Lernen häufiger Muster aus intervallbasierten Datenströmen - Semantik und Optimierungen

Dennis Geesen[1], H.-Jürgen Appelrath[1], Marco Grawunder[1], Daniela Nicklas[2]

[1]Informationssysteme,[2]Datenbank- und Internettechnologien
Department für Informatik, Universität Oldenburg, 26121 Oldenburg
{dennis.geesen, appelrath, marco.grawunder, daniela.nicklas}@uni-oldenburg.de

Abstract: Das Erkennen und Lernen von Mustern über Ereignisdatenströmen ist eine wesentliche Voraussetzung für effektive kontextbewusste Anwendungen, wie sie bspw. in intelligenten Wohnungen (Smart Homes) vorkommen. Zur Erkennung dieser Muster werden i.d.R. Verfahren aus dem Bereich des Frequent Pattern Mining (FPM) eingesetzt. Das Erlernen relevanter Muster findet aktuell entweder auf aufgezeichneten Ereignisströmen statt oder wird online mit Hilfe spezieller, an die Besonderheiten der Stromverarbeitung angepasste FPM-Algorithmen durchgeführt. Auf diese Weise muss entweder auf die Onlineverarbeitung verzichtet oder existierende und bewährte effiziente FPM-Algorithmen können nicht eingesetzt werden. In diesem Beitrag stellen wir einen Ansatz vor, der es ermöglicht, beliebige Datenbank-basierte FPM-Algorithmen ohne Anpassung auch auf Datenströmen durchzuführen. Da unsere Semantik auf der bekannten relationalen Algebra basiert, können weitere Optimierungen bspw. durch Anfrageumschreibungen erfolgen. Wir evaluieren den Ansatz im Datenstrom-Framework Odysseus und zeigen, dass bspw. beim Einsatz des FPM-Algorithmus „FP-Growth“ das Lernen in konstanter Zeit erfolgen kann und somit ein kontinuierliches Lernen auf dem Datenstrom möglich ist.

1 Einleitung

Durch den zunehmenden Einsatz von Sensoren und Aktoren in intelligenten Wohnungen oder Alltagsgegenständen werden neuartige Anwendungen ermöglicht, die individuell und situativ handeln und bestimmte Aktionen nicht nur basierend auf Sensorschwellwerten auslösen. Eine bestimmte Lampe wird nicht eingeschaltet, weil es dunkel ist, sondern es wird zusätzlich berücksichtigt, ob der Bewohner diese Lampe in der aktuellen Situation (dem Kontext) überhaupt verwenden würde. Solche kontextbewussten Anwendungen (vgl. [DAS01]) basieren i.d.R. auf einer Fusion von Sensordaten und müssen sich an die verschiedenen Umgebungen, Personen und Einstellungen anpassen, sie erlernen können. Aus zusammen auftretenden Aktionen, die entweder durch den Bewohner ausgelöst werden, wie z.B. das Betätigen eines Schalters, oder durch vorliegende äußere Umstände, z.B. die aktuelle Temperatur und Helligkeit können anschließend Regeln, wie z.B. *wenn der Fernseher an ist und es draußen dunkel, dann schalte die Ecklampe an und die Deckenlampe aus* erzeugt werden. Wurde eine solche Regel erlernt, kann sie danach von der Anwendung verwendet werden, um bspw. das Licht im Raum fernsehgerecht zu gestalten.

Das Problem solcher lernender Anwendungen ist zu unterscheiden, wann Aktionen nur zufällig zusammen auftreten und wann sie so häufig sind, dass sie mit hoher Wahrscheinlichkeit eine Vorliebe des Bewohners sind. In der explorativen Analyse statischer Daten wird dazu Frequent Pattern Mining (FPM) [HCXY07] eingesetzt, um häufige Muster zu finden. Dessen Algorithmen sind jedoch nicht auf die Verarbeitung potentiell unendlicher Sensordaten ausgelegt, sodass die Umsetzung meist funktionsspezifisch in monolithischen Anwendungen erfolgt. Entsprechend werden sie separat implementiert und sind daher selten wiederverwendbar und aufwändig anpassbar. Des Weiteren werden gleiche Algorithmen häufig auch parallel ausgeführt, selbst wenn sie auf denselben Sensordaten arbeiten, was sich negativ auf Rechenzeit und Speicherverbrauch auswirkt.

Für eine wiederverwendbare, flexible und universell einsetzbare Sensordatenfusion wurden Datenstrommanagementsysteme (DSMS) als Technologie entwickelt, die Rechenzeit und Speicherverbrauch durch Optimierungstechniken, wie Restrukturierung oder Mehrfachverwendung von Verarbeitungsschritten verringern können. Daraus ergibt sich die Annahme, dass selbige Eigenschaften auch für FPM verwendet werden können. In dieser Arbeit stellen wir daher einen Ansatz vor, der FPM in ein DSMS integriert. Neben den Optimierungstechniken oder Wiederverwendbarkeit, können zusätzlich auch Funktionen wie Scheduling oder die Anfrageschnittstelle des DSMS ausgenutzt werden. Diese stellt eine höhere Ebene für die Anwendungsentwicklung bereit, welches wie bei Datenbankmanagementsystemen (DBMS) die Entwicklungszeit verringert (vgl. [EN06, GS02]). Des Weiteren speichert ein DSMS keine Rohdaten ab, wie es alternativ ein DBMS machen würde. Handelt es sich dabei um detaillierte personenbezogene Daten, wirkt sich dies positiv auf Datenschutz und Akzeptanz der Benutzer aus.

Die Verarbeitung von potentiell unendlichen Datenströmen bietet jedoch bestimmte Herausforderungen, die bei der Integration berücksichtigt werden müssen. Des Weiteren muss auch der zeitliche Zusammenhang der zu lernenden Aktivitäten beachtet werden. Obwohl es viele Lösungen gibt, die auf Datenströmen ausgelegt sind (vgl. [GHP+03, LL09, KRS11] u.a.), verlangen diese eine Anpassung des FPM, z.B. durch approximative Zusammenfassung von Daten. Ferner wurden einige Algorithmen zwar zur Evaluation in ein DSMS integriert, jedoch wurden bisher keine Optimierungsmöglichkeiten betrachtet, die durch die Integration in ein DSMS ermöglicht werden. Unser Beitrag ist daher wie folgt:

- Wir zeigen, wie FPM mit Hilfe des Intervall-Ansatzes [KS09] in ein DSMS integriert werden kann. Obwohl dies auf den ersten Blick eine Einschränkung bedeutet – nicht mehr alle Ereignisse – stellt es sich doch als sinnvoller dar, da typischerweise in diesem Szenario nur zeitlich korrelierende Ereignisse relevant sind. Der Ansatz lässt sich auch auf andere intervallbasierte Systeme übertragen.
- Wir definieren eine formale Semantik für FPM auf Basis einer relationalen Algebra für Datenströme. Die klare Semantik bietet eine deterministische und nachvollziehbare Berechnung von häufigen Mustern.
- Auf Basis der formalen Semantik evaluieren wir Optimierungsmöglichkeiten, indem ein FPM-Algorithmus als Algebraoperator in Zusammenhang mit existierenden Operatoren wie Selektion und Projektion betrachtet wird.

- Durch die Kapselung des konkreten Algorithmus zum FPM sind die vorgestellten Konzepte nicht auf einen bestimmten Algorithmus beschränkt. Es können bereits evaluierte und bewährte Verfahren auch aus Datenströmen eingesetzt werden.

Die restliche Arbeit gliedert sich wie folgt. Abschnitt 2 führt zunächst das verwendete Datenstrommodell anhand eines Beispiels ein und erläutert notwendige Definitionen und Annahmen. Darauf aufbauend zeigt Abschnitt 3, wie FPM in ein DSMS integriert werden kann und bietet dazu eine formale Semantik. Die Semantik wird in Abschnitt 4 verwendet, um algebraische Optimierungsmöglichkeiten aufzuzeigen. Integration als auch dessen Optimierung werden in Abschnitt 5 evaluiert. Während Abschnitt 6 verwandte Arbeiten betrachtet, fasst Abschnitt 7 die Arbeit abschließend zusammen und gibt einen Ausblick.

2 Motivation und Datenstrommodell

In einem Smart Home werden meist regelbasierte Systeme eingesetzt, die anhand erkannter Sensormesswerte bestimmte Aktorik ausführen. Ein Problem hierbei stellt die Definition der Regeln dar, die typischerweise von Benutzern vorgegeben werden müssen. Häufig können oder wollen sie sich jedoch nicht mit der Regelerstellung befassen oder wissen welche Regeln sinnvoll sind. Daher verwenden Hersteller meist eine Reihe von einfachen Standardregeln. Mit Hilfe von Lernverfahren können jedoch auch individuelle Regeln abgeleitet werden, indem regelmäßige Aktionen eines Benutzers erkannt werden. Abbildung 1 stellt mehrere solcher Aktionen dar. Um eine Regel abzuleiten, werden zunächst

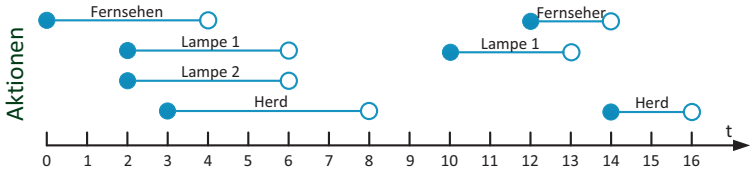


Abbildung 1: Beispiel der Gültigkeit von zusammen auftretenden Aktionen

häufig zusammen auftretende Aktionen betrachtet. Dazu wird der jeweilige Verwendungszeitraum als Gültigkeitsintervall gesehen. Beim *Fernseher* bspw. ist dies vom Einschaltzeitpunkt $t = 0$ bis Ausschaltzeitpunkt $t = 4$. Man sieht dann z.B., dass *Lampe 1*, *Lampe 2* und der *Herd* gleichzeitig mit dem *Fernseher* verwendet wurden. Damit liegt auch ein Muster *Fernseher, Lampe 1, Lampe 2, Herd* vor. Dieses tritt im Gegensatz zu dessen Teilmuster *Fernseher, Lampe 1* nur einmalig auf. Da das Teilmuster zweimal auftritt, ist es hier häufig und kann als Basis für eine Regel dienen, die bspw. *Fernseher* $\Rightarrow$ *Lampe 1* lautet, um automatisch *Lampe 1* einzuschalten, wenn die Aktion *Fernseher* auftritt.

Anhand des Beispiels aus Abbildung 1 kann bereits der intervallbasierte Charakter erkannt werden, auf dem das hier vorgestellte Prinzip beruht. Entsprechend bietet sich ein Datenstrommodell an, das auch auf Gültigkeitsintervalle basiert. Insbesondere eignet sich der Intervall-Ansatz [KS09], bei dem der Datenstrom durch Fenster in endliche Teile partitio-

niert wird. Zum einen definiert der Ansatz bereits Gültigkeitsintervalle pro Datentupel, so dass dies genau einer Aktion und dessen Verwendungszeitraum entspricht. Zum anderen bietet der Ansatz eine formale Semantik, die u.a. deterministische Ergebnisse und Optimierungen ermöglicht. Dazu wird der Datenstrom in mehrere Schnappschüsse unterteilt, wobei jeder Schnappschuss mit dem Zustand eines DBMS vergleichbar ist. Dadurch bietet der Ansatz äquivalente Operatoren der relationalen Algebra [DGK82] wie Selektion, Projektion oder Aggregation und auch dessen algebraische Optimierungsmöglichkeiten. Die Semantik dazu basiert auf der Definition eines logischen Datenstroms (vgl. [KS09]). Sei dazu $\mathbb{S}^l$ die Menge aller logischen Datenströme und $\mathbb{S}^l_{\mathcal{A}} \subseteq \mathbb{S}^l$ die Menge aller logischen Datenströme vom Schema $\mathcal{A}$. Dann ist ein *logischer Datenstrom* $S^l \in \mathbb{S}^l_{\mathcal{A}}$ wie folgt definiert: $S^l := \{(e,t,n) | e \in \Omega_{\mathcal{A}} \wedge t \in T \wedge n \in \mathbb{N} \wedge n > 0\}$ Dabei ist $\Omega_{\mathcal{A}}$ die Menge aller Nutzdatentupel vom Schema $\mathcal{A}$ und T die Menge aller Zeitstempel, sodass $\Omega_{\mathcal{A}} \times T \times \mathbb{N}$ ein Tupel im Datenstrom ist und ein Nutzdatentupel a mit einem Schema A zum Zeitpunkt t genau n mal auftritt. Abbildung 2 veranschaulicht diese Ansichtsweise für das zuvor genannte Beispiel. Betrachtet man zu jedem Zeitpunkt alle gültigen Aktionen, erhält man jeweils einen

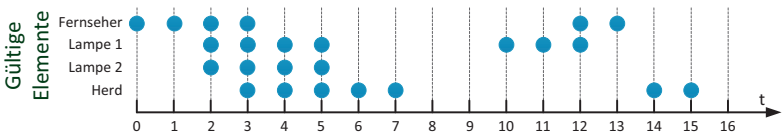


Abbildung 2: Beispiel der Gültigkeit von Tupeln eines logischen Datenstroms

Schnappschuss. So ergeben sich aus dem Beispiel in Abbildung 2 die in Tabelle 1 dargestellten Schnappschüsse. Obwohl diese logische Darstellung die erforderliche Semantik

T	Gültige Aktionen
0	{*Fernseher*}
1	{*Fernseher*}
2	{*Fernseher*, *Lampe* 1, *Lampe* 2}
3	{*Fernseher*, *Lampe* 1, *Lampe* 2, *Herd*}
4	{*Lampe* 1, *Lampe* 2, *Herd*}
5	{*Lampe* 1, *Lampe* 2, *Herd*}
6	{*Herd*}
7	{*Herd*}

T	Gültige Aktionen
8	{}
9	{}
10	{*Lampe* 1}
11	{*Lampe* 1}
12	{*Fernseher*, *Lampe* 1}
13	{*Fernseher*}
14	{*Herd*}
15	{*Herd*}

Tabelle 1: Gültige Schnappschüsse pro Zeitpunkt für den logischen Datenstrom

und dadurch auch eine Optimierung ermöglicht, entspricht diese Repräsentation nicht dem in Abbildung 1 gezeigten Intervallen und müsste demnach umgewandelt werden. Jedoch wäre eine Umwandlung in einen logischen Datenstrom aufwändig und zum anderen ist die Repräsentation einer Aktion ineffizient, wenn diese über mehrere zusammenhängende Zeitpunkte gültig ist, da es bspw. für *Lampe 2* vier statt nur eine Instanz geben müsste. Daher wird die logische Darstellung im Wesentlichen für die Semantik und darauf aufbauend für die Optimierung verwendet. Die eigentliche Verarbeitung erfolgt, wie auch im Intervall-Ansatz, durch eine physische Repräsentation. Diese fasst mehrere zusammenhängende Gültigkeitszeitpunkte zu einem Gültigkeitsintervall zusammen. Dieses Intervall

beschreibt jeweils, dass ein Tupel von einem Startzeitstempel $t_s \in T$ bis zu einem Endzeitstempel $t_e \in T$ gültig ist. Sei $\mathbb{S}^p$ die Menge aller physischen Datenströme, dann ist $\mathbb{S}^p_{\mathcal{A}} \subseteq \mathbb{S}^p$ die Menge aller physischen Datenströme mit dem Schema $\mathcal{A}$ und ein *physischer Datenstrom* $S^p \in \mathbb{S}^p_{\mathcal{A}}$ (vgl. [KS09]) definiert durch:$S^p := \{(e, [t_s, t_e)|a \in \Omega_{\mathcal{A}} \wedge t_s, t_e \in T\}$. Im Gegensatz zum logischen Datenstrom wird hier die Reihenfolge aller Tupel e betrachtet, indem sie monoton aufsteigend anhand der Startzeitstempel t_s sortiert sind und zwei Aktionen sind gleichzeitig gültig, wenn sich ihre Zeitintervalle überschneiden.

3 Intervallbasiertes Frequent Pattern Mining

Für die Integration von FPM wir ein neuer Operator eingeführt, der analog zum Intervall-Ansatz sowohl auf logischen als auch auf physischen Datenströmen definiert wird. Dazu wird zunächst die Semantik des Operators als eine Abbildung zwischen logischen Datenströmen beschrieben. Existierende Verfahren zum FPM werden meist zur Warenkorbanalyse eingesetzt, indem bspw. zusammen gekaufte Produkte als eine sogenannte Transaktion betrachtet werden, zwischen denen dann Gemeinsamkeiten gesucht werden. Die Idee hinter FPM auf intervallbasierten Aktionen ist es, gleichzeitig gültige Aktionen genau als eine solche Transaktion aufzufassen. Dadurch können vorhandene Algorithmen zur Warenkorbanalyse wie bspw. Apriori [AS+94] oder FP-Growth [HPY00] wiederverwendet werden und Algorithmen müssen nicht für Datenströme angepasst werden. Diese Algorithmen erwarten jedoch eine endliche Anzahl an Transaktionen, obwohl der Datenstrom unendlich viele Transaktionen liefern kann. Analog zu einem gleitenden Fenster (vgl. [KS09]), wie er im Intervall-Ansatz zum Bestimmten der Gültigkeit einer Aktion verwendet wird, wird hier ein Fenster bestimmt, in dem zusammenhängend gelernt wird. Anhand eines vorgegebenen Zeitraums oder einer Anzahl, werden veraltete Transaktionen aus dem Fenster entfernt, sobald neue Transaktionen hinzukommen. Dieses Fenster erlaubt es außerdem, dass sich das Lernen anpassen kann, indem es veraltete Muster verlernt. Dadurch kann mit Concept Drifts (vgl. [GZK05]) umgegangen werden, die bspw. auftreten, wenn der Bewohner sein Verhalten ändert oder Geräte und damit Aktionen nicht mehr existieren. In Tabelle 2 links ist beispielhaft ein Ausschnitt von $w = 5$ Transaktionen zum Zeitpunkt $t = 6$ dargestellt. Hat man einen Zeitfortschritt zu $t = 7$, dann verschiebt sich

Transaktionen (t=6)
{Fernseher, Lampe 1, Lampe 2}
{Fernseher, Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Herd}

Transaktionen (t=7)
{Fernseher, Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Herd}
{Herd}

Tabelle 2: Beispiel für zwei Ausschnitte von Transaktionsmengen

der Ausschnitt um eine Transaktion, wie Tabelle 2 rechts zeigt. Da es sich um eine endliche Menge von Transaktionen handelt, können existierende Algorithmen zum FPM auf statischen Daten übernommen werden. Der hier vorgestellte Ansatz ist dadurch nicht auf

einen bestimmten Algorithmus zugeschnitten. Jedoch muss die Semantik der Algorithmen äquivalent sein, indem sie dasselbe Ergebnis liefern würden. Dabei ist eine Kombination von Aktionen ein häufiges Muster, wenn es einen Support s erfüllt, indem die Kombination mindestens s mal im Ausschnitt vorkommt. Des Weiteren muss auch jede Teilmenge des häufigen Musters selbst ein häufiges Muster sein [HPY00]. Ist X die Menge aller möglichen Aktionen, so ergibt die Potenzmenge $\wp(X)$ die Menge aller möglichen Muster (bis auf die leere Menge), die ein FPM als mögliche Kandidaten betrachtet. Von dieser Kandidatenmenge sind dann nur solche häufig, die mindestens s mal in der betrachteten Transaktionsmenge vorkommen. Hierbei ist anzumerken, dass verschiedene Algorithmen dieselben Ergebnisse liefern, aber gerade das Erzeugen der Kandidatenmenge, das Zählen des Support und das Prüfen gegenüber dem Datenbestand den unterschied der Algorithmen ausmacht. Für den Ausschnitt $t = 6$ aus Tabelle 2 ergibt sich die in Tabelle 3 gezeigte Kandidatenmenge. Betrachtet man noch den Support s, so dass bspw. ein minimaler Sup-

mögliches Muster	Support s
{Fernseher}	2
{Herd}	4
{Lampe 1}	4
{Lampe 2}	4
{Fernseher, Herd}	1
{Fernseher, Lampe 1}	2
{Fernseher, Lampe 2}	2
{Herd, Lampe 1}	3

mögliches Muster	Support s
{Herd, Lampe 2}	3
{Lampe 1, Lampe 2}	4
{Fernseher, Herd, Lampe 1}	1
{Fernseher, Herd, Lampe 2}	1
{Fernseher, Lampe 1, Lampe 2}	2
{Herd, Lampe 1, Lampe 2}	3
{Fernseher, Herd, Lampe 1, Lampe 2}	1

Tabelle 3: Kandidaten für $t = 6$

port von $s = 2$ notwendig ist, ergeben sich die in Tabelle 4 gezeigten häufigen Muster. Analog dazu werden auch zum Zeitpunkt $t = 7$ entsprechende Kandidaten berechnet. Um

häufiges Muster ($t = 6$)	s
{Fernseher}	2
{Herd}	4
{Lampe 1}	4
{Lampe 2}	4
{Fernseher, Lampe 1}	2
{Fernseher, Lampe 2}	2
{Herd, Lampe 1}	3
{Herd, Lampe 2}	3
{Lampe 1, Lampe 2}	4
{Fernseher, Lampe 1, Lampe 2}	2
{Herd, Lampe 1, Lampe 2}	3

häufiges Muster ($t = 7$)	s
{Herd}	5
{Lampe 1}	3
{Lampe 2}	3
{Herd, Lampe 1}	3
{Herd, Lampe 2}	3
{Lampe 1, Lampe 2}	3
{Herd, Lampe 1, Lampe 2}	3

Tabelle 4: Häufige Muster für minimalen Support von $s = 2$

die Semantik eines solchen Verfahrens zum FPM zu beschreiben, werden die beiden zuvor genannten Schritte formalisiert. Dazu wird auf Grundlage logischer Datenströme eine Abbildung definiert, die den betrachteten Ausschnitt an Transaktionen bestimmt. Darauf

aufbauend wird anschließend eine weitere Abbildung definiert, welche die Erzeugung von häufigen Mustern nach obiger Semantik formalisiert.

Sei dazu $w \in \mathbb{N}$ mit $w > 0$ ein Parameter und $S^l \in \mathbb{S}^l$ ein logischer Datenstrom mit entsprechenden Tupeln $(e, \hat{t}, n) \in S^l$. Dann definieren wir eine Abbildung $\tau_t^w : \mathbb{S}^l \to \mathbb{S}^l$, die zu einem Zeitpunkt t bis zu w Transaktionen aus S^l entnimmt:

$$\tau_t^w(S^l) := \{(e, \hat{t}, n) \in S^l | max(0, t - w + 1) \leq \hat{t} \leq t\} \tag{1}$$

Wie erwähnt, ist die so erzeugte Transaktionsmenge im Gegensatz zu S^l endlich, so dass im Anschluss daran ein entsprechendes FPM ausgeführt werden kann. Obwohl hierzu ein konkreter Algorithmus, wie Apriori oder FP-Growth verwendet wird, abstrahieren wir das konkrete Verfahren. Im Wesentlichen werden alle vorhandenen Ausprägungen, die ein Tupel (e, t, n) für e annehmen kann, betrachtet. Aus dieser Menge wird dann eine Potenzmenge erzeugt, die der Kandidatenmenge entspricht. Die daraus entnommenen häufigen Muster ist letztendlich eine Teilmenge der Potenz- bzw. Kandidatenmenge. Dies bedeutet, dass FPM als eine Abbildung $\phi_t : \mathbb{S}^l \to \mathbb{S}^l$ gesehen werden kann, die zu einem Zeitpunkt $t \in T$ eine Menge von häufigen Mustern berechnet und wie folgt definiert werden kann:

$$\phi_t(S^l) := \{(m, t, 1) | m \subseteq \wp(X) \wedge X := \{e | (e, t, n) \in S^l\} \tag{2}$$

Hierbei ist zu erkennen, dass m hier im Vergleich zu τ_t^w selbst eine Menge von Tupeln ist. Der Unterschied liegt darin, dass das Nutzdatentupel e ein festes Schema hat, wie z.B. der Name der Aktion. Das Problem bei häufigen Mustern ist jedoch, dass es kein festes Schema gibt, da die Muster unterschiedliche Längen haben. Um dennoch eine Verarbeitung mit festem Schema zu erlauben, werden die häufigen Muster zu einem Tupel verschachtelt. Gibt es demnach mehrere Nutzdatentupel e_i, die jeweils ein eigenes Schema $\mathcal{A}_i$ haben, sodass $e_i \in \Omega_{\mathcal{A}_i}$ mit $1 \leq i \leq k$, dann können diese zu einer Menge $\{e_1, ..., e_k\}$ mit Schema $\mathcal{A}$ zusammengefasst werden. Dabei ist $k \in \mathbb{N}$ die Länge eines häufigen Musters und kann somit variieren. Für den Spezialfall $k = 1$ kann vereinfacht $\Omega_{\mathcal{A}} = \{\Omega_{\mathcal{A}_1}\}$ angenommen werden. Ein Tupel hat daher statt der Form $\Omega_{\mathcal{A}} \times T \times \mathbb{N}$ die Form $\{\Omega_{\mathcal{A}}\} \times T \times \mathbb{N}$, sodass man ein Datenmodell erhält, welches der Non-First-Normal-Form (NF^2) [SP82] ähnelt. Obwohl auch hier die Tupel wie bei der NF^2 verschachtelt werden (vgl. *nest*-Operation), gelten hier jedoch alle algebraischen Operationen auf das gesamte Tupel und nicht nur jeweils auf die verschachtelten Teile. Eine Selektion bspw. würde bei der NF^2 nur verschachtelte Teile entfernen und das hier vorgestellte Modell hingegen das gesamte Tupel verwerfen, wenn nur ein Teil nicht dem Prädikat genügt. Die Unterscheidung ist zum einen dadurch motiviert, dass ein häufiges Muster nur als Gesamtes seine Aussage behält und zum anderen ist jede Teilmenge eines häufigen Musters selbst ein häufiges Muster, sodass ein Entfernen eines Teils zu einem vorhandenen Muster führen würde und damit redundant vorhanden wäre. Anzumerken ist, dass ϕ_t auch für potenziell unendliche Datenströme definiert ist, da t lediglich den Zeitstempel setzt, aber keinen Ausschnitt wählt. Ferner kann hier auch ein FPM-Algorithmus eingesetzt werden, der bspw. inkrementell berechnet werden kann und für Datenströme ausgelegt ist. Für existierende Algorithmen, die auf endliche Datensätze ausgelegt sind, sollte ϕ_t jedoch stets in Verbindung mit τ_t^w verwendet werden. Für ein intervallbasiertes FPM ergibt sich daher der eigentliche logische FPM-Operator ρ_t^w für einen logischen Datenstrom S^l durch die Komposition $\phi_t \circ \tau_t^w$ wie

folgt $\rho_t^w(S^l) := \phi_t(\tau_t^w(S^l))$. Hierbei sei auf den Zusammenhang von s und w hingewiesen, indem ein Wert maximal w-mal auftreten kann und maximal s mal auftreten muss. Folglich muss $s \leq w$ gelten, damit ein FPM hier noch Ergebnisse liefern kann.

Da die logische Repräsentation wie erwähnt ineffizient ist, wird die Implementierung anhand physischer Datenströme vorgenommen. Zusätzlich kann dadurch derselbe logische Operator während der Transformation durch verschiedene physische Implementierungen mit konkreten Algorithmen wie Apriori oder FP-Growth ausgetauscht werden. Um hier jedoch von einem konkreten Algorithmus zu abstrahieren, stellen wir einen allgemeinen physischen Operator vor. Die physische Repräsentation von ρ_t^w kann dabei durch den in Algorithmus 1 gezeigten Pseudocode umgesetzt werden.

Algorithmus 1 FPM-Operator

Require: Physischer Datenstrom S_{in}
Ensure: Physischer Datenstrom S_{out}
1: $min_{t_s} \in T \cup \{\bot\}$;$min_{t_s} \leftarrow \bot$
2: Sei *puffer* eine Prioritätsqueue für Elemente $(e, [t_s, t_e))$ mit Ordnungsrelation $\leq_{t_s}$
3: Sei *transactions* eine FIFO-Liste für Mengen der Form $\{(e, [t_s, t_e))\}$
4: Sei w die Anzahl der Transaktionen
5: Sei *fpm* ein Algorithmus zum FPM
6: **for** $s := (e, [t_s, t_e)) \hookleftarrow S_{in_j}$ **do**
7: **if** $t_s > min_{t_s}$ **then**
8: $ta \leftarrow \emptyset$
9: **while not** *puffer.isEmpty()* **do**
10: $\hat{s} := (\hat{e}, [\hat{t_s}, \hat{t_e})) \leftarrow$ *puffer.peek()*
11: **if** $\hat{t_s} \leq t_s$ **then**
12: $\hat{s} \leftarrow$ *puffer.poll()*
13: *ta.insert*$(\hat{s})$
14: **else**
15: **break**
16: **end if**
17: **end while**
18: **if** *transactions.size()* == w **then**
19: *transactions.poll()*
20: **end if**
21: *transactions.offer(ta)*
22: *frequentsets = fpm(transactions)*
23: **for all** *frequentset* $\hookleftarrow$ *frequentsets* **do**
24: $\widetilde{s} :=$ (*frequentset*, $[min_{t_s}, t_s)$)
25: $\widetilde{s} \hookrightarrow S_{out}$
26: **end for**
27: **end if**
28: $min_{t_s} = t_s$
29: *puffer.offer(s)*
30: **end for**

Der Algorithmus speichert alle eingehenden Tupel in einen *puffer*. Wenn bei einem Zeitfortschritt $t_s > min_{t_s}$ gilt, sind alle nötigen Tupel vorhanden, aus denen dann eine Trans-

aktion *ta* gebildet wird, die zu den vorhandenen Transaktionen *transactions* hinzugefügt wird. Gibt es w Transaktionen, dann wird die älteste Transaktion entfernt und setzt damit Abbildung τ_t^w um. Abbildung ϕ_t hingegen wird einerseits durch den austauschbaren Algorithmus *fpm* für das eigentliche FPM und andererseits durch setzen des Zeitintervalls $[min_{t_s}, t_s)$ umgesetzt. Hierbei sollte *fpm* eine Menge aus mehreren *frequentset* liefern, die wiederum Kombinationen aus den Eingangstupeln *e* sind. Jedes *frequentset* wird dem ausgehenden Datenstrom S_{out} übergeben. Der Algorithmus ist datengetrieben und wird durch *s* angestoßen. Jedoch wird eine Transaktion erst erzeugt, wenn zum Zeitpunkt t_s alle Tupel für den Ausschnitt $[min_{t_s}, t_s)$ bekannt sind. Entsprechend muss auf ein Tupel mit $t_s + 1$ gewartet werden, sodass blockiert wird. Kommen bspw. keine Tupel mehr, können Heartbeats bzw. Punctuations [TMSF03] jedoch eine vorzeitige Berechnung auslösen, indem sie den Zeitfortschritt angeben. Dazu ersetzt der Zeitstempel des Heartbeats t_h die entsprechenden t_s in Zeile 7-27.

4 Algebraische Optimierungen

Durch die semantische Beschreibung eines FPM auf Basis der relationalen Algebra entsteht die Möglichkeit für Optimierungen. Analog zu einem DBMS verfügt ein DSMS häufig auch über einen Anfrageoptimierer, der eine Anfrage in Form einen gerichteten Graphen aus Algebraoperatoren (dem Anfrageplan) entgegennimmt und auf Grundlage von Äquivalenz-Regeln, die zwischen verschiedenen Operatorkombinationen gelten, optimiert. So können bspw. zwei Operatoren getauscht oder unnötige Operatoren entfernt werden, ohne dass sich das Ergebnis dadurch ändert. Die gebräuchlichsten Optimierungen basieren dabei auf dem Tauschen mit Selektionen und Projektionen, da diese die Anzahl der Tupel bzw. Attribute verringert und im Vergleich zu anderen Operatoren wie Verbünde weniger rechenintensiv sind. Daher wird versucht, Selektionen möglichst nah zu den Quellen zu verschieben und mit Projektionen alle unnötigen Attribute zu entfernen. Im Folgenden betrachten wir daher Selektionen und Projektionen in Bezug auf FPM. Um zu zeigen, dass eine Selektion bzw. Projektion in Kombination mit einem FPM-Operator verlustfrei optimiert werden kann, muss die Äquivalenz gezeigt werden. Für die intervallbasierten Standardoperatoren zeigt [KS09] entsprechende Äquivalenzen, bei dem zwei Kombinationen aus Operatoren äquivalent sind, wenn dessen resultierenden Datenströme zu jedem Zeitpunkt dieselben Tupel enthalten. Im Folgenden untersuchen wir damit die Kombination von FPM und Selektion, sowie FPM und Projektion.

4.1 Optimierung mit Selektionen

Möchte ein Bewohner bspw. nicht, dass der *Fernseher* automatisch geschaltet wird, würde man entsprechende Muster mit *Fernseher* nach dem FPM entfernen. Es bietet sich an, dass eine solche Selektion nicht nach dem kostenintensiven FPM-Operator durchgeführt wird, sondern wenn möglich bereits vorher, damit durch die Selektion weniger Elemente verarbeitet werden müssen. Hierbei muss jedoch zum einen beachtet werden, dass der

verwendete FPM-Algorithmus Häufigkeiten nur absolut und nicht relativ zählen darf, da eine Selektion die Grundmenge verändert und zum anderen liegen vor und nach dem FPM verschiedene Schema bzw. Modelle vor, wie in Abschnitt 3 beschrieben wird. Vor einem FPM gilt das flache Modell, bei dem die Selektion für ein Prädikat p nach [KS09] durch $\sigma_p(S) := \{(e,t,n) \in S | p(e)\}$ definiert ist. Nach einem FPM liegen jedoch verschachtelte Tupel vor, auf denen das Prädikat für alle Teiltupel gelten muss. Eine verschachtelte Selektion ist daher wie folgt definiert: $\hat{\sigma}_p(S) := \{(\hat{e},\hat{t},\hat{n}) \in S | \forall \hat{a} \in \hat{e} : p(\hat{a})\}$. Der Unterschied zum NF^2 Modell, bei dem nur die Teiltupel und nicht das gesamte Tupel entfernt wird, ist wie folgt motiviert. Sei bspw. *{Fernseher, dunkel, Lampe}* ein häufiges Muster, aus dem eine Regel erzeugt wird, welche die *Lampe* anschaltet sobald es *dunkel* ist und der *Fernseher* angemacht wird. Möchte der Bewohner keine Regeln mit *Fernseher*, so kann das gesamte zugehörige häufige Muster *{Fernseher, dunkel, Lampe}* entfernt werden, da auf Grund der Eigenschaften eines häufigen Musters, das Teilmuster *{dunkel, Lampe}* ohnehin vorhanden ist und sonst Redundanz vorliegen würde. Liegt ein Datenstrom S vor, auf dem ein FPM und anschließend eine Selektion ausgeführt wird, gilt $\hat{\sigma}_p(\rho_t^w(S))$. Bei einer Optimierung würde die Selektion vor dem FPM ausgeführt werden, sodass $\rho_t^w(\sigma_p(S))$. Daraus folgt folgende Optimierungsregel[1]:$\hat{\sigma}_p(\rho_t^w(S)) = \rho_t^w(\sigma_p(S))$.

4.2 Optimierung mit Projektion

Bei der Optimierung mit einer Projektion π wird die Verwendung von Attributen betrachtet. Sowohl der Teil τ_t^w als auch der Teil ϕ_t der Abbildung ρ_t^w verwenden zwar nicht explizit Attribute, jedoch basiert das Zählen der Häufigkeiten im FPM-Algorithmus auf der Äquivalenz von Tupeln, die sich nach den verwendeten Attributen richtet. Zwei verschiedene Tupel *(Fernseher, an)* und *(Fernseher, aus)* würden so zu äquivalenten Tupeln *(Fernseher)*. Obwohl es durchaus möglich ist, nur einen Teil der Attribute (im Beispiel nur das erste Attribut) im FPM zu verwenden, bietet sich dieses nicht an. Zum einen würde das FPM dadurch selbst eine Projektion durchführen und die Funktion wäre dann redundant und nicht mehr gekapselt. Zum anderen wäre die Frage, wie nicht verwendete Attribute für das häufige Muster zusammengeführt werden. Im Beispiel würde man zwar zweimal *Fernseher* zählen, jedoch wäre die Frage ob *an* oder *aus* verwendet werden soll. Ist $\hat{\pi}$ analog zur Selektion eine Projektion auf verschachtelten Tupeln und π das entsprechende Gegenstück für flache Tupel, dann folgt die Regel $\hat{\pi}_p(\rho_t^w(S)) \neq \rho_t^w(\pi(S)$. Somit ist eine Verschiebung einer Projektion bei Verwendung eines FPM-Operators nicht möglich[2].

5 Evaluation

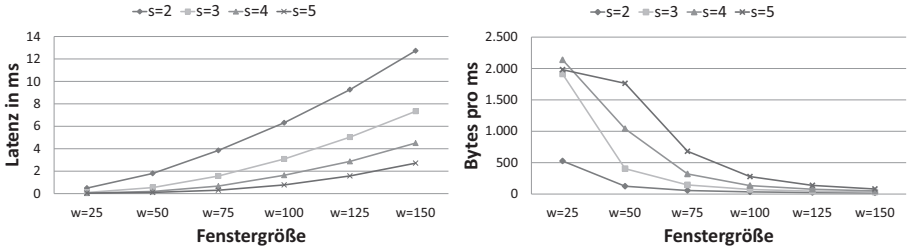
Für die Evaluation wurde der physische FPM-Operator in das DSMS Odysseus [AGG+12] integriert. Als Testdatensatz dient *T10I4D100K* [AS+94], der sehr häufig zum Vergleich

[1] Ein formaler Beweis ist vorhanden, kann aber aus Platzgründen nicht dargestellt werden
[2] Ein formaler Beweis wurde aus Platzgründen entfernt

von FPM-Algorithmen verwendet wird. Die insgesamt $D = 100.000$ Transaktionen haben durchschnittlich $T = 10$ Werte und haben häufige Muster mit einer durchschnittlichen Länge von $I = 4$. Die insgesamt 1.010.229 Werte werden nacheinander durch Odysseus eingelesen. Dabei haben die Werte einer Transaktion dasselbe Gültigkeitsintervall erhalten, sodass bspw. die Werte der ersten Transaktion $[0, 50)$ und die der zweiten Transaktion $[100, 150)$ als Gültigkeit erhalten haben. Für die Evaluation wurde der FP-Growth [HPY00] implementiert, kann prinzipiell jedoch gegen andere Verfahren ausgetauscht werden. Der verwendete Rechner verfügt über eine Intel Core i5 CPU mit zwei Kernen mit je 2.50 GHz und 16 GB Arbeitsspeicher unter einem 64-Bit Windows 7. Es wurden Durchsatz und Latenz gemessen, da diese die maßgeblichen Metriken in der Datenstromverarbeitung sind. Bei der Latenz, die Dauer zwischen Ein- und Austritt eines Wertes angibt, konnte der Wert nicht direkt gemessen werden, weil ein erzeugtes häufiges Muster nur indirekt von eingehenden Werten abhängt. Analog zur Latenzberechnung bei einer Aggregation wurde die Latenz von dem Wert gemessen, der als letztes zur Berechnung des häufigen Musters beigetragen hat. Die Evaluation selbst behandelt in Abschnitt 5.1 zunächst die Integration des FPM und anschließend in Abschnitt 5.2 die Optimierung.

5.1 Integration von FPM

Bei der Integration wurden die Machbarkeit und die Skalierbarkeit für einen potenziell unendlichen Datenstrom evaluiert. Dazu zeigen die Latenzen in Abbildung 3a, dass einerseits ein kleinerer Support s und andererseits ein größeres Fenster w zu höheren Latenzen führt. In beiden Fällen müssen jeweils mehr Daten vorgehalten werden, um potenziel-



(a) Latenz (b) Durchsatz

Abbildung 3: Evaluation bei verschiedenen Fenstergrößen und Support

le häufige Muster zu erkennen. Da bei n möglichen Ausprägungen bis zu 2^n potenzielle Muster (vgl. Potenzmenge in Definition 2) entstehen können, wächst auch hier der Aufwand nicht linear mit zunehmender Größe des Baums. Ein selbiges Verhalten spiegelt sich auch im Durchsatz wider, der in Abbildung 3b gezeigt wird. Auch hier sinkt der Durchsatz bei kleinerem Support s bzw. größerem Fenster w. Hierbei ist zu erwähnen, dass der maximale Durchsatz systembedingt bei etwa 2.200 Bytes pro ms lag. Betrachtet man die Latenz über die Zeit, wie sie in Abbildung 4 dargestellt ist, erkennt man für verschiedene Support s, dass die Latenz nicht ansteigt, sondern um einen konstanten Wert berechnungs-

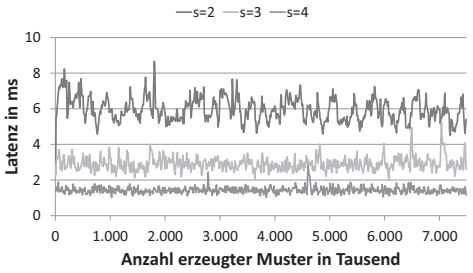
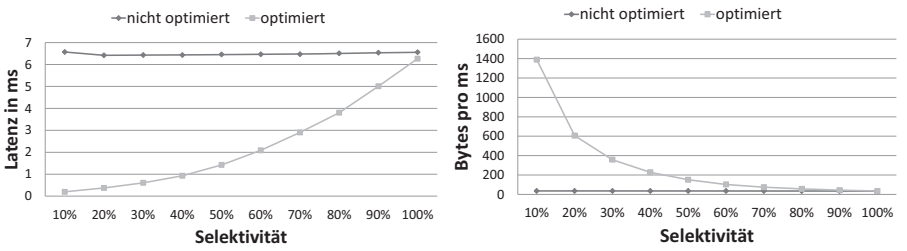


Abbildung 4: Durchschnittliche Latenz über die Zeit für $w = 100$

bedingt schwankt. Hieraus lässt sich erkennen, dass die Integration dieses Verfahrens auch für potenziell unendliche Datenströme skaliert.

5.2 Optimierung von FPM

Um den Mehrwert der Optimierung zu zeigen, wird dieser mit dem unoptimierten Fall verglichen. Maßgeblich bei der Selektion ist die Selektivität des Prädikats. Da die Werte in *T10I4D100K* gleich verteilt zwischen 0 und 999 liegen (vgl. [AS+94]), wurde bspw. bei eine Selektivität von 10% das Prädikat „< 100“ verwendet, um die Selektivität zu simulieren. Abbildung 5a zeigt die Latenzen, die dem erwarteten Verhalten entsprechen. Die



(a) Latenz (b) Durchsatz

Abbildung 5: Vergleich bei Optimierung mit Selektion

Latenzen sind im nicht optimierten Fall konstant, da sowohl FPM als auch die nachfolgende Selektion unabhängig von der Selektivität für jedes Element durchgeführt werden muss. Der optimierte Fall hingegen ist stark von der Selektivität abhängig, da am Anfang (je nach Selektivität) bereits viele Daten verworfen werden können und dadurch die zwischengespeicherten Daten bzw. Kandidaten für das FPM kleiner gehalten werden. Dies wird verdeutlicht, indem bei einer Selektivität von 100% fast keine Optimierung mehr zu erkennen ist. Der dennoch vorhandene Unterschied bei 100% liegt daran, dass die Selektion $\hat{\sigma}$ im nicht optimierten Fall bei Werten von 0 bis 999 je nach Support bis zu 2^{1000} Muster prüfen muss. Im optimierten Fall sind es hingegen alle 1.010.229 Werte des Da-

tensatzes. Analog zur Latenz spiegelt auch der Durchsatz das Ergebnis wider, indem eine kleinere Selektivität im optimierten Fall zu einem höheren Durchsatz führt und sich dieser bei 100% dem unoptimierten Fall angleicht.

6 Verwandte Arbeiten

Die Berechnung häufiger Muster bzw. Frequent Pattern Mining (FPM) wird auch als Warenkorb- oder Assoziationsanalyse bezeichnet und ist ein Ansatz des Data Mining (vgl. [HKP11, WFH11] u.a.) und werden dabei i.d.R. zur explorativen Analyse von statischen Daten benutzt. Apriori [AS+94] ist einer der ersten Algorithmen zum FPM, auf dessen Basis anschließend Weitere entwickelt wurden. Da die Kandidatengenerierung von Apriori gerade bei kleinem Support exponentiell wächst, wurde FP-Growth [HPY00] entwickelt, der einen Präfix-Baum als Alternative einsetzt, auf denen dann wiederum neuere Verfahren basieren. Da klassische Data-Mining-Algorithmen für Datenbanken ausgelegt sind, gibt es weitere Anpassungen, die Besonderheiten für Datenströme berücksichtigen. Als Pendant für Datenströme hat sich das Data Stream Mining (vgl. [Gam07, Gam10, GZK05] u.a.) entwickelt, in denen auch FPM auf Datenströmen [JG06] betrachtet wird. Einige Ansätze wie [LL09, TMP08] beschäftigen sich dabei mit dem Zählen von Häufigkeiten (vgl.. [CH08]), da dies ein grundlegendes Problem potentiell unendlicher Datenströme ist. Als Alternative gibt es Fensteransätze, die den Datenstrom in endliche Teile partitionieren. FP-Stream [GHP+03] bspw. versucht eine Historie von Fenstern aufzubauen, um so auch eine (nachträgliche) explorative Analyse auf Datenströmen zu ermöglichen. Weitere Algorithmen, wie [CWYM04, LL09, KRS11] u.a., fokussieren jeweils spezielle Probleme bei Datenströmen, beschränken sich auf den eigentlichen Algorithmus. So haben sie zwar verschiedene Ansätze oder betrachten andere Arten von häufigen Mustern, jedoch wird nicht auf die Integration in ein bestehendes DSMS eingegangen. Unser Ansatz hingegen zeigt eine solche Integration, bei dem zusätzlich keine speziellen Anpassungen an Datenströme nötig sind und Algorithmen wie FP-Growth wiederverwendet werden können.

Des Weiteren bietet keine der vorhandenen Algorithmen eine formale Semantik, die auf einer vorhandenen Algebra beruht. DSMS wie Aurora [A+03], Borealis [A+05], STREAM [ABB+04], Odysseus [AGG+12] oder PIPES [KS09] sind i.A. auf einer Algebra aufgebaut. Dieser Ansatz beruht dabei auf der Intervall-Algebra nach [KS09], die u.a. in PIPES und Odysseus eingesetzt wird, da diese sich sehr gut für die Repräsentation von Gültigkeitsintervallen der Aktionen eignet. Einige Systeme, insbesondere Stream Mill Miner [T+11], integrieren zwar Data Mining und Datenstrommanagement, berücksichtigen dabei jedoch nur Klassifikation und Clustering. Des Weiteren beruhen die Umsetzungen auf einer benutzerdefinierten Aggregation und verfügen damit nicht über eine formale Semantik. Aus diesem Grund werden auch keine Optimierungen betrachtet. Dieser Ansatz betrachtet sowohl FPM im Kontext von Datenstrommanagement, beschreibt diesen formal und betrachtet gleichzeitig auch Optimierungen von Data Mining und vorhandenen Datenstrom-Operatoren. Neben der Optimierung der Verarbeitung, spiegelt sich ein integrierter Ansatz auch in der Anwendungsentwicklung wider, wie es [GS02] analog für die Integration von Data Mining in ein DBMS zeigt.

7 Zusammenfassung

In diesem Beitrag haben wir einen neuen Operator zum Frequent Pattern Mining (FPM) auf Datenströmen eingeführt. Dazu haben wir auf Basis der existierenden relationalen Algebra eine klare Semantik formuliert und eine mögliche Implementierung gezeigt. Durch die Verwendung des Intervall-Ansatzes ist es möglich, beliebige FPM-Algorithmen für statische Datenquellen in einem DSMS einzusetzen, ohne dass die Algorithmen für die Verarbeitung von Datenströmen angepasst werden müssen. Des Weiteren haben wir ein neues Konzept eingeführt, indem der FPM-Operator in Zusammenhang mit Projektion und Selektion betrachtet wird, um daraus Optimierungen zu ermöglichen. Entsprechend haben wir gezeigt, dass ein Vertauschen mit einer Projektion nicht möglich ist und dass das Vertauschen mit einer Selektion Latenzen senken und den Durchsatz steigern kann. Sowohl die Performanz der eigentlichen Integration als auch die Optimierung bei einer vorhandenen Selektion haben wir anschließend für verschiedene Parameterausprägungen evaluiert. Die Ergebnisse der Evaluation spiegelten dabei die zu erwarteten Effekte wider.

Obwohl Selektion und Projektion durch das Entfernen von Attributen und Tupeln das größte Optimierungspotenzial bietet, untersuchen wir ergänzend weitere Operatoren wie Kreuzprodukt oder Differenz, um damit die minimale Menge der Operatoren der relationalen Algebra abzudecken. Das Wiederverwenden eines FPM-Operators ist ein weiterer Punkt, der ebenfalls Optimierungspotenzial in DSMS bietet. Hierbei muss unter anderen die semantische Äquivalenz zweier FPM-Operatoren betrachtet werden, die eventuell auch durch ein Teilen des einen FPM-Operators oder durch zusätzliche Operatoren, wie z.B. eine Selektion, erreicht werden kann. Des Weiteren evaluieren wir die Umsetzbarkeit des Konzepts anhand einer prototypischen Fallstudie, indem Odysseus und der FPM-Operator als Teil einer Demonstration mit verschiedenen Sensoren verwendet werden.

Literatur

[A+03] D. J. Abadi et al. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2), 2003.

[A+05] D. Abadi et al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

[ABB+04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, R. Motwani, R. Motwani und U. Srivastava. STREAM: The Stanford Data Stream Management System, 2004.

[AGG+12] H.-Jürgen Appelrath, Dennis Geesen, Marco Grawunder, Timo Michelsen und Daniela Nicklas. Odysseus: a highly customizable framework for creating efficient event stream management systems. In *DEBS '12*. ACM, 2012.

[AS+94] R. Agrawal, R. Srikant et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, Jgg. 1215, 1994.

[CH08] Graham Cormode und Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2), 2008.

[CWYM04] Y. Chi, H. Wang, P.S. Yu und R.R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *ICDM'04*. IEEE, 2004.

[DAS01] A.K. Dey, G.D. Abowd und D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI*, 16(2-4), 2001.

[DGK82] Umeshwar Dayal, Nathan Goodman und R.H. Katz. An extended relational algebra with control over duplicate elimination. In *ACM PODS*. ACM, 1982.

[EN06] Ramez Elmasri und Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 5th edition. Auflage, 2006.

[Gam07] Joao Gama. *Learning from Data Streams: Processing Techniques in Sensor Networks*. Springer Berlin Heidelberg, 2007.

[Gam10] Joao Gama. *Knowledge discovery from data streams*. Taylor and Francis, 2010.

[GHP+03] C. Giannella, J. Han, J. Pei, X. Yan und P. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. *Next generation data mining*, 2003.

[GS02] Ingolf Geist und Kai-uwe Sattler. Towards data mining operators in database systems: Algebra and implementation. In *DBFusion*. Citeseer, 2002.

[GZK05] Mohamed Medhat Gaber, Arkady Zaslavsky und Shonali Krishnaswamy. Mining Data Streams : A Review. *ACM SIGMOD Record*, 34(2), 2005.

[HCXY07] J. Han, H. Cheng, D. Xin und X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1), 2007.

[HKP11] Jiawei Han, Micheline Kamber und Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3. Auflage, 2011.

[HPY00] J. Han, J. Pei und Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, Jgg. 29. ACM, 2000.

[JG06] Nan Jiang und Le Gruenwald. Research issues in data stream association rule mining. *SIGMOD Rec.*, 35(1), Marz 2006.

[KRS11] D. Klan, Th. Rohe und K.-U. Sattler. Quantitatives Frequent Pattern Mining in drahtlosen Sensornetzen. In *BTW Workshops*, March 2011.

[KS09] Jürgen Krämer und Bernhard Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM TODS*, 34(1), April 2009.

[LL09] H.F. Li und S.Y. Lee. Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 36(2), 2009.

[SP82] H.J. Schek und P. Pistor. Data structures for an integrated data base management and information retrieval system. In *VLDB*, 1982.

[T+11] Hetal Thakkar et al. SMM : a Data Stream Management System for Knowledge Discovery. In *ICDE*, 2011.

[TMP08] F. Tantono, N. Manerikar und T. Palpanas. Efficiently discovering recent frequent items in data streams. In *SSDM*. Springer, 2008.

[TMSF03] P.a. Tucker, D. Maier, T. Sheard und L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE TKDE*, 15(3), Mai 2003.

[WFH11] Ian H. Witten, Eibe Frank und Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3. Auflage, 2011.