsten ; Wirth, Rüdiger: Interactivity Closes the Gap. Lessons Learned in an Automotive Industry Application. In: Ghani, R. (Hrsg.) ; Soares, C. (Hrsg.): *KDD 2006 Workshop on Data Mining and Business Applications*, 2006, S. 7 – 11

[Borgelt 2000] Borgelt, Christian: *Data Mining with Graphical Models*, Otto-von-Guericke Universität Magdeburg, Diss., 2000

[Borgelt 2002] Borgelt, Christian: *Graphical Models: Methods for Data Analysis and Mining.* Wiley, 2002

[Bosch 1996] Bosch, Karl: *Großes Lehrbuch der Statistik.* R. Oldenburg Verlag, 1996

[Brachman u. Anand 1996] Brachman, Ronald J. ; Anand, Tej: The Process of Knowledge Discovery in Databases. In: **[Fayyad u. a., 1996d]**, S. 37–57

[Bradley 2003] Bradley, Elizabeth: Intelligent Data Analysis. An Introduction. In: **[Berthold u. Hand, 2003]**, Kapitel Analysis of Time Series, S. 199 – 227

[Brazma u. a. 1998a] Brazma, Alvis ; Jonassen, Inge ; Vilo, Jaak ; Ukkonen, Esko: Pattern Discovery in Biosequences. In: Honavar, V. (Hrsg.) ; Slutzki, G. (Hrsg.): *Grammatical Inference, 4th International Colloquium* Bd. 1433, Springer, 1998, S. 257–270

[Brazma u. a. 1998b] Brazma, Alvis ; Jonassen, Inge ; Vilo, Jaak ; Ukkonen, Esko: Predicting gene regulatory elements from their expression data in the complete yeast genome. In: *German Conference on Bioinformatics*, 1998

[Breiman u. a. 1984] Breiman, Leo u. a.: *Classification and Regression Trees.* New York : Chapman & Hall, 1984

[Buxbaum 1992] Buxbaum, Otto: *Betriebsfestigkeit. Sichere und wirtschaftliche Bemessung schwingbruchgefährdeter Bauteile.* Stahleisen, 1992

[Cadez u. a. 2000] Cadez, Igor V. ; Gaffney, Scott ; Smyth, Padhraic: A general probabilistic framework for clustering individuals and objects. In: *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, S. 140–149

[Cao u. a. 2005] Cao, Huiping ; Mamoulis, Nikos ; Cheung, David W.: Mining Frequent Spatio-Temporal Sequential Patterns. In: *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005, S. 82–89

[Casas-Garriga 2005] Casas-Garriga, Gemma: Summarizing Sequential Data with Closed Partial Orders. In: *Proceedings of the 2005 SIAM International Data Mining Conference*, 2005, S. 380–392

[Cass 2007] Cass, Tony: CASTOR2 rises to LHC's data storage challenge. In: *CERN Courier* 47 (2007), November, Nr. 9

[Chapman u. a. 1999] Chapman, Pete ; Clinton, Julian ; Kerber, Randy ; Khabaza, Thomas ; Reinartz, Thomas ; Shearer, Colin ; Wirth, Rüdiger: *CRISP-DM 1.0 Step-by-step data mining guide*, 1999. `http://www.crisp-dm.org`

[Chen u. a. 2005] Chen, Gong ; Wu, Xindong ; Zhu, Xingquan: Sequential Pattern Mining in Multiple Streams. In: *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005, S. 585–588

[Clark u. Niblett 1989] Clark, Peter ; Niblett, Tim: The CN2 Induction Algorithm. In: *Machine Learning* 3 (1989), S. 261–283

[Cohen 2001] Cohen, Paul R.: Fluent Learning: Elucidating the Structure of Episodes. In: **[Hoffmann u. a., 2001]**, S. 268–277

[Cohen 1995] Cohen, William W.: Fast Effective Rule Induction. In: *Proceedings of the 12th International Conference on Machine Learning*, 1995, S. 115–123

[Das u. a. 1998] Das, Gautam ; Lin, King-Ip ; Mannila, Heikki ; Renganathan, Gopal ; Smyth, Padhraic: Rule Discovery from Time Series. In: *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, 1998, S. 16–22

[Dasgupta u. Forrest 1996] Dasgupta, Dipankar ; Forrest, Stephanie: Novelty Detection in Time Series Data using Ideas from Immunology. In: *Proceedings of The International Conference on Intelligent Systems*, 1996

[Dayal u. a. 1995] Dayal, Umeshwar (Hrsg.) ; Gray, Peter M. D. (Hrsg.) ; Nishio, Shojiro (Hrsg.): *Proceedings of 21th International Conference on Very Large Data Bases*. Morgan Kaufmann, 1995

[Dubois u. a. 2003] Dubois, D. ; HadjAli, A. ; Prade, H.: Fuzziness and Uncertainty in Temporal Reasoning. In: *Journal of Universal Computer Science* 9 (2003), Nr. 9, S. 1168–1194

[Eichinger u. a. 2006] Eichinger, Frank ; Nauck, Detlef D. ; Klawonn, Frank: Sequence Mining for Customer Behaviour Predictions in Telecommunications. In: Ackermann, M. (Hrsg.) ; Soares, C. (Hrsg.) ; Guidemann, B. (Hrsg.): *Proceedings of the ECML/PKDD Workshop on Practical Data Mining*, 2006, S. 3–10

[El-Sayed u. a. 2004] El-Sayed, Maged ; Ruiz, Carolina ; Rundensteiner, Elke A.: FS-Miner: efficient and incremental mining of frequent sequence patterns in web logs. In: Laender, A. H. F. (Hrsg.) ; Lee, D. (Hrsg.) ; Ronthaler, M. (Hrsg.): *6th ACM CIKM International Workshop on Web Information and Data Management*, ACM, 2004, S. 128–135

[Everitt 1992] Everitt, Brian S.: *The Analysis of Contingency Tables*. 2. Chapman & Hall, 1992 (Monographs on Statistics and Applied Probability 45)

[Everitt 1993] Everitt, Brian S.: *Cluster Analysis*. John Wiley and Sons, 1993

[Faloutsos u. a. 1994] Faloutsos, Christos ; Ranganathan, M. ; Manolopoulos, Yannis: Fast Subsequence Matching in Time-Series Databases. In: Snodgrass, R. T. (Hrsg.) ; Winslett, M. (Hrsg.): *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, ACM Press, 1994, S. 419–429

[Fayyad u. a. 1996a] Fayyad, Usama M. ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: From Data Mining to Knowledge Discovery: An Overview. In: **[Fayyad u. a., 1996d]**, S. 1–34

[Fayyad u. a. 1996b] Fayyad, Usama M. ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: From Data Mining to Knowledge Discovery in Databases. In: *AI Magazine* 17 (1996), Nr. 3, S. 37–54

[Fayyad u. a. 1996c] Fayyad, Usama M. ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: *Commun. ACM* 39 (1996), Nr. 11, S. 27–34

[Fayyad u. a. 1996d] Fayyad, Usama M. (Hrsg.) ; Piatetsky-Shapiro, Gregory (Hrsg.) ; Smyth, Padhraic (Hrsg.) ; Uthurusamy, Ramasamy (Hrsg.): *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996

[Ferreira u. Azevedo 2005] Ferreira, Pedro G. ; Azevedo, Paulo J.: Protein Sequence Classification Through Relevant Sequence Mining and Bayes Classifiers. In: Bento, C. (Hrsg.) ; Cardoso, A. (Hrsg.) ; Dias, G. (Hrsg.): *Progress in Artificial Intelligence, 12th Portuguese Conference on Artificial Intelligence* Bd. 3808, Springer, 2005, S. 236–247

[Friedman u. Fisher 1999] Friedman, Jerome H. ; Fisher, Nicholas I.: Bump hunting in high-dimensional data. In: *Statistics and Computing* 9 (1999), Nr. 2, S. 123–143

[Fürnkranz 1999] Fürnkranz, Johannes: Seperate-and-Conquer Rule Learning / Austrian Research Institute for Artificial Intelligence. 1999 (OEFAI-TR-96-25). – Forschungsbericht

[Gama u. a. 2007] Gama, João (Hrsg.) ; Gaber, Mohamed M. (Hrsg.) ; Aguilar-Ruiz, Jesús S. (Hrsg.): *Proceedings of the International Workshop on Knowledge Discovery from Ubiquitous Data Streams.* 2007

[Garofalakis u. a. 1999] Garofalakis, Minos N. ; Rastogi, Rajeev ; Shim, Kyuseok: SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. In: Atkinson, M. P. (Hrsg.) ; Orlowska, M. E. (Hrsg.) ; Valduriez, P. (Hrsg.) ; Zdonik, S. B. (Hrsg.) ; Brodie, M. L. (Hrsg.): *Proceedings of 25th International Conference on Very Large Data Bases*, Morgan Kaufmann, 1999, S. 223–234

[Garofalakis u. a. 2002] Garofalakis, Minos N. ; Rastogi, Rajeev ; Shim, Kyuseok: Mining Sequential Patterns with Regular Expression Constraints. In: *IEEE Trans. Knowl. Data Eng.* 14 (2002), Nr. 3, S. 530–552

[Geurts 2001] Geurts, Pierre: Pattern Extraction for Time Series Classification. In: **[Raedt u. Siebes, 2001]**, S. 115–127

[Gillies 2001] Gillies, James: Green light for massive increase in computing power for LHC data. In: *CERN Courier* 41 (2001), November, Nr. 9

[Guil u. a. 2005] Guil, Francisco ; Bailón, Antonio B. ; Bosch, Alfonso ; Marín, Roque: An Iterative Method for Mining Frequent Temporal Patterns. In: Moreno-Díaz, R. (Hrsg.) ; Pichler, F. (Hrsg.) ; Quesada-Arencibia, A. (Hrsg.): *10th International Conference on Computer Aided Systems Theory* Bd. 3643, Springer, 2005, S. 189–198

[Guimarães u. Ultsch 1999] Guimarães, Gabriela ; Ultsch, Alfred: A Method for Temporal Knowledge Conversion. In: Hand, D. J. (Hrsg.) ; Kok, J. N. (Hrsg.) ; Berthold, M. R. (Hrsg.): *Advances in Intelligent Data Analysis, 3rd International Symposium* Bd. 1642, Springer, 1999, S. 369–382

[Han u. a. 2007] Han, Jiawei ; Cheng, Hong ; Xin, Dong ; Yan, Xifeng: Frequent pattern mining: current status and future directions. In: *Data Min. Knowl. Discov.* 15 (2007), Nr. 1, S. 55–86

[Han u. a. 2001] Han, Jiawei ; Jamil, Hasan M. ; Lu, Ying ; Chen, Liangyou ; Liao, Yaqin ; Pei, Jian: DNA-Miner: A System Prototype for Mining DNA Sequences. In: *Proceedings of the ACM SIGMOD Conference*, 2001, S. 618

[Han u. a. 2000a] Han, Jiawei ; Pei, Jian ; Mortazavi-Asl, Behzad ; Chen, Qiming ; Dayal, Umeshwar ; Hsu, Mei-Chun: FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000, S. 355–359

[Han u. a. 2000b] Han, Jiawei ; Pei, Jian ; Yin, Yiwen: Mining Frequent Patterns without Candidate Generation. In: Chen, W. (Hrsg.) ; Naughton, J. F. (Hrsg.) ; Bernstein, P. A. (Hrsg.): *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ACM, 2000, S. 1–12

[Hand u. a. 2002] Hand, David (Hrsg.) ; Keim, Daniel (Hrsg.) ; Ng, Raymond (Hrsg.): *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2002

[Harms u. Deogun 2004] Harms, Sherri K. ; Deogun, Jitender S.: Sequential Association Rule Mining with Time Lags. In: *J. Intell. Inf. Syst.* 22 (2004), Nr. 1, S. 7–22

[Harms u. a. 2001] Harms, Sherri K. ; Deogun, Jitender S. ; Saquer, Jamil ; Tadesse, Tsegaye: Discovering Representative Episodal Association Rules from Event Sequences Using Frequent Closed Episode Sets and Event Constraints. In: Cercone, N. (Hrsg.) ; Lin, T. Y. (Hrsg.) ; Wu, X. (Hrsg.): *Proceedings of the 2001 IEEE International Conference on Data Mining*, IEEE Computer Society, 2001, S. 603–606

[Harms u. a. 2002] Harms, Sherri K. ; Deogun, Jitender S. ; Tadesse, Tsegaye: Discovering Sequential Association Rules with Constraints and Time Lags in Multiple Sequences. In: Hacid, M.-S. (Hrsg.) ; Ras, Z. W. (Hrsg.) ; Zighed, D. A. (Hrsg.) ; Kodratoff, Y. (Hrsg.): *Foundations of Intelligent Systems, 13th International Symposium* Bd. 2366, Springer, 2002, S. 432–441

[Hätönen u. a. 1996] Hätönen, Kimmo ; Klemettinen, Mika ; Mannila, Heikki ; Ronkainen, Pirjo ; Toivonen, Hannu: Knowledge Discovery from Telecommunication Network Alarm Databases. In: Su, S. Y. W. (Hrsg.): *Proceedings of the 12th International Conference on Data Engineering*, IEEE Computer Society, 1996, S. 115–122

[Heckerman u. a. 1995] Heckerman, David ; Geiger, Dan ; Chickering, David M.: Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. In: *Machine Learning* 20 (1995), Nr. 3, S. 197–243

[Hegland 2000] Hegland, Markus: Computational challenges in data mining. In: *ANZIAM Journal* 42(E) (2000), S. C1–C43

[Heuer u. Saake 1995] Heuer, Andreas ; Saake, Gunter: *Datenbanken: Konzepte und Sprachen.* International Thompson Publishing, 1995

[Hipp 2003] Hipp, Jochen: *Wissensentdeckung in Datenbanken mit Assoziationsregeln. Verfahren zur effizienten Regelgenerierung und deren Integration in den Wissensentdeckungsprozeß*, Eberhard-Karls-Universität Tübingen, Diss., 2003

[Hipp u. a. 2001] Hipp, Jochen ; Güntzer, Ulrich ; Grimmer, Udo: Data Quality Mining - Making a Virute of Necessity. In: *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001

[Hipp u. a. 2000] Hipp, Jochen ; Güntzer, Ulrich ; Nakhaeizadeh, Gholamreza: Algorithms for Association Rule Mining - A General Survey and Comparison. In: *SIGKDD Explorations* 2 (2000), Nr. 1, S. 58–64

[Hipp u. a. 2002] Hipp, Jochen ; Güntzer, Ulrich ; Nakhaeizadeh, Gholamreza: Data Mining of Association Rules and the Process of Knowledge Discovery in Databases. In: Perner, P. (Hrsg.): *Industrial Conference on Data Mining* Bd. 2394, Springer, 2002, S. 15–36

[Hoffmann u. a. 2001] Hoffmann, Frank (Hrsg.) ; Hand, David J. (Hrsg.) ; Adams, Niall M. (Hrsg.) ; Fisher, Douglas H. (Hrsg.) ; Guimarães, Gabriela (Hrsg.): *Advances in Intelligent Data Analysis, 4th International Conference*. Bd. *2189*. Springer, 2001

[Höppner 2002] Höppner, Frank: Time Series Abstraction Methods - A Survey. In: Schubert, S. E. (Hrsg.) ; Reusch, B. (Hrsg.) ; Jesse, N. (Hrsg.): *GI Jahrestagung* Bd. 19, GI, 2002, S. 777–786

[Höppner 2001] Höppner, Frank: Discovery of Temporal Patterns. Learning Rules about the Qualitative Behaviour of Time Series. In: [**Raedt u. Siebes, 2001**], S. 192–203

[Höppner 2002a] Höppner, Frank: *Knowledge Discovery from Sequential Data*, TU Braunschweig, Diss., 2002

[Höppner 2002b] Höppner, Frank: Learning Dependencies in Multivariate Time Series. In: *Proceedings of the ECAI'02 Workshop on Knowledge Discovery in (Spatio-) Temporal Data*, 2002, S. 25 – 31

[Höppner u. Klawonn 2001] Höppner, Frank ; Klawonn, Frank: Finding Informative Rules in Interval Sequences. In: [**Hoffmann u. a., 2001**], S. 125–134

[Höppner u. Klawonn 2002] Höppner, Frank ; Klawonn, Frank: Finding informative rules in interval sequences. In: *Intelligent Data Analysis* 6 (2002), Nr. 3, S. 237–255

[Huber 2004] Huber, Robert: Entwicklung der Bussysteme / Elektroniksystem- und Logistik-GmbH. 2004. – Forschungsbericht

[Joshi u. a. 2001] Joshi, Mahesh V. ; Agarwal, Ramesh C. ; Kumar, Vipin: Mining Needle in a Haystack: Classifying Rare Classes via Two-phase Rule Induction. In: *Proceedings of the ACM SIGMOD 2001 Conference*, 2001, S. 91–102

[Kam u. Fu 2000] Kam, P.-S. ; Fu, Ada Wai-Chee: Discovering Temporal Patterns for Interval-Based Events. In: Kambayashi, Y. (Hrsg.) ; Mohania, M. K. (Hrsg.) ; Tjoa, A. M.

(Hrsg.): *Data Warehousing and Knowledge Discovery, 2nd International Conference* Bd. 1874, Springer, 2000, S. 317–326

[Kao u. a. 2005] Kao, Ben ; Zhang, Minghua ; Yip, Chi L. ; Cheung, David W. ; Fayyad, Usama M.: Efficient Algorithms for Mining and Incremental Update of Maximal Frequent Sequences. In: *Data Mining and Knowledge Discovery* 10 (2005), Nr. 2, S. 87–116

[Kass 1980] Kass, G. V.: An Exploratory Technique for Investigating Large Quantities of Categorical Data. In: *Applied Statistics* 29 (1980), S. 119–127

[Kempe u. Hipp 2006] Kempe, Steffen ; Hipp, Jochen: Mining Sequences of Temporal Intervals. In: Fürnkranz, J. (Hrsg.) ; Scheffer, T. (Hrsg.) ; Spiliopoulou, M. (Hrsg.): *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases* Bd. 4213, Springer, 2006, S. 569–576

[Kempe u. a. 2008a] Kempe, Steffen ; Hipp, Jochen ; Kruse, Rudolf: FSMTree: An Efficient Algorithm for Mining Frequent Temporal Patterns. In: Preisach, C. (Hrsg.) ; Burkhardt, H. (Hrsg.) ; Schmidt-Thieme, L. (Hrsg.) ; Decker, R. (Hrsg.): *Data Analysis, Machine Learning and Applications*, Springer, 2008, S. 253–260

[Kempe u. a. 2008b] Kempe, Steffen ; Hipp, Jochen ; Lanquillon, Carsten ; Kruse, Rudolf: Mining Frequent Temporal Patterns in Interval Sequences. Zur Veröffentlichung angenommen in: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (2008)

[Kempe u. Kruse 2008] Kempe, Steffen ; Kruse, Rudolf: Mining Temporal Patterns in an Automotive Environment. In: Magdalena, L. (Hrsg.) ; Ojeda-Aciego, M. (Hrsg.) ; Verdegay, J.L̃. (Hrsg.): *Proceedings of the 12th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2008, S. 521–528

[Keogh 1997] Keogh, Eamonn: A Fast and Robust Method for Pattern Matching in Time Series Databases. In: *Proceedings of 9th International Conference on Tools with Artificial Intelligence*, 1997, S. 578 – 584

[Keogh u. a. 2004a] Keogh, Eamonn ; Chu, Selina ; Hart, David ; Pazzani, Michael: Segmenting Time Series: A Survey and Novel Approach. In: Last, M. (Hrsg.) ; Kandel, A. (Hrsg.) ; Bunke, H. (Hrsg.): *Data Mining in Time Series Databases* Bd. 57. World Scientific, 2004, S. 1–22

[Keogh u. a. 2001] Keogh, Eamonn J. ; Chakrabarti, Kaushik ; Mehrotra, Sharad ; Pazzani, Michael J.: Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In: *Proceedings of the ACM SIGMOD 2001 Conference*, 2001, S. 151–162

[Keogh u. Kasetty 2002] Keogh, Eamonn J. ; Kasetty, Shruti: On the need for time series data mining benchmarks: a survey and empirical demonstration. In: [**Hand u. a., 2002**], S. 102–111

[Keogh u. Kasetty 2003] Keogh, Eamonn J. ; Kasetty, Shruti: On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In: *Data Min. Knowl. Discov.* 7 (2003), Nr. 4, S. 349–371

[Keogh u. a. 2002] Keogh, Eamonn J. ; Lonardi, Stefano ; Chiu, Bill Y.: Finding surprising patterns in a time series database in linear time and space. In: **[Hand u. a., 2002]**, S. 550–556

[Keogh u. a. 2004b] Keogh, Eamonn J. ; Lonardi, Stefano ; Ratanamahatana, Chotirat: Towards parameter-free data mining. In: Kim, W. (Hrsg.) ; Kohavi, R. (Hrsg.) ; Gehrke, J. (Hrsg.) ; DuMouchel, W. (Hrsg.): *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2004, S. 206–215

[Keogh u. Pazzani 1999] Keogh, Eamonn J. ; Pazzani, Michael J.: Scaling up Dynamic Time Warping to Massive Dataset. In: Zytkow, J. M. (Hrsg.) ; Rauch, J. (Hrsg.): *Principles of Data Mining and Knowledge Discovery, 3rd European Conference* Bd. 1704, Springer, 1999, S. 1–11

[Klösgen 2002] Klösgen, Willi: Decision Rules. In: **[Klösgen u. Zytkow, 2002a]**, S. 277 – 282

[Klösgen u. Zytkow 2002a] Klösgen, Willi (Hrsg.) ; Zytkow, Jan M. (Hrsg.): *Handbook of Data Mining and Knowledge Discovery.* Oxford University Press, 2002

[Klösgen u. Zytkow 2002b] Klösgen, Willi ; Zytkow, Jan M.: The Knowledge Discovery Process. In: **[Klösgen u. Zytkow, 2002a]**, S. 10 – 21

[Krahl u. a. 1998] Krahl, Daniela ; Windheuser, Ulrich ; Zick, Friedrich-Karl: *Data Mining. Einsatz in der Praxis.* Addison-Wesley, 1998

[Last u. a. 2001] Last, Mark ; Klein, Yaron ; Kandel, Abraham: Knowledge discovery in time series databases. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 31 (2001), Nr. 1, S. 160–169

[Lavrac u. a. 2004] Lavrac, Nada ; Kavsek, Branko ; Flach, Peter A. ; Todorovski, Ljupco: Subgroup Discovery with CN2-SD. In: *Journal of Machine Learning Research* 5 (2004), S. 153–188

[Lawrenz 2007] Lawrenz, Wolfhard: *CAN Controller Area Network.* Hüthig, 2007

[Lesh u. a. 2000] Lesh, Neal ; Zaki, Mohammed J. ; Ogihara, Mitsunori: Scalable Feature Mining for Sequential Data. In: *IEEE Intelligent Systems* 15 (2000), Nr. 2, S. 48–56

[Li u. Biswas 2000] Li, Cen ; Biswas, Gautam: A Bayesian Approach to Temporal Data Clustering using Hidden Markov Models. In: Langley, P. (Hrsg.): *Proceedings of the 17th International Conference on Machine Learning*, Morgan Kaufmann, 2000, S. 543–550

[Li u. a. 2000] Li, Yingjiu ; Wang, Xiaoyang S. ; Jajodia, Sushil: Discovering Temporal Patterns in Multiple Granularities. In: **[Roddick u. Hornsby, 2001]**, S. 5–19

[Lin u. a. 2002] Lin, Jessica ; Keogh, Eamonn ; Lonardi, Stefano ; Patel, Pranav: Finding Motifs in Time Series. In: Hand, D. (Hrsg.) ; Keim, D. (Hrsg.) ; Ng, R. (Hrsg.): *Workshop on Temporal Data Mining, 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002

[Lin u. a. 2003] Lin, Jessica ; Keogh, Eamonn J. ; Lonardi, Stefano ; Chiu, Bill Y.: A symbolic representation of time series, with implications for streaming algorithms. In: Zaki, M. J. (Hrsg.) ; Aggarwal, C. C. (Hrsg.): *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery.*, ACM, 2003, S. 2–11

[Lin u. a. 2004a] Lin, Jessica ; Keogh, Eamonn J. ; Lonardi, Stefano ; Lankford, Jeffrey P. ; Nystrom, Donna M.: VizTree: a Tool for Visually Mining and Monitoring Massive Time Series Databases. In: Nascimento, M. A. (Hrsg.) ; Özsu, M. T. (Hrsg.) ; Kossmann, D. (Hrsg.) ; Miller, R. J. (Hrsg.) ; Blakeley, J. A. (Hrsg.) ; Schiefer, K. B. (Hrsg.): *Proceedings of the 13th International Conference on Very Large Data Bases*, Morgan Kaufmann, 2004, S. 1269–1272

[Lin u. a. 2004b] Lin, Jessica ; Vlachos, Michail ; Keogh, Eamonn J. ; Gunopulos, Dimitrios: Iterative Incremental Clustering of Time Series. In: Bertino, E. (Hrsg.) ; Christodoulakis, S. (Hrsg.) ; Plexousakis, D. (Hrsg.) ; Christophides, V. (Hrsg.) ; Koubarakis, M. (Hrsg.) ; Böhm, K. (Hrsg.) ; Ferrari, E. (Hrsg.): *Advances in Database Technology, 9th International Conference on Extending Database Technology* Bd. 2992, Springer, 2004, S. 106–122

[Lin u. Lee 2005] Lin, Ming-Yen ; Lee, Suh-Yin: Fast Discovery of Sequential Patterns through Memory Indexing and Database Partitioning. In: *J. Inf. Sci. Eng.* 21 (2005), Nr. 1, S. 109–128

[MacQueen 1967] MacQueen, J.B̃.: Some methods of classification and analysis of multivariate observations. In: *Proceedings of the 5th Berkeley Symposium on Mathemtical Statistics and Probability*, 1967, S. 281–297

[Maimon u. Last 2000] Maimon, Oded ; Last, Mark: *Knowledge Discovery and Data Mining: The Info-fuzzy Network Methodology.* Kluwer Academic Publishers, 2000

[Mannila u. Toivonen 1996] Mannila, Heikki ; Toivonen, Hannu: Discovering Generalized Episodes Using Minimal Occurrences. In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, S. 146–151

[Mannila u. a. 1995] Mannila, Heikki ; Toivonen, Hannu ; Verkamo, A. I.: Discovering Frequent Episodes in Sequences. In: *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, 1995, S. 210–215

[Mannila u. a. 1997] Mannila, Heikki ; Toivonen, Hannu ; Verkamo, A. I.: Discovery of Frequent Episodes in Event Sequences. In: *Data Mining and Knowledge Discovery* 1 (1997), Nr. 3, S. 259–289

[Mannila u. a. 1994] Mannila, Heikki ; Toivonen, Hannu ; Verkamo, Inkeri: Efficient Algorithms for Discovering Association Rules. In: *AAAI Workshop on Knowledge Discovery in Databases*, 1994, S. 181–192

[Masseglia u. a. 1998] Masseglia, Florent ; Cathala, Fabienne ; Poncelet, Pascal: The PSP Approach for Mining Sequential Patterns. In: Zytkow, J. M. (Hrsg.) ; Quafafou,

M. (Hrsg.): *Principles of Data Mining and Knowledge Discovery, 2nd European Symposium* Bd. 1510, Springer, 1998, S. 176–184

[Sun Microsystems 2007a] Sun Microsystems: *Java 2 Platform Standard Edition 5.0 API Specification*. Version: 2007. `http://java.sun.com/j2se/1.5.0/docs/api/index.html`, Abruf: 17. November 2007. – Online Quelle

[Sun Microsystems 2007b] Sun Microsystems: *JDK Tools and Utilities*. Version: 2007. `http://java.sun.com/j2se/1.5.0/docs/tooldocs/index.html`, Abruf: 17. November 2007. – Online Quelle

[Sun Microsystems 2007c] Sun Microsystems: *Tuning Garbage Collection with the 5.0 Java Virtual Machine*. Version: 2007. `http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html`, Abruf: 5. Dezember 2007. – Online Quelle

[McCreight 1976] McCreight, Edward M.: A Space-Economical Suffix Tree Construction Algorithm. In: *J. ACM* 23 (1976), Nr. 2, S. 262–272

[McGarry 2005] McGarry, Ken: A survey of interestingness measures for knowledge discovery. In: *The Knowledge Engineering Review* 20 (2005), Nr. 1, S. 39–61

[Mitchell 1997] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill, 1997

[Mooney u. Roddick 2004] Mooney, Carl ; Roddick, John F.: Mining Relationships Between Interacting Episodes. In: Berry, M. W. (Hrsg.) ; Dayal, U. (Hrsg.) ; Kamath, C. (Hrsg.) ; Skillicorn, D. B. (Hrsg.): *Proceedings of the 4th SIAM International Conference on Data Mining*, SIAM, 2004

[Mörchen u. a. 2004] Mörchen, Fabian ; Ultsch, Alfred ; Hoos, Olaf: Discovering Interpretable Muscle Activation Patterns with the Temporal Data Mining Method. In: Boulicaut, J. (Hrsg.) ; Esposito, F. (Hrsg.) ; Giannotti, F. (Hrsg.) ; Pedreschi, D. (Hrsg.): *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases* Bd. 3202, Springer, 2004, S. 512–514

[Mörchen u. a. 2005] Mörchen, Fabian ; Ultsch, Alfred ; Hoos, Olaf: Extracting interpretable muscle activation patterns with time series knowledge mining. In: *Int. J. Know.-Based Intell. Eng. Syst.* 9 (2005), Nr. 3, S. 197–208

[Mörchen 2003] Mörchen, Fabian: Time series feature extraction for data mining using DWT and DFT / Department of Mathematics and Computer Science, Philipps-University Marburg. 2003 (33). – Forschungsbericht

[Mörchen 2006a] Mörchen, Fabian: Algorithms for time series knowledge mining. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM Press, 2006, S. 668–673

[Mörchen 2006b] Mörchen, Fabian: A better tool than Allen's relations for expressing temporal knowledge in interval data. In: *Theory and Practice of Temporal Data Mining (TPTDM 2006) — Workshop of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, S. 25–34

[Mörchen 2006c] Mörchen, Fabian: *Time Series Knowledge Mining*, Philipps-University Marburg, Diss., 2006

[Mörchen 2007] Mörchen, Fabian: Unsupervised Pattern Mining from Symbolic Temporal Data. In: *SIGKDD Explorations* 9 (2007), Juli, Nr. 1, S. 41 – 54

[Mörchen u. Ultsch 2007] Mörchen, Fabian ; Ultsch, Alfred: Efficient mining of understandable patterns from multivariate interval time series. In: *Data Mining and Knowledge Discovery* 15 (2007), Nr. 2, S. 181–215

[Nakhaeizadeh 1998] Nakhaeizadeh, Gholamreza (Hrsg.): *Data Mining. Theoretische Aspekte und Anwendungen.* Physika-Verlag, 1998

[Nakhaeizadeh u. a. 1998] Nakhaeizadeh, Gholamreza ; Reinartz, Thomas ; Wirth, Rüdiger: Wissensentdeckung in Datenbanken und Data Mining: Ein Überblick. In: **[Nakhaeizadeh, 1998]**, S. 1 – 33

[Nauck u. a. 1996] Nauck, Detlef ; Klawonn, Frank ; Kruse, Rudolf: *Neuronale Netze und Fuzzy-Systeme.* Wiesbaden : Vieweg, 1996 (Series Artificial Intelligence)

[Oates u. Cohen 1996] Oates, Tim ; Cohen, Paul R.: Searching for Structure in Multiple Streams of Data. In: *Proceedings of the 13th International Conference on Machine Learning*, 1996, S. 346–354

[Oates u. a. 1999] Oates, Tim ; Firoiu, Laura ; Cohen, Paul R.: Clustering Time Series with Hidden Markov Models and Dynamic Time Warping. In: *Proceedings of the IJCAI-99 Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, 1999, S. 17 – 21

[Oates u. a. 1998] Oates, Tim ; Jensen, David ; Cohen, Paul R.: Discovering Rules for Clustering and Predicting Asynchronous Events. In: *AAAI Workshop on Predicting the Future: AI Approaches to Time Series Analysis*, 1998, S. 73 – 79

[Oates u. a. 1997a] Oates, Tim ; Schmill, Matthew ; Jensen, David ; Cohen, Paul R.: A family of algorithms for finding temporal structure in data. In: *In 6th International Workshop on AI and Statistics*, 1997, S. 371–378

[Oates u. a. 1997b] Oates, Tim ; Schmill, Matthew D. ; Cohen, Paul R.: Parallel and Distributed Search for Structure in Multivariate Time Series. In: Someren, M. (Hrsg.) ; Widmer, G. (Hrsg.): *Proceedings of the 9th European Conference on Machine Learning* Bd. 1224, Springer, 1997, S. 191–198

[Papapetrou u. a. 2006a] Papapetrou, Panagiotis ; Benson, Gary ; Kollios, George: Discovering Frequent Poly-Regions in DNA Sequences. In: *Workshops Proceedings of the 6th IEEE International Conference on Data Mining*, IEEE Computer Society, 2006, S. 94–98

[Papapetrou u. a. 2006b] Papapetrou, Panagiotis ; Benson, Gary ; Kollios, George: Discovering Frequent Poly-Regions in DNA Sequences / Boston University Computer Science. 2006 (Technical Report No. 2006-027). – Forschungsbericht

[Papapetrou u. a. 2005] Papapetrou, Panagiotis ; Kollios, George ; Sclaroff, Stan ; Gunopulos, Dimitrios: Discovering Frequent Arrangements of Temporal Intervals. In: *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005

[Parthasarathy u. a. 1999] Parthasarathy, Srinivasan ; Zaki, Mohammed J. ; Ogihara, Mitsunori ; Dwarkadas, Sandhya: Incremental and Interactive Sequence Mining. In: *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management*, ACM, 1999, S. 251–258

[Patel u. a. 2002] Patel, Pranav ; Keogh, Eamonn J. ; Lin, Jessica ; Lonardi, Stefano: Mining Motifs in Massive Time Series Databases. In: *Proceedings of the 2002 IEEE International Conference on Data Mining*, 2002, S. 370–377

[Pearl 1988] Pearl, Judea: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988

[Pei u. a. 2001] Pei, Jian ; Han, Jiawei ; Mortazavi-Asl, Behzad ; Pinto, Helen ; Chen, Qiming ; Dayal, Umeshwar ; Hsu, Meichun: PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In: *Proceedings of the 17th International Conference on Data Engineering*, IEEE Computer Society, 2001, S. 215–224

[Pei u. a. 2004] Pei, Jian ; Han, Jiawei ; Mortazavi-Asl, Behzad ; Wang, Jianyong ; Pinto, Helen ; Chen, Qiming ; Dayal, Umeshwar ; Hsu, Meichun: Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. In: *IEEE Trans. Knowl. Data Eng.* 16 (2004), Nr. 11, S. 1424–1440

[Perng u. a. 2000] Perng, Chang-Shing ; Wang, Haixun ; Zhang, Sylvia R. ; Jr., Douglas Stott P.: Landmarks: a New Model for Similarity-based Pattern Querying in Time Series Databases. In: *Proceedings of the 16th International Conference on Data Engineering*, 2000, S. 33–42

[Piatetsky-Shapiro 2002] Piatetsky-Shapiro, Gregory: *KDnuggets Polls: What main methodology are you using for data mining?* Version: 2002. `http://www.kdnuggets.com/polls/2002/methodology.htm`, Abruf: 07. August 2007. – Online Quelle

[Piatetsky-Shapiro 2004] Piatetsky-Shapiro, Gregory: *KDnuggets Polls: Data Mining Methodology?* Version: 2004. `http://www.kdnuggets.com/polls/2004/data_mining_methodology.htm`, Abruf: 07. August 2007. – Online Quelle

[Piatetsky-Shapiro 2007] Piatetsky-Shapiro, Gregory: *KDnuggets Polls: What main methodology are you using for data mining?* Version: 2007. `http://www.kdnuggets.com/polls/2007/data_mining_methodology.htm`, Abruf: 15. Oktober 2007. – Online Quelle

[Plantevit u. a. 2005] Plantevit, Marc ; Choong, Yeow W. ; Laurent, Anne ; Laurent, Dominique ; Teisseire, Maguelonne: M^2SP: Mining Sequential Patterns Among Several Dimensions. In: *9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2005, S. 205–216

[Quinlan 1993] Quinlan, J. R.: *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993

[Quinlan u. Cameron-Jones 1995] Quinlan, J. R. ; Cameron-Jones, R. M.: Induction of Logic Programs: FOIL and Related Systems. In: *New Generation Comput.* 13 (1995), Nr. 3&4, S. 287–312

[Rabiner 1989] Rabiner, Lawrence R.: A tutorial on hidden Markov models and selected applications in speech recognition. In: *Proceedings of the IEEE* 77 (1989), Nr. 2, S. 257–286

[Raedt u. Siebes 2001] Raedt, Luc D. (Hrsg.) ; Siebes, Arno (Hrsg.): *Principles of Data Mining and Knowledge Discovery, 5th European Conference.* Bd. *2168.* Springer, 2001

[Rafiei u. Mendelzon 1997] Rafiei, Davood ; Mendelzon, Alberto O.: Similarity-Based Queries for Time Series Data. In: Peckham, Joan (Hrsg.): *Proceedings ACM SIGMOD 1997 International Conference on Management of Data*, ACM Press, 1997, S. 13–25

[Reinartz 1999] Reinartz, Thomas: *Focusing Solutions for Data Mining. Analytical Studies and Experimental Results in Real-World Domains.* Springer, 1999

[Riddle u. a. 1994] Riddle, Patricia J. ; Segal, Richard ; Etzioni, Oren: Representation design and brut-force induction in a Boeing manufacturing domain. In: *Applied Artificial Intelligence* 8 (1994), Nr. 1, S. 125–147

[Roddick u. Hornsby 2001] Roddick, John F. (Hrsg.) ; Hornsby, Kathleen (Hrsg.): *Temporal, Spatial, and Spatio-Temporal Data Mining, 1st International Workshop.* Bd. *2007.* Springer, 2001

[Roddick u. a. 2000] Roddick, John F. ; Hornsby, Kathleen ; Spiliopoulou, Myra: An Updated Bibliography of Temporal, Spatial, and Spatio-temporal Data Mining Research. In: **[Roddick u. Hornsby, 2001]**, S. 147–164

[Roddick u. Mooney 2005] Roddick, John F. ; Mooney, Carl: Linear Temporal Sequences and Their Interpretation Using Midpoint Relationships. In: *IEEE Trans. Knowl. Data Eng.* 17 (2005), Nr. 1, S. 133–135

[Roddick u. Spiliopoulou 1999] Roddick, John F. ; Spiliopoulou, Myra: A Bibliography of Temporal, Spatial and Spatio-Temporal Data Mining Research. In: *SIGKDD Explorations* 1 (1999), Nr. 1, S. 34–38

[Roddick u. Spiliopoulou 2002] Roddick, John F. ; Spiliopoulou, Myra: A Survey of Temporal Knowledge Discovery Paradigms and Methods. In: *IEEE Trans. Knowl. Data Eng.* 14 (2002), Nr. 4, S. 750–767

[Sahar 1999] Sahar, Sigal: Interestingness via What is Not Interesting. In: *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, S. 332–336

[Savary u. Zeitouni 2005] Savary, Lionel ; Zeitouni, Karine: Indexed Bit Map (IBM) for Mining Frequent Sequences. In: Jorge, A. (Hrsg.) ; Torgo, L. (Hrsg.) ; Brazdil, P. (Hrsg.) ; Camacho, R. (Hrsg.) ; Gama, J. (Hrsg.): *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases* Bd. 3721, Springer, 2005, S. 659–666

[Schockaert u. a. 2006] Schockaert, Steven ; Cock, Martine D. ; Kerre, Etienne E.: An Efficient Characterization of Fuzzy Temporal Interval Relations. In: *IEEE World Congress on Computational Intelligence (FUZZ-IEEE)*. Vancouver, Canada : IEEE Computational Intelligence Society, July 2006, S. 9026 – 9033

[Shah u. a. 1999] Shah, Devavrat ; Lakshmanan, Laks V. S. ; Ramamritham, Krithi ; Sudarshan, S.: Interestingness and Pruning of Mined Patterns. In: *1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999

[Smyth 1999] Smyth, Padhraic: Probabilistic model-based clustering of multivariate and sequential data. In: Heckerman, D. (Hrsg.) ; Whittaker, J. (Hrsg.): *Proceedings of the Seventh International Workshop on AI and Statistics*, 1999

[Spiliopoulou u. a. 1999] Spiliopoulou, Myra ; Faulstich, Lukas C. ; Winkler, Karsten: A Data Miner Analyzing the Navigational Behaviour of Web Users. In: *Proceedings of the Workshop on Machine Learning in User Modelling of the ACAI'99 International Conference*, 1999

[Srikant u. Agrawal 1995a] Srikant, Ramakrishnan ; Agrawal, Rakesh: Mining Generalized Association Rules. In: **[Dayal u. a., 1995]**, S. 407–419

[Srikant u. Agrawal 1995b] Srikant, Ramakrishnan ; Agrawal, Rakesh: Mining Sequential Patterns: Generalizations and Performance Improvements. / IBM Research Division. 1995 (RJ9994). – Forschungsbericht

[Srikant u. Agrawal 1996] Srikant, Ramakrishnan ; Agrawal, Rakesh: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers, P. M. G. (Hrsg.) ; Bouzeghoub, M. (Hrsg.) ; Gardarin, G. (Hrsg.): *Proceedings of the 5th International Conference on Extending Database Technology* Bd. 1057, Springer, 1996, S. 3–17

[Tan u. Kumar 2000] Tan, Pang-Ning ; Kumar, Vipin: Interestingness Measures for Association Patterns: A Perspective / University of Minnesota - Computer Science and Engineering. 2000 (00-036). – Forschungsbericht

[Tan u. a. 2002] Tan, Pang-Ning ; Kumar, Vipin ; Srivastava, Jaideep: Selecting the Right Interestingness Measure for Association Patterns. In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002, S. 32–41

[Tzvetkov u. a. 2003] Tzvetkov, Petre ; Yan, Xifeng ; Han, Jiawei: TSP: Mining Top-K Closed Sequential Patterns. In: *Proceedings of the 3rd IEEE International Conference on Data Mining*, IEEE Computer Society, 2003, S. 347–354

[Tzvetkov u. a. 2005] Tzvetkov, Petre ; Yan, Xifeng ; Han, Jiawei: TSP: Mining top-k closed sequential patterns. In: *Knowl. Inf. Syst.* 7 (2005), Nr. 4, S. 438–457

[Ukkonen 1995] Ukkonen, Esko: On-Line Construction of Suffix Trees. In: *Algorithmica* 14 (1995), Nr. 3, S. 249–260

[Ultsch 2004] Ultsch, Alfred: Unification-based Temporal Grammar / Philipps-Universität Marburg. 2004 (Technical Report 37). – Forschungsbericht

[Vilain u. a. 1989] Vilain, Marc ; Kautz, Henry ; Beek, Peter van: Constraint propagation algorithms for temporal reasoning: a revised report. (1989), S. 373–381

[Villafane u. a. 1999] Villafane, Roy ; Hua, Kien A. ; Tran, Duc A. ; Maulik, Basab: Mining Interval Time Series. In: Mohania, M. K. (Hrsg.) ; Tjoa, A. M. (Hrsg.): *Data Warehousing and Knowledge Discovery, 1st International Conference* Bd. 1676, Springer, 1999, S. 318–330

[Villafane u. a. 2000] Villafane, Roy ; Hua, Kien A. ; Tran, Duc A. ; Maulik, Basab: Knowledge Discovery from Series of Interval Events. In: *J. Intell. Inf. Syst.* 15 (2000), Nr. 1, S. 71–89

[Vilo 1998] Vilo, Jaak: Discovering Frequent Patterns from Strings / Department of Computer Science, University of Helsinki. 1998 (C-1998-9). – Forschungsbericht

[Wagner 1994] Wagner, Klaus W.: *Einführung in die Theoretische Informatik – Grundlagen und Modelle.* Springer-Verlag, 1994

[Wang u. Han 2004] Wang, Jianyong ; Han, Jiawei: BIDE: Efficient Mining of Frequent Closed Sequences. In: *Proceedings of the 20th International Conference on Data Engineering*, IEEE Computer Society, 2004, S. 79–90

[Wang u. a. 2003] Wang, Jianyong ; Han, Jiawei ; Pei, Jian: CLOSET+: searching for the best strategies for mining frequent closed itemsets. In: Getoor, L. (Hrsg.) ; Senator, T. E. (Hrsg.) ; Domingos, P. (Hrsg.) ; Faloutsos, C. (Hrsg.): *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2003, S. 236–245

[Webb 1995] Webb, Geoffrey I.: OPUS: An Efficient Admissible Algorithm for Unordered Search. In: *Journal of Artificial Intelligence Research* 3 (1995), S. 431–465

[Weiss u. Indurkhya 1998] Weiss, Sholom M. ; Indurkhya, Nitin: *Predictive Data Mining. A Practical Guide.* Morgan Kaufman Publishers, 1998

[Williams u. Huang 1996] Williams, Graham J. ; Huang, Zhexue: Modelling the KDD Process / CSIRO Division of Information Technology. 1996 (TR DM 96013). – Forschungsbericht

[Winarko u. Roddick 2005] Winarko, Edi ; Roddick, John F.: Discovering Richer Temporal Association Rules from Interval-Based Data. In: Tjoa, A. M. (Hrsg.) ; Trujillo, J. (Hrsg.): *Data Warehousing and Knowledge Discovery, 7th International Conference* Bd. 3589, Springer, 2005, S. 315–325

[Winarko u. Roddick 2007] Winarko, Edi ; Roddick, John F.: ARMADA - An Algorithm for Discovering Richer Relative Temporal Association Rules from Interval-based Data. In: *Data and Knowledge Engineering* (2007), S. to appear

[Wittkötter u. Steffen 2002] Wittkötter, Meike ; Steffen, Marion: Customer Value als Basis des CRM. In: Ahlert, D. (Hrsg.) ; Becker, J. (Hrsg.) ; Knackstedt, R. (Hrsg.) ; Wunderlich, M. (Hrsg.): *Customer Relationship Management im Handel*. Springer, 2002, S. 73–83

[Xiong u. Yeung 2002] Xiong, Yimin ; Yeung, Dit-Yan: Mixtures of ARMA Models for Model-Based Time Series Clustering. In: *Proceedings of the 2002 IEEE International Conference on Data Mining*, 2002, S. 717–720

[Xiong u. Yeung 2004] Xiong, Yimin ; Yeung, Dit-Yan: Time series clustering with ARMA mixtures. In: *Pattern Recognition* 37 (2004), Nr. 8, S. 1675–1689

[Yan u. a. 2003] Yan, Xifeng ; Han, Jiawei ; Afshar, Ramin: CloSpan: Mining Closed Sequential Patterns in Large Databases. In: Barbará, D. (Hrsg.) ; Kamath, C. (Hrsg.): *Proceedings of the 3rd SIAM International Conference on Data Mining*, SIAM, 2003

[Yang u. a. 2001] Yang, Jiong ; Wang, Wei ; Yu, Philip S.: Infominer: mining surprising periodic patterns. In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, S. 395–400

[Yang u. a. 2002] Yang, Jiong ; Wang, Wei ; Yu, Philip S.: InfoMiner+: Mining Partial Periodic Patterns with Gap Penalties. In: *Proceedings of the 2002 IEEE International Conference on Data Mining*, 2002, S. 725–728

[Yang u. Wu 2006] Yang, Qiang ; Wu, Xindong: 10 Challenging Problems in Data Mining Research. In: *International Journal of Information Technology and Decision Making* 5 (2006), Nr. 4, S. 597–604

[Yi u. Faloutsos 2000] Yi, Byoung-Kee ; Faloutsos, Christos: Fast Time Sequence Indexing for Arbitrary Lp Norms. In: Abbadi, A. E. (Hrsg.) ; Brodie, M. L. (Hrsg.) ; Chakravarthy, S. (Hrsg.) ; Dayal, U. (Hrsg.) ; Kamel, N. (Hrsg.) ; Schlageter, G. (Hrsg.) ; Whang, K.-Y. (Hrsg.): *Proceedings of 26th International Conference on Very Large Data Bases*, Morgan Kaufmann, 2000, S. 385–394

[Yi u. a. 1998] Yi, Byoung-Kee ; Jagadish, H. V. ; Faloutsos, Christos: Efficient Retrieval of Similar Time Sequences Under Time Warping. In: *Proceedings of the 14th International Conference on Data Engineering*, IEEE Computer Society, 1998, S. 201–208

[Zaki 1998] Zaki, Mohammed J.: Efficient Enumeration of Frequent Sequences. In: Gardarin, G. (Hrsg.) ; F., J. C. (Hrsg.) ; Pissinou, N. (Hrsg.) ; Makki, K. (Hrsg.) ; Bouganim, L. (Hrsg.): *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management*, ACM, 1998, S. 68–75

[Zaki 2000] Zaki, Mohammed J.: Sequence Mining in Categorical Domains: Incorporating Constraints. In: *Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management*, ACM, 2000, S. 422–429

[Zaki 2001a] Zaki, Mohammed J.: Sequence Mining in Categorical Domains: Algorithms and Applications. In: Sun, R. (Hrsg.) ; Giles, C. L. (Hrsg.): *Sequence Learning - Paradigms, Algorithms, and Applications* Bd. 1828, Springer, 2001, S. 162–187

[Zaki 2001b] Zaki, Mohammed J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. In: *Machine Learning* 42 (2001), Nr. 1/2, S. 31–60

[Zaki u. Hsiao 2002] Zaki, Mohammed J. ; Hsiao, Ching-Jiu: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: Grossman, R. L. (Hrsg.) ; Han, J. (Hrsg.) ; Kumar, V. (Hrsg.) ; Mannila, H. (Hrsg.) ; Motwani, R. (Hrsg.): *Proceedings of the 2nd SIAM International Conference on Data Mining*, SIAM, 2002

[Zaki u. Hsiao 2005] Zaki, Mohammed J. ; Hsiao, Ching-Jui: Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. In: *IEEE Trans. Knowl. Data Eng.* 17 (2005), Nr. 4, S. 462–478

[Zaki u. a. 1998] Zaki, Mohammed J. ; Lesh, Neal ; Ogihara, Mitsunori: PlanMine: Sequence Mining for Plan Failures. In: *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998, S. 369–374

[Zaki u. a. 2000] Zaki, Mohammed J. ; Lesh, Neal ; Ogihara, Mitsunori: PlanMine: Predicting Plan Failures Using Sequence Mining. In: *Artif. Intell. Rev.* 14 (2000), Nr. 6, S. 421–446

[Zaki u. a. 1997] Zaki, Mohammed J. ; Parthasarathy, Srinivasan ; Ogihara, Mitsunori ; Li, Wei: New Algorithms for Fast Discovery of Association Rules. In: *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1997, S. 283–286

[Zell 1994] Zell, Andreas: *Simulation Neuronaler Netze.* Oldenbourg, 1994

[Zhao u. Bhowmick 2003] Zhao, Qiankun ; Bhowmick, Sourav S.: Sequential Pattern Mining: A Survey / Nanyang Technological University. 2003 (2003118). – Forschungsbericht

Lebenslauf

Persönliche Daten

Name	Steffen Kempe
Anschrift	Trollingerweg 17 89075 Ulm
E-Mail	Steffen.Kempe@daimler.com
geboren am	29.05.1979 in Magdeburg
Familienstand	ledig
Staatsangehörigkeit	deutsch

Bildungsweg

10/1998 – 09/2003	Studium der Informatik an der Otto-von-Guericke-Universität Magdeburg mit Nebenfachausbildung Logistik Abschlüsse: 25.04.2002: Bakkalaureus der Informatik, mit Auszeichnung 24.09.2003: Diplom-Informatiker, mit Auszeichnung
09/1997 – 08/1998	Ableistung der allgemeinen Wehrpflicht in Fürstenau
09/1991 – 07/1997	Besuch des Börde-Gymnasiums Wanzleben Abschluss: Allgemeine Hochschulreife
09/1985 – 07/1991	Besuch der Polytechnischen Oberschule Johannes-R.-Becher Klein Wanzleben

Arbeitserfahrung

seit 03/2008	Daimler AG, Group Research, Böblingen Wissenschaftlicher Mitarbeiter in der Abteilung Qualitätsanalysen
04/2005 – 02/2008	Daimler AG, Group Research, Ulm, Doktorand
10/2003 – 03/2005	DaimlerChrysler AG, Research and Technology, Ulm Wissenschaftlicher Mitarbeiter in der Abteilung Data Mining Solutions