САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет
кафедра системного программирования

Построение базиса ассоциативных правил

Дипломная работа студенки 5 курса
Бардашовой Нины Викторовны

Научный руководитель	..... /подпись/	/А.А. Панин/
Рецензент	..... /подпись/	/Ю.В.Литвинов/
Допустить к защите: Зав. кафедрой	..... /подпись/	/А.Н.Терехов/

Санкт-Петербург
2008 г.

Содержание

1 Введение

В наше время все большее количество компаний, стремясь к повышению эффективности и прибыльности бизнеса пользуются цифровыми (автоматизированными) способами обработки данных и записи их в БД. Это несет в себе как огромные преимущества, так и рождает определенные проблемы, связанные с объемом полученных данных, а именно: при колоссальном увеличении объема полученной информации усложняется ее обработка и анализ, делать выводы по полученным данным становится все сложнее, и вероятность того, что некоторые детали могут быть упущены неумолимо растет.

Данная проблема явилась причиной развития различных подходом и методов, позволяющие проводить автоматический анализ данных. Для решения данных вопросов существуют математические методы, которые и образуют направление DataMining. Термин DataMining часто переводится как добыча данных, извлечение информации, раскопка данных, интеллектуальный анализ данных, средства поиска закономерностей, извлечение знаний, анализ шаблонов, Понятие "обнаружение знаний в базах данных"(Knowledge Discovering Databases, KDD) можно считать синонимом DataMining [1].DataMining-мультидисциплинарная область, возникшая и развивающаяся на базе таких наук как прикладная статистика, распознавание образов, искусственный интеллект, теория баз данных и так далее.

Понятие DataMining, появившееся в 1978 году, приобрело высокую популярность в современной трактовке примерно с первой половины 1990-хгодов.До этого времени обработка и анализ данных осуществлялся в рамках прикладной статистики, при этом в основном решались задачи обработки не больших баз данных.

Информация, найденная в процессе применения методов DataMining, должна быть нетривиальной и ранее неизвестной. Знания должны описывать новые связи между свойствами, предсказывать значение одних признаков на основе других. Найденные знания должны быть применимы и на новых данных с некоторой степенью достоверности. Полезность заключается в том, чтобы эти знания могли принести определенную выгоду при их применении. Поставленные задачи зачастую требуют, чтобы полученные знания были в понятном для пользователя-нематематика виде. Например проще всего воспринимаются логически конструкции типа "если... то...". Алгоритмы, используемый в DataMinig, требуют большого количества вычислений. Раньше это явилось сдерживающим фактором широкого практического применения. Однако сегодняшний рост производительности современных процессоров снял остроту этой проблемы. Теперь за приемлемое время можно провести качественный анализ сотен тысяч миллионов записей.

В данной работе рассматриваются Ассоциативные правила. Область применения, алгоритмы их построения . Область применения ассоциативных

правил достаточно широка. Основная решаемая задача - нахождения закономерности между связанными событиями, сущностями в больших базах данных.Например: дана база данных покупок в супермаркете, по которой можно сделать вывод о том, что 80%. покупателей, приобретающих булку и сахар, так же приобретают молоко. Либо другой пример ассоциативного правила: покупатель, приобретающий банку краски, приобретет кисточку для краски с вероятностью 50%. И таких примеров можно привести огромное количество. Впервые задача поиска ассоциативных правил (association rule mining) была предложена для нахождения типичных шаблонов покупок, совершаемых в супермаркетах, поэтому иногда ее еще называют анализом рыночной корзины (market basket analysis). В [1] дано полное описание общих задач, решаемых ассоциативными правилами, контекста и определений. Как уже было сказано активно данный вопрос начали изучать в 1993г [1].С данной [1] работы в основном и начался интерес к ассоциативным правилам. Были придуманы алгоритмы, которые решают данную задачу. Наиболее распространенный из них - Apriori[2]. Этот и другие алгоритмы (н-р как SETM, AIS, различные модификаций Apiori) обладают таким недостатком, что во время их работы генерируется слишком большое количество промежуточных данных - наборов, из которых выводятся ассоциативный правила, а так же на выходе получается слишком большое кол-во самих правил, что затрудняет анализ полученный результатов. Решение данной задачи и является целью данной дипломной работы. Так же в качестве цели поставлена задача, что в качестве начальных параметров возможно было передавать и булево выражение, которое было бы верно для всех полученных ассоциативных правил. Далее будет описана формализованная модель условия данной задачи.

2 Математическая модель

Построим математическую модель ассоциативных правил, дадим формальные определения. Пусть I=$\{i_1, i_2, ..., i_m\}$ - множество сущностей (*items* - атомарная сущность, определяемая для конкретной базы данных).

База данных: D=$\{t_1, t_2, ..., t_n\}$ - множество транзакций, каждая из которых состоит из множества элементов I из I, и уникального идентификатора - TID. T=(tid,I) Говорится, что транзакция t содержит множество X в том случае, если $X \subseteq I$

Покрытие множества X в D, состоит из множества транзакций, которые содержат X:

$$cover(X, D) := \{tid | (tid, I) \in D, X \subseteq I\}.$$

Поддержкой (support) множества X в D называется количество транзакций в покрытии множества X в базе данных D

$$support(X, D) :=\mid cover(X, D) \mid$$

Частотой (frequency) множества X в D называется вероятность, с которой множество X встречается в траназакциях $T \in D$

$$frequency(X, D) := P(X) = \frac{support(X, D)}{\mid D \mid}$$

Одним из параметров, задаваемым при поиске ассоциативных правил является частота (frequency) данного множества. *minimal support tresholdσ*

Теперь можно определить задачу поиска кандидатов (*Itemset Mining*): Задано множество элементов I, задана база данных D на множестве I, и задано значение минимальной поддержки σ, нужно найти

$$F(D, \sigma) := \{X \subseteq I \mid support(X, D) \geq \sigma\}$$

Ассоциативное правило (association rule) это выражение вида $X \Rightarrow Y$ где X и Y множества, причём $X \cap Y = \{\}$. Это означает, что транзакция, содержащая X, так же содержит в себе и Y, т.е. что из условия X следует условие Y.

Поддержкой ассоциативного правила является поддержкой $X \cup Y$ в D.

Достоверностью (confidence) ассоциативного правила $X \Rightarrow Y$ в D является условная вероятность , что при условии того что множество X поддерживается данной транзакцией, то и множество Y так же поддерживается данной транзакцией.

$$confidence(X \Rightarrow Y, D) := P(Y \mid X) = \frac{support(X \cup Y, D)}{support(X, D)}$$

Правило называется достоверным, если его достоверность превосходит минимально заданную границу γ

теперь мы можем определить задачу поиска ассоциативных правил (Association Rule Mining): Задано множество элементов I, задана база данных D над I, заданы так же минимальные значения достоверности и поддержки σ, γ, необходимо найти:

$$(R)(D, \sigma, \gamma) := \{X \Rightarrow Y \mid X, Y \subseteq I, X \cap Y = \{\}, X \cup Y \in F(D, \sigma), \\ confidence(X \Rightarrow Y, D) \geqslant \gamma$$

Как уже было сказано, одним из условий поставленной задачи является и то, что набор полученных ассоциативных правил должен удовлетворять некоторому заданному шаблону. По сути это булева функция, которая должна быть верна для данного правила. Дадим математическую формулировку. Пусть B - булево выражение на множестве I, зная, что любое булево выражение можно привести к дизъюнктивной нормальной форме (DNF) можем предположить, что B имеет вид $D_1 \vee D_2 \vee ... \vee D_m$, где каждый дизъюнкт D_i имеет вид $a_{i1} \wedge a_{i2} \wedge ... \wedge a_{in}$, где каждый элемент a - это отрицание ($\rightharpoonup a$) или утверждение (a), где $a \in I$.

Итак, задача алгоритма, по данной транзакционной базе D и заданному ограничению B построить базис ассоциативных правил, поддержка и достоверность которых будет $\geq$ минимально заданным значениям.

Проблема нахождения ассоциативных правил содержит в себе 2 подзадачи:

1. Найти все комбинации наборов, чья поддержка больше чем заданная (frequent).

2. На основе полученных наборов, учитывая минимальную заданную достоверность правила, получить конечный набор ассоциативных правил.

Нахождение всех наборов с поддержкой выше заданной - проблема весьма нетривиальная из-за их количества, притом оно увеличивается экспоненциально. Если мощность множества $\|I\| = m$, то возможное количество таких наборов 2^m.

С вычислительной точки зрения вторая подзадача полностью зависит от 1й, с-но большинство работ в данной области направлено на оптимизацию решения первой задачи. Но и 2я задача несет в себе некоторое количество трудностей. Зачастую, данные в транзакционных базах данных являются сильно связанными, и несмотря на высокое значения параметра достоверности правила и задания булевого выражения , в итоге получается слишком большое количество ассоциативных правил, по которым очень сложно сделать какие-либо достоверные выводы и выявить интересующие закономерности. Посему, в данной дипломной работе так же ставится задача анализа и реализации подхода не только оптимизации первой подзадачи с вычислительной точки зрения, но и сокращения количества полученных ассоциативных правил, без возможной потери данных. По сути, построение некоторого базиса ассоциативных правил.

3 Обзор существующих алгоритмов

Данная работа ставит перед собой несколько подзадач, существуют алгоритмы, которые решают эти подзадачи независимо друг от друга:

3.1 задача оптимизации поиска наборов с поддержкой выше заданной.

Существует достаточно большое количество алгоритмов, решающих проблему поиска ассоциативных правил. Наиболее широким применением пользуется алгоритм Apriori[todo bibl]. Именно этот алгоритм использовался в проекте, на базе которого проводилась данная работа. Но данный подход обладал недостатками, описанными в пункте (todo разбить все более мелко, а так параграф выше). Алгоритм Apriori, генерирующий набор с указанной поддержкой основывается на следующем утверждении: *все*

подмножества множества с поддержкой больше заданной, так же имеют такую поддержку. И все множества, содержащие в себе множества с поддержкой ниже заданной, так же имеют такую поддержку, другими словами поддержка любого набора элементов не может превышать минимальной поддержки любого из его подмножеств. По сути это свойство антимонотонности. Формально это можно записать так:

$$confidence(XZ \Rightarrow Y \cup Z) \leq confidence(X \Rightarrow Y)$$

Это верно для любых X,Y,Z, таких что $X \cup Y = \{\}$

При использовании алгоритма Apriori, объекты даются нам в лексикографическом порядке. Наборы с поддержкой выше заданной подсчитываются итеративно, по возрастанию размера. Процесс проходит k итераций, где k- это количество элементов в самом большом наборе. Для каждой итерации $i \leq k$ подсчитываются все наборы размера i. На первой итерации подсчитывается L_1 с соответствующей поддержкой. Каждая i-я итерация состоит из 2х частей:

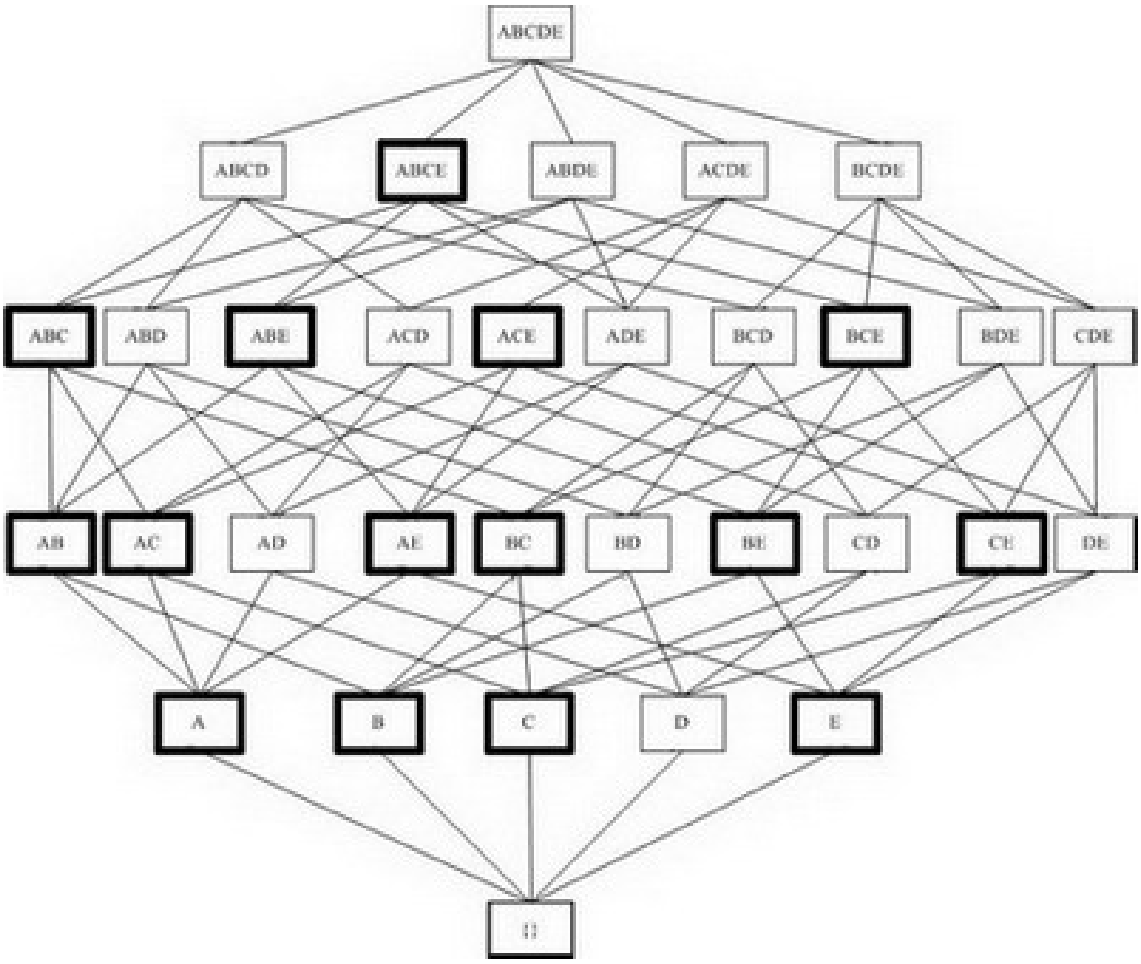
1. Генерируется множество кандидатов C_i из множества L_{i-1}.
2. Подсчитывается поддержка каждого из кандидатов, если она больше чем заданная величина, то этот кандидат добавляется в искомое множество.

```
1 L_1= {Frequent 1-itemset};
2   for(k=2;L_(k-1)!=0;k++){
3   C_k=AprioriGenerationFunction (L_{k-1});    //Генерируемнабор кандидатов k-ур
4   forAll (Transactions t: D) {
5     C_t=subSet(C_k,t);
6        forAll (Candidates c:C_t){ //Кандидаты,содержащиеся в данной транзакции
7           c.count++;
8              }
9        }
10    }
11 L_k={c in C_k | c.count> minsupport};
```

D итоге, искомое множество наборов с поддержкой выше заданной, это $\bigcup_k L_k$.

TID	Items
1	A C D
2	B C E
3	A B C E
4	B E
5	A B C E

Рассмотрим пример: Дана база данных транзакций (5 транзакций), множество I, состоящее всего из 5ти элементов. При работе алгоритма получается, что генерируется 32 кандидата. И при заданной минимально поддержке в 40% только 15 кандидатов подходят. Но для того, чтобы это проверить, нужно подсчитать поддержку каждого из множеств (обращение к базе данных). В таком виде данный подход является непрактичным при большом значении m



Для оптимизации данного подхода был придуман алгоритм Closed [todo bibl], который основывается на совершенно другом подходе. Рассмотрим его базовые идеи и определения.

Определим понятие контекста DataMining - это тройка вида $D = (O, I, R)$ где O и I это конечные множества объектов, и элементов из которых состоят искомые наборы. $R \subseteq OxI$ - бинарное отношение. По сути это отношение между набором транзакций в базе данный и набором элементов.

Определение (todo - задать шаблоны для определений) - Отображение Галуа. Пусть $D = (O, I, R)$ - контекст, для $O \subseteq O$ и $I \subseteq I$ мы определяем:

$$f(O) : 2^O \to 2^I f(O) = \{i \in I \mid \forall o \in O, (o, i) \in R\}$$

$$g(I) : 2^I \rightarrow 2^O g(I) = \{o \in O \mid \forall i \in I, (o,i) \in R\}$$

Пара отображений (f, g) это отображение Галуа. Операторы $h = fog$ и $h' = gof$ - Операторы Галуа. [todo bibl].

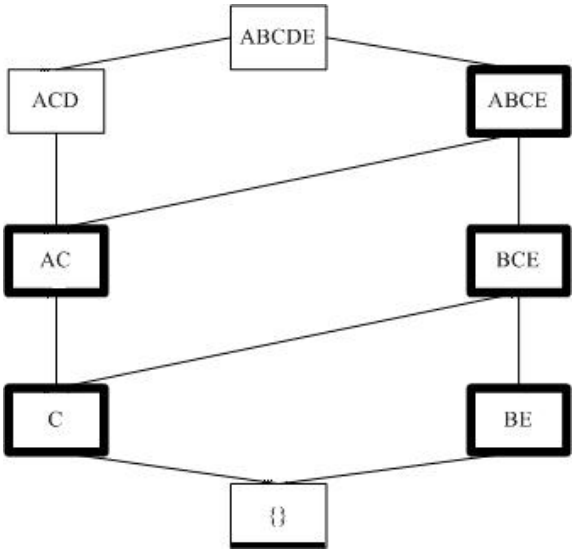
Определение. Пусть $C \subseteq I$ это множество элементов из D. C - замкнутое множество, если $h(C) = C$.Минимальное (самое маленькое) замкнутое множество, содержащее набор I, получается с помощью применения оператора h к I.

Определение. Пусть C - множество замкнутых наборов, полученных из D с использованием отображения Галуа. Тогда пара $L_C = (C, \leq)$ - замкнутая решетка.Данная структура обладает следующими свойствами:

1. Существует частичный порядок над элементами решетки, такой что для $\forall$ элементов $C_1, C_2 \in L_C$ $C_1 \leq C_2$ если $C_1 \subseteq C_2$.

2. Для всех элементов решётки существуют infinum(Join) и supremim(Meet). Т.е. для $\forall S \subseteq L_C$:
$$Join(S) = h(\bigcup_{C \in S} C)$$
$$Meet(S) = \bigcap C \in SC$$

Алгоритм Close фундаментально отличается от существующих алгоритмов. Он основывается на идеи упрощения решетки поиска для поиска наборов с поддержкой выше заданной. Замкнутое множество, это максимальное (самое большое) множество элементов, соответствующее множеству объектов. Например, рассматривая пример, приведённый выше, набор $\{B, C, E\}$ замкнутое множество т.к. это максимальное множество, соответствующее объектам $\{2, 3, 5\}$. Поддержка набора $\{B, C, E\} = 3 \geq minsupport$.В терминах задачи базы данных покупок это означает, что 60% покупателей покупают в основном товары B,C,E. По сути замкнутая решетка является подграфом графа работы алгоритма Apriori (см рисунок.). На рисунке представлен граф(замкнутая решетка) базы D, минимальное заданной поддержкой = 2.



Использование замкнутых решеток для нахождения множества наборов с поддержкой выше чем заданная - является значительным улучшением с точки зрения производительности. Это исходит из того, что количество множеств с необходимой поддержкой намного больше чем количество замкнутых множеств с такой же поддержкой, т.е. мы по сути минимизируем пространство поиска.

Подитожим всё вышесказанное, приведя основные свойства, на которых строится Closed алгоритм:

1. Все подмножества наборов с поддержкой выше заданной имеют такую же поддержку.

2. Все множества, содержащие в себе множества с поддержкой ниже заданной, так же имеют такую же поддержку.

3. Все замкнутые подмножества замкнутых множеств с поддержкой выше заданной имеют такую же поддержку.

4. Все замкнутые множества, содержащие в себе замкнутые наборы с поддержкой ниже заданной, так же имеют такую же поддержку.

5. Максимальное (самое большое) множество с поддержкой выше заданной эквивалентно максимальному (самому большому) замкнутому множеству с поддержкой выше заданной.

6. Поддержка не замкнутого множества I эквивалентна поддержке минимальному замкнутому множеству, которое содержит набор I.

С подробным доказательством данных высказываний можно ознакомится в [todo bibl]

Основываясь на этих свойствах можно описать базовые шаги алгоритма Close:

1. В базе данных найти все замкнутые наборы с поддержкой больше чем заданная.

2. Найти все искомые наборы из наборов найденных на 1м этапе. Эта фаза состоит из из этапа генерирования всех подмножеств максимального замкнутого множества с необходимой поддержкой, и подсчета их поддержек, основываясь на свойстве 6.

3. По полученным множествам получить ассоциативные правила, достоверность которых выше чем заданная.

Самая ресурсоёмкая часть алгоритма - 1. После ее выполнения нам уже не потребуется ни одно обращение к базе данных.

Используя идею данного алгоритма мы можем решить первую из поставленных задач этой работы - уменьшить ресурсоёмкость фазы генерирования необходимых наборов.

3.2 Булево выражение - как ограничение на искомые правила

Вторая поставленная задача (чтобы ассоциативное правило удовлетворяло некоторому заранее заданному булеву выражению) решается путём проверки истинности данного выражения на каждом этапе генерирования следующего набора возможных кандидатов.

3.3 Построение Базиса

Для описания алгоритма решения последней из поставленных задач (уменьшение количества генерируемых ассоциативных правил, т.е. построения их базиса), мы дадим несколько определений. По сути, мы определим те же понятия, что и в главе 2, но в терминах

3.3.1 Семантика ассоциативных правил

Отображений Галуа и замкнутых множеств.

Пусть $I \subseteq I$ (todo изменить шрифт второй I) множество из D. Тогда поддержкой множества I в D будет:

$$support(I) = \frac{\|g(I)\|}{\|O\|}$$

Множество наборов с поддержкой выше заданной:

$$L = \{I \subseteq I \mid support(I) \geq misupport\}$$

Максимальное множество с поддержкой выше заданного:

$$M = \{I \in L \mid \nexists I' \in L, I \subset I'\}$$

Множество всех замкнутых множеств над базой данных D

$$FC = \{C \subseteq I \mid C = h(C) \text{и} support(C) \geq minsupport\}$$

Утверждение 1 *.Все искомые наборы с поддержкой выше заданной, а так же достоверные ассоциативные правила могут быть получены из FC. Это утверждение следует из свойст 5 и 6.*

Множество всех максимальных замкнутых наборов:

$$MC = \{C \in FC \mid \nexists C' \in FC, C \subset C'\}$$

Ассоциативное правило, это выражение вида $I_1 \Rightarrow I_2$ где $I_1, I_2 \subset I$ и $I_1 \cap I_2 = \{\}$.Поддержка и достоверность ассоциативного правила $r : I_1 \Rightarrow I_2$ выражаются следующим образом:

$$support(r) = \frac{\|g(I_1 \cup I_2)\|}{\|(O)\|}$$

,

$$confidence(r) = \frac{support(I_1 \cup I_2)}{support(I_1)} = \frac{\|g(I_1 \cup I_2)\|}{\|g(I_1)\|}$$

Искомое множество ассоциативных правил при заданных значениях поддержки и достоверности, возможно определить через множество всех максимально замкнутых наборов:

$$AR(D, minsupport, minconfidence) = \{r : I_2 \Rightarrow (I_1 - I_2) \mid I_2 \subset I_1, \\ I_1 \in L = \bigcup_{C \in MC} 2^C \text{и} confidence(r) \geq minconfidence\}$$

3.3.2 Базис ассоциативных правил. Определение

Как уже говорилось ранее, мы хотим решить задачу - уменьшения количества полученных ассоциативных правил. По сути получить из базис. Такой алгоритм описан в [todo bibl]. Мы можем построить так называемый Люксембургский базис для ассоциативных правил. Дадим некоторые определения:

Определение 1 *базис ассоциативных правил*

$$LB = \{(r, support(r), confidence(r)) \mid r = I_1 \Rightarrow I_2, I_1, I_2 \in FC, \\ I_1 \prec I_2, confidence(r) \geq minconfidence, support(I_2 \geq minsupport)\}$$

На основе приведённых ниже правил из базиса мы можем получить полный набор ассоциативных правил, который бы получился при простом применении алгоритма Apriori.

- $support(X \to Y) = support(X \to Y$ $Z)$, для $\forall Z \subseteq X \subseteq M, Y \subseteq M$.
- $support(h(X) \to h(Y)) = support(X \to Y)$, для $\forall X, Y \subseteq M$
- $if confidence(X \to Y) = p$ и $confidence(Y \to Z) = q \Rightarrow confidence(X \to Y) = p * q$ для $\forall X \subset Y \subset Z$.
- $support(X \to Y) = support(Y \to Z)$ для $\forall X, Y \subseteq Z$.
- $confidence(X \to X) = 1$ для $\forall X \subseteq M$

Доказательства данных свойств приведены в [todo bibl]

Основываясь на вышеприведенных свойствах можно доказать теорема о том, что люксенбургский базис действительно является базисом ассоциативных правил, и с-но из него могут быть получены все ассоциативные правила. Более того, Люксенбургский базис является минимальным.

Утверждение 2 *Все возможные ассоциативные правила, их поддержки и достоверность могут быть получены из Люксенбургского базиса, используя приведенные выше правила. Причем данный базис является минимальным.*

Доказательство:
todo - дописать!!!

□

4 Эффективный алгоритм построение базиса ассоциативных правил

И наконец, рассмотрев три подхода, который каждый в отдельности решает одну из поставленный задач, мы можем написать алгоритм, который будет генерировать базис асоциативных правил, с учетом дополнительных ограничений (как-то тот факт, что ассоциативное правило должно удовлетворять некоему заранее заданному булеву выражению). Причем данный алгоритм более оптимален, по сравнению с Apriori.

Перед тем как описать алгоритм, дадим определение термина, который используется непосредственно в самом алгоритме. Генератор (generator) p замкнутого множества c - это минимальный набор, при применении к которому оператора h будет получено множество c, т.е. $h(p) = c$.

```
// Генерируем множество всевозможных наборов
1 FCC_1.generators = {1-itemsets};
2 FOR (i=1;FCC_i.generator!=0;i++){
3    FCC_i.closures = null;
4    FCC_i.supports = 0;
5    FCC_i = GenClosure(FCC_i);                // Находим замыкания
6    FORALL(Condidates c:FCC_i){
7        if(c.support>=minsupport){            // Если набор с поддержкой больше м
8            FC_i.add(c);                      // Добавляем его в набор кандидатов
9            }
10   }
11   FCC_i+1=GenGenerator(FC_i);               // Генерируем следующее "поколение
12   FC.addall(FC_i);
13 }
  // Ищем базис ассоциативных правил LB
14 FOR (i=2;i<k;i++)                           // k - размерсамого большого множе
15    FORALL(FreqItemset L:FC_i){
16        FOR (j=0;j<i;j++){S_j = Subset(FC_j,L);}
17        FOR (j=i-1;j>=1;j--){
18            FORALL(FreqItemSet L':S_j){
19            conf=L.support/L'.support;
20            if(conf>=mincong){
21                LB.add(new Rule(L',L\L');)
22                }
23            for(l=j;l>=1;l--){
24                S_l=Sl\Subset(S_l,L');}
25                }
26            }
27        }
```

Сначала инизиализируется множество генераторов. По сути это просто элементы множества I. Каждая следующая итерация состоит из 3х фаз:

- К каждому генератору применяется оператор замыкания, чтобы найти кандидата и его поддержку (Line -5).
- Множество полученный кандидатов сокращается, путем добавления в результирующий набор только тех, поддержка которых выше заданной.
- Ищем следующее "поколение"генераторов

Цикл завершится, когда FCC_i будет пусто. К этому моменту все наборы, с поддержкой выше заданной будут найдены. Доказательство корректности данного алгоритма [todo bibl].

Далее ищем базис ассоциативных правил. Итеративно мы беребираем все наборы $L \in FC_i$ (напомним, что в FC содержатся только те наборы, поддержка которых выше минимальной), далее определяем, такой набор $L' \in \bigcup_{j<1} FC_i$ и $L' \subset L$, что у полученного ассоциативного правила вида $L' \rightarrow L \backslash L'$ достоверность будет выше заданной. На каждой i^{th} итерации рассматриваются все наборы из FC_i. Для каждого множества $FC_j, 1 \leq j < i$ находится множество S_j, содержащие в себе все наборы в FC_i, которые являются подмножествами L. Далее рассмотрим их в порядке убывания (по их размерности).Для каждого такого подмножества $L' \in S_j$ считается достоверность ассоциативного правила вида $r = L' \rightarrow L \backslash L'$. Если достоверность правила подходящая, то данное правила добавляется к базису LB, и все подмножества L' удаляются из $S_l, l < j$. В конце работы алгоритма множество LB содержит в себе базис ассоциативных правил. Доказательство корректности данного алгоритма приведено в [todo bibl].

Рассмотрим некоторые функции, использованные в данном алгоритме.

4.1 функция GenClouse()

функция GenClouse() - Её задача - для каждого генератора определить его замкнутое множество, и подсчитать поддержку данного множества. Перейдя от математического определения к более конкретному, можно сказать, что замыкание генератора I, полученное применением к нему оператора h(I), это ничто иное, как пересечение всех объектов в базе данных, которые содержат I:

$$h(I) = \bigcap_{o \in O} \{f(\{o\}) \mid I \subseteq f(\{o\})\}$$

```
1 FORALL(Object o:O){
2 G_o=Subset(FCC_i.generator,f({o}));
3 FORALL(Generator p:G){
4   IF (p.closue==null){
5       p.closue=f({o})
6       }else{
7       p.closue=p.closue.interseq(f({o}));}
8    p.support++;
9   }
10}
11 return {c in FCC_{i}|c.closue!=null and !BoolExpr(p)}; // проверяет. что набор
```

Функция работает следующим образом: Для каждого объекта o из D создается G_o, который содержит все генераторы из FCC_i, которые являются подмножествами $f(\{o\})$. Далее для каждого генератора подсчитывается его замыкание и поддержка.

4.2 функция GenGenerator()

Функция в качестве аргумента принимает набор FC_i и возвращает набор (основываясь на лемме 1) генераторов FCC_{i+1}, которые будут использованы на $i+1$ итерации, чтобы получить множество кандидатов.Функция сначала генерирует все потенциально возможные $i+1$ набор генераторов, далее (лемма 2), удаляются те генераторы, которые производят неинтересные для нас наборы (наборы с поддержкой меньше заданной, либо те наборы, которые могут быть получены из других генераторов).

Утверждение 3 *Пусть существует 2 набора* I_1, I_2, *тогда* $h(I_1 \cup I_2) = h(h(I_1) \cup h(I_2))$.

Утверждение 4 *Пусть* I *- это* i*-генератор, а* $S = \{s_1, s_2, ..., s_{i-1}\}$ *это* $(i-1)$ *подмножество* I, *где* $\bigcup_{s \in S} s = I$. *Если* $\exists s_a \in S$, *такие что,* $I \subseteq h(s_a)$, *тогда* $h(I) = h(s_a)$.

```
1 insert into FCC_i+1.generator
2 select p.item_1,p.item_2,...,p.item_i,q.item_i
3 from FC_i.generator p, FC_i.geberator q
4 where p.item_1=q.item_1,...,p.item_i-1=q.item_i-1, p.item_i<q.item_i;
5. FORALL (Generator p:FCC_i+1.generator){
6    FORALL(Set s:p.i-subset){
7        if(!FC_i.contain(s))
8            FCC_i+1.remove(s)
9    }
10 }

11 FORALL (Generator p: FCC_i+1.generator){
12   S_p=SubSet(FC_i.generator,p)                    // generator'ы, которые я
13   FORALL(Generator s:S_p){
14       if (s.closure.subset(p)){
15           FCC_i+1.generator.delete(p);}
16   }
17 }
18 return FCC_i+1;
```

4.3 функция Subset(SetOfSets, set)

функция Subset(SetOfSets, set) возвращает все множества из SetOfSets, которые являются подможножествами set. Реализация данной функции будет описана позднее, т.к. она напрямую зависит от способа представления SetOfSet в данном алгоритме.

4.4 Пример работы алгоритма

В завершении приведём пример работы алгоритма, за исходную базу данный возмём пример, уже разбиравшейся в главе (todo ссылки).

Исходный пример: множество $I = \{A, B, C, D, E\}$, пусть *minsupport*=40% (минимальная поддержка набора), а *minconfidence*=1/2. Транзакционная база данный выглядит следующим образом:

TID	Items
1	A C D
2	B C E
3	A B C E
4	B E
5	A B C E

$\rightarrow$ FCC_1

Generator	Closure	Support
{A}	{AC}	3
{B}	{BE}	4
{C}	{C}	4
{D}	{ACD}	1
{E}	{BE}	4

$\rightarrow$ FC_1

Generator	Closure	Support
{A}	{AC}	3
{B}	{BE}	4
{C}	{C}	4

$\rightarrow$ FCC_2

Generator	Closure	Support
{AB}	{ABCE}	2
{BC}	{BCE}	3

$\rightarrow$ FC_2

Generator	Closure	Support
{AB}	{ABCE}	2
{BC}	{BCE}	3

Искомое множество замкнутых наборов с поддержкой выше заданной: FC

Closure	Support
{AC}	3
{BE}	4
{C}	4
{ABCE}	2
{BCE}	3

Далее, подсчитав базис ассоциавных правил, мы получаем

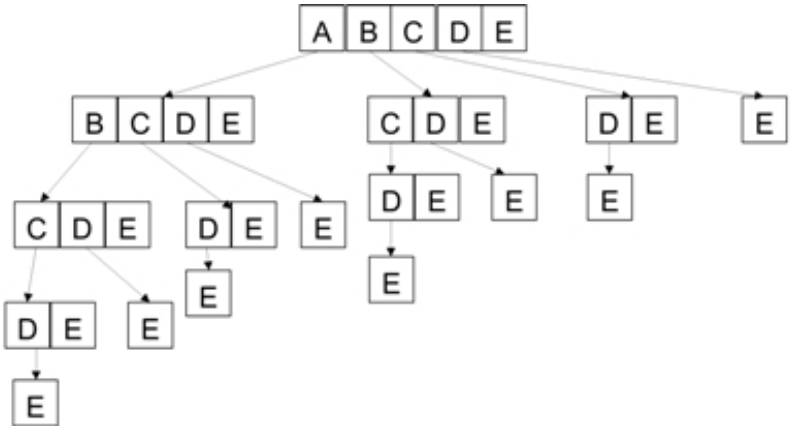
Association Rule	Support	Confidence
$BCE \rightarrow A$	0.4	2/3
$AC \rightarrow BE$	0.4	2/3
$BE \rightarrow C$	0.6	3/4
$C \rightarrow BE$	0.6	3/4
$C \rightarrow A$	0.6	3/4
$\{\} \rightarrow BE$	0.8	4/5
$\{\} \rightarrow C$	0.8	4/5

Для сравнения, Если бы мы для этой же базы данный, с такими же начальными условиями применили алгоритм Apriori, то получили бы 67 ассоциативных правил.

5 реализация

5.1 структура данных

Выбор структуры данных, которая будет хранить множество наборов кандидатов (генераторы, их замыкания и поддержки) - один из основополагающих факторов, который определяет эффективность данного алгоритма. В данной работе была использована структура префиксного дерева. В дереве, каждому k-набору соответствует вершина дерева, находящееся на глубине k, а путь от корня к данной вершине, длиной k-1, соотвествует набору элементов в данном множестве. В приведённом примере показано префиксное дерево для множества A, B, C, D, E.

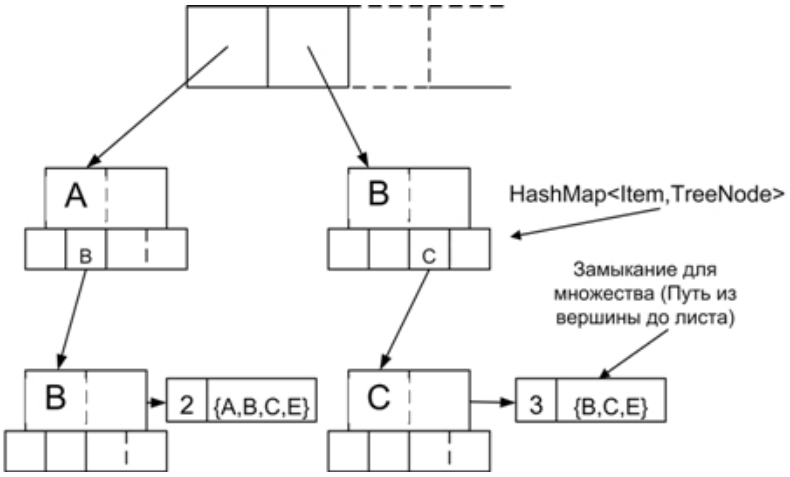


. Корнем дерева является пустая вершина (на рисунке она не обозначена). Все наборы размерности 1 - составляют первый уровень дерева и т.д. Каждый набор размерностью k присоединяется к вершине на уровне k-1, где пусть до это вершины, является префиксом данного набора. При такой структуре данных генерация нового элемента весьма упрощается.

```
1 insert into FCC_i+1.generator
2 select p.item_1,p.item_2,...,p.item_i,q.item_i
3 from FC_i.generator p, FC_i.geberator q
4 where p.item_1=q.item_1,...,p.item_i-1=q.item_i-1,
p.item_i<q.item_i;
```

Таким образом на i-м шаге, чтобы получить следующий набор нам достаточно взять для каждого пути в дереве длиной i добавить соответствующие элементы в лексикографическом порядке.

Рассмотрим подробнее структуру данных:



На данном рисунке приведён пример структуры данных для набора FCC_2, из примера приведённого выше. Как уже было сказано, то по сути каждое ребро в дереве соотвествует какому-либо элементу. Таким образом двигаясь от корня дерева к его листьям мы можем получить требуемый набор. В данном контексте - генератор, это путь в дереве. Замыкание генератора(пути), и его поддержка записаны в узле, которым заканчивается путь. Каждая вершина дерева , содержит в себе множество HashMap<Item,TreeNode>, которое является набором потомков для данной вершины.

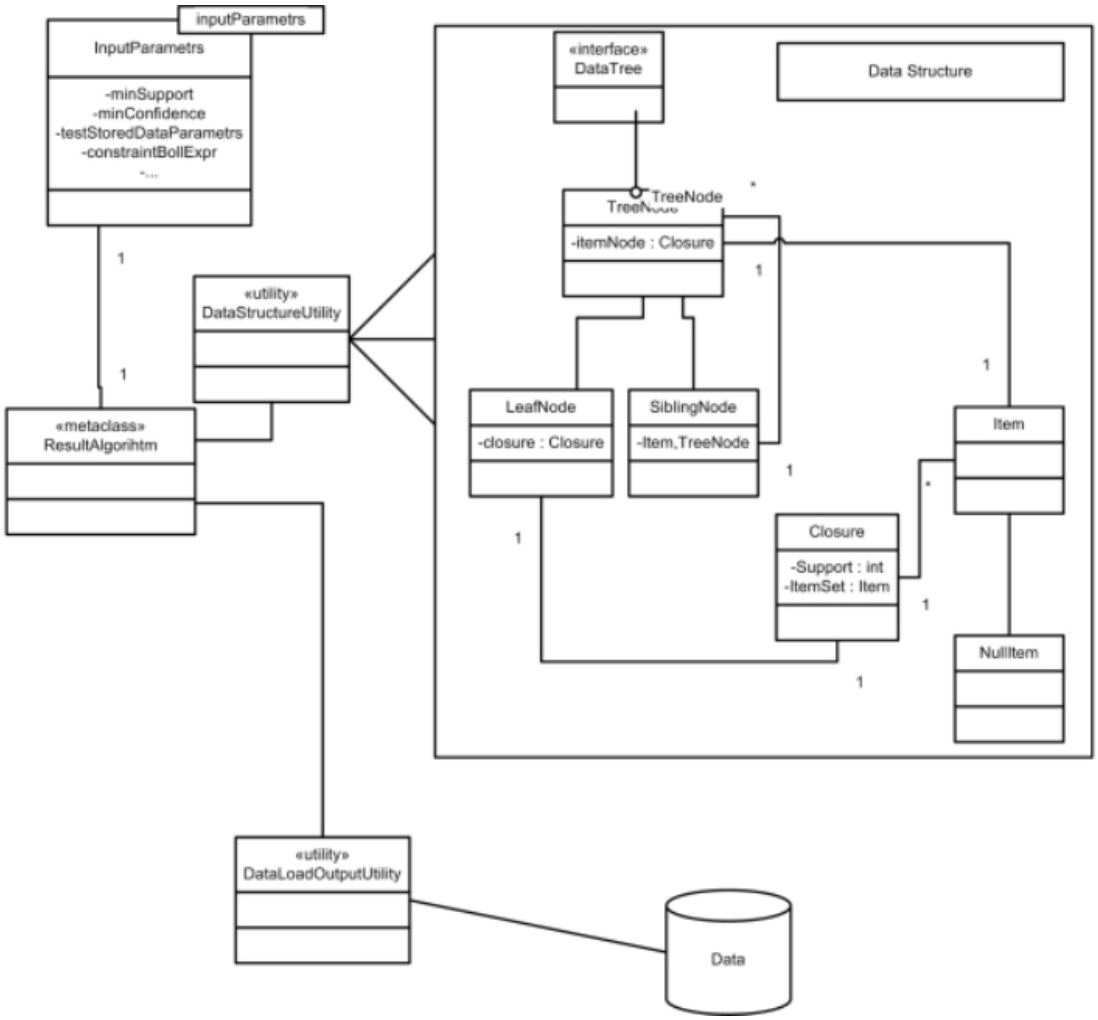
Теперь мы можем более подробно описать функцию *Subset*, упоминающуюся в [todo ref] параграфе. Функция в качестве аргумента принимает принимает множество генераторов (путей) G и некоторый набор c. В качестве результата она возвращает такой набор $p \in G$, где p является подмножеством c. Таким образом задача сводится к задаче нахождения множества путей в дереве для данного набора.

Добавление элемента в дерево происходит следующим образом. Для данного набора ищется пусть в дереве, если он существует, то мы увеличиваем значения параметра поддержки для замыкания данного набора на 1, строим пересечение текущего замыкания и транзакции. если не существует, то добавляем новый узел в дерево, а замыкание становится равно 'текущей' транзакции (в даннном примере это $f(o)$).

```
FORALL(Generator p:G){
  IF (p.closue==null){
      p.closue=f({o})
      }else{
      p.closue=p.closue.interseq(f({o}));}
   p.support++;
  }
```

Для дальнейшего сравнения производительности был имплементирован алгоритм Apriori, в котором так же была использована структура префиксного дерева.

Рассмотрим более широко приложение.



. На данной диаграмме представлена иерархия классов, а так же представленный основные классы, которые выполняют утилитные функции по работе с вышеописанной структурой, а так же предоставляющие методы работы с базой данных.

5.2 структура приложения

diplom.datamining.dataStructure TODO

diplom.datamining.utilites TODO

diplom.datamining.algorithm TODO

5.3 используемые средства

Прототип алгоритма был реализован на языке программирования - Java 5.0. Средой разработки была IDEA 7.0.1, Используемая СУБД - MySql 4.0.18. Данный выбор был обусловлен тем, что автор имеет большой опыт разработки коммерческих приложений с использованием вышеперечисленный технологий.

Но в дальнейшем в рамках исследования данный алгоритм будет переписан на C++, в следствии чего ожидается повышение общей производительности за счёт использования менее тяжеловесной платформы.

В программе широко используются такие шаблоны проектирования как Facade, Visitor,Factory Method.

6 Эксперименты и полученные результаты

Для начала хотелось бы сказать несколько слов о базе данных, используемой для проведения экспериментов. Она была получена путем искусственной генерации, и имеет следующий вид:

TID	A	B	C	D	E	F	G	H	I	K	...
1001	1	0	1	1	0	0	1	1	0	0	1
1002	1	0	0	1	1	0	0	1	1	1	0
1003	1	1	0	0	0	1	1	1	1	1	0
...	.	.	.	.	.	.	.	.	.	.	.

База данных, на которой проводились эксперименты, содержала 10^6 транзакций. Множество элементов I - латинский алфавит.

В дальнейшем существует вероятность внедрения данного решения в программный продукт компании, где в данный момент работает автор. В этом случае будет стоять задача адаптации данного алгоритма уже для 'настоящих' данных.

6.1 результаты экспериментов

Эксперименты проводились на машине со следующими вычислительными мощностями: AMD Athlon(tm) 64 Processor 3200+ 2.21Ghz, 1,5Gb of Ram Приведём некоторые результаты экспериментов

6.1.1 время работы алгоритмов

6.2 количество полученных правил

7 заключение

В данной работы были рассмотрены вопросы связанные с построением ассоциативных правил. На основе различных подходов был разработан алгоритм, включающий в себя внутренние улучшения (такие как улучшение производительности) и решение важной задачи - уменьшение количества получаемых ассоциативных правил. Было доказано, что полученный набор

является базисом множества ассоциативных правил. Был разработан прототип алгоритма и проведены исследования, сравнивающие результаты, полученные при использовании алгоритма Apriori и данного.

todo - сказать про возможность внедрения в production?

В заключении хотелось бы сказать про возможные направления исследования в данной области

- Зачастую данные представимы в иерархическом виде, отсюда возможны изменения алгоритма, учитывающие изначальную иерархию.
- Интересно применить данный подход не на искусственно полученных данных, а на БД, описывающих реальные события.
- как уже было сказано, по причине некоторой тяжеловесности Java платформы интересны будут результаты работы алгоритма на языке C++.
- Прототип. реализованный в рамках этой работы является однопоточным приложением. Интересно было бы исследовать возможность распаралеливания.

Сделать - библиографию!!!