Извлечение частых множеств из транзакционных данных вертикального формата

К.т.н., профессор В.Г.Подлесных

**Аннотация**

Извлечение частых множеств из очень больших баз данных лежит в основе быстрого обнаружения ассоциативных правил, web-технологий, классификации и других проблем управления данными. Описана реализация предложенного алгоритма извлечения частых множеств из транзакционных данных посредством конвертирования их в вертикальный формат и нахождения множеств пересечения идентификаторов транзакций, в которых заданный элемент или комбинация отсутствуют. Метод использует объектно-ориентированный подход. Сравнительным тестированием показано, что скорость конвертирования данных в вертикальный разностный формат не уступает скорости чтения базы быстрейшим на сегодня алгоритмом FP-growth.

**Ключевые слова**: частое множество, большая база данных, ассоциативное правило, транзакции вертикального формата, объектно-ориентированный подход, пересечение идентификаторов транзакций, извлечение знаний.

**Key words:** frequent set, large dataset, association rule, vertical format transaction, object-oriented approach, transaction identifier intersections, knowledge extraction.

 Начало данному направлению положила задача анализа покупательских корзин в супермаркетах [1]. Под транзакцией подразумевается операция покупки клиентом некоторого набора предметов. Записи о наборах в транзакционных базах имеют, как правило, различную длину, и для представления в виде прямоугольной таблицы отсутствующие в данной транзакции предметы заменяются нулями. Таким форматом представления данных пользуется известный алгоритм Apriori для получения ассоциативных правил [2,3]. В алгоритме Apriori на первом этапе используется комбинационный метод генерации потенциальных кандидатов в частые множества с последующим размещением их в хеш-дереве. Путем расположения элементов в лексикографическом порядке (сортировки) и использования правила анти монотонности, авторам удалось сократить число кандидатов, исключив потенциально не частые комбинации. Правило анти монотонности гласит, что любой частый потомок можно сформировать только из частых подмножеств предыдущего уровня, поэтому после соединения частых множеств на уровне k дополнительно проверяют, не появились ли в этой новой комбинации подмножества, не вошедшие в подмножества частых на предыдущем уровне k -1. Если появилось хотя бы одно такое подмножество, то вся новая комбинация не является частой и исключается из дальнейшего комбинирования. Несмотря на эти меры, число кандидатов остается очень большим. На втором этапе пропускают каждую транзакцию через хеш-дерево и увеличивают счетчики тех кандидатов, которые встречаются в транзакции. Если после пропускания всех транзакций через дерево кандидатов содержание счетчика превышает или равно заданному уровню минимальной поддержки minsup, то соответствующая комбинация считается частой. Еще одним фундаментальным принципом генерации кандидатов в Apriori является правило общего (k – 2) префикса у соединяемых комбинаций, где k – уровень комбинирования. Например, для получения комбинации ABC на третьем уровне следует соединять частые двойные комбинации АВ и АС с одним общим элементом в префиксе, в данном случае, А, но не имеет смысла соединять частые АВ и ВС с разными префиксами, что приведет к повтору.

Самым быстрым считается алгоритм FP-growth [4]. Этот алгоритм основан на сортировке частых элементов в транзакциях и построении “растущего” дерева отсортированных транзакций, начинающихся с редко встречающихся элементов. Определение поддержки комбинаций осуществляют путем вычисления последовательностей проекций дерева по отдельным элементам. Преимуществом в сравнении с Apriori является отсутствие операции формирования кандидатов частых множеств, что и ускоряет процесс поиска. Помимо этих примеров, в которых используется традиционный горизонтальный формат представления записей, для поиска частых множеств можно использовать вертикальный формат [5,6]. При этом поля таблицы (столбцы) представляют наименования элементов, а строки содержат идентификаторы транзакций, в которых данный элемент присутствует.

Вертикальный формат упрощает определение поддержки элементов и комбинаций, не требуя формирования и хранения кандидатов.

Рассмотрим простой пример. Пусть имеется пять элементов: А, В, С, Д, Е и шесть транзакций, помеченных идентификаторами tid (рис. 1,а).

 a b c

|  |  |
| --- | --- |
| tid | Элементы транзакций |
|  1 | {А,В,D,Е} |
| 2 | {В,С,Е} |
| 3 | {А,В,D,Е} |
| 4 | {А,В,С,Е} |
| 5 | {А,В,С,D,Е} |
| 6 | {В,С,D} |

|  |
| --- |
|  tidsets |
| А | В | С | D | Е |
| 1 | 1 | 2 | 1 | 1 |
| 3 | 2 | 4 | 3 | 2 |
| 4 | 3 | 5 | 5 | 3 |
| 5 | 4 | 6 | 6 | 4 |
|  | 5 |  |  | 5 |
|  | 6 |  |  |  |

|  |
| --- |
|  diffsets |
| A | B | C | D | E |
| 2 | {} | 1 | 2 | 6 |
| 6 |  | 3 | 4 |  |

Рисунок 1. Различное представление транзакций: а – обычное горизонтальное; b, c - вертикальное: с перечислением в столбцах значений идентификаторов транзакций, в которых данный элемент присутствует (tidsets, b) или отсутствует (diffsets, c).

При заданном уровне минимально допустимой поддержки элементов и их комбинаций, равной 50%, все пять исходных элементов являются часто встречающимися, так как содержатся не менее чем в четырех транзакциях каждый (рис.1,b). Для определения присутствия двойных комбинаций с помощью tidsets надо искать множества пересечения идентификаторов транзакций соответствующих элементов, которые будем обозначать как t. Например, для определения множества идентификаторов транзакций, поддерживающих комбинацию АВ, следует найти пересечение множеств:

t(A)t(B) = {1345}{123456} = {1345}. Легко убедиться, что эта комбинация содержится в четырех указанных транзакциях из шести, т.е. является частой. В отличие от этого комбинация АС не является частой, так как t(A)t(C) = {1345}{2456} = {45}, т.е. АС содержится только в двух транзакциях (30% поддержки). Поэтому эта комбинация исключается из процесса генерирования комбинаций на следующем уровне. Плотная база характеризуется заданным уровнем поддержки большинства элементов и минимальным числом нулевых значений (страхование, перепись населения, скоринг и другие клиентские базы). Для плотных баз объем исходной информации можно значительно уменьшить, если рассматривать разностные множества идентификаторов транзакций (diffsets), в которых данный элемент или комбинация отсутствует (рис.1,с). В нашем примере tidsets использует 18 ячеек памяти, а diffsets –только 7. При этом для плотных баз размер tidsets может стать очень большим, что делает целесообразным переход к diffsets. В теории более сложным выглядит получение разностных множеств идентификаторов транзакций. Будем обозначать разностные множества символом d. В нашем примере d(A) = {26}, d(B) = {}, d(C) = {13}, d(D) = {24}, d(E) = {6}, где в фигурных скобках перечислены индексы транзакций, не содержащих данный элемент. Разностные множества 2-го уровня типа d(PX), где элемент Р – префикс класса, а Х – присоединяемый элемент, лексикографический индекс которого выше индекса Р (по правилу Apriori), представляют результат конъюнкции Р и Х. Формула d(PX) представляет собой множество идентификаторов транзакций, в которых при наличии префикса Р не содержится элемент Х. Это правило позволяет вычислять множество d(PX) по разности множеств t(P) – t(X), т.е. путем вычеркивания из уменьшаемого t(P) общих индексов транзакций с вычитаемым t(X). Например, d(AВ) = t(A) – t(В) = {1345}-{123456} = {}. Пустое множество означает, что отсутствуют транзакции, в которых при наличии А нет В. В этом легко убедиться на рис.1,а. d(AC) = t(A) - t(C) = {1345}-{2456}={13}. Этот результат означает, что в транзакциях с индексами 1 и 3 при наличии А не содержится С. Для вычисления поддержки Ϭ(РХ), измеряемой числом транзакций, используется разностная формула:

 Ϭ(PX) = Ϭ(P) - |d(PX)| (1)

 где модуль |d(PX)| - это число элементов в разностном множестве d(PX). Логически формула (1) декларирует, что поддержка комбинации РХ меньше поддержки префикса Р на число транзакций, не содержащих Х при наличии Р. В частности, Ϭ(AB) = Ϭ(A) - |d(AB| = 4 - |{}| = 4 (транзакции). Во втором случае Ϭ(АС) = Ϭ(А) - |d(AC)| = 4 – 2 = 2. Эта комбинация не частая, и не должна использоваться при генерации кандидатов следующего уровня. В конечном итоге из возможных 10 комбинаций на 2-ом уровне частыми множествами оказалось только 8. Комбинационное дерево будет и дальше сжиматься, пока на пятом, последнем уровне не останется одна частая комбинация АВDE, поддерживаемая тремя транзакциями. Этот метод экономит память и время в сравнении с перебором потенциальных кандидатов алгоритмом Apriori.

На третьем уровне (k = 3) для получения тройных комбинаций элементов необходимо, как уже отмечалось, перебирать пары упорядоченных двойных комбинаций предыдущего уровня при условии одинакового префикса, состоящего в данном случае из одного элемента (k - 2 = 1) путем присоединения к данной двойной комбинации старшего элемента из второй двойной комбинации. Например, для получения тройной комбинации АBD надо к АВ присоединить путем конъюнкции BD, т.е., по сути, старший элемент D.

Теперь рассмотрим, как подсчитать поддержку тройных комбинаций PXY, имея d(PX) и d(PY). Для этого правило (1) можно использовать рекурсивно:

 Ϭ(PXY) = Ϭ(PX) - |d(PXY)| (2)

где новым префиксом является РХ. Его поддержка уже известна из (1), правда, при ее подсчете пока использовались множества tidsets, объем которых для плотных баз больше, чем diffsets, но этот недостаток чуть позже будет устранен. Но здесь неизвестно число транзакций в разностном множестве |d(PXY)|. Попытаемся найти это множество через двойные разностные множества d(PX) и d(PY). По определению

 d(PXY) = t(PX) – t(PY) (3)

Используем тождественное преобразование, добавляя и вычитая t(P), с перегруппировкой слагаемых получим:

 d(PXY) =(t(P) – t(PY)) – (t(P) – t(PX)) = d(PY) – d(PX) (4)

Обратите внимание на произошедший обмен местами PY и PX.

Из этой формулы рекурсивно следует, что

 d(PX) = d(X) – d(P) (5)

что позволяет выразить все двойные разности не через t, как было ранее, а через множества d. Например,

 d(AC) = d(C) – d(A) = {13} – {26} = {13},

что совпадает с вычислениями, приведенными выше через t, но экономит память.

Выразим добавление 3-го предмета Y к PX с помощью разностных формул (4) и (5). Например, d(ABD) = d(AD) – d(AB) =[(d(D) – d(A)] – [d(B)- d(A)] = [{24} – {26}] – [{26}- {}] = {4} – {26} = {4}. Этот результат означает, что в транзакции с индексом 4 при наличии АВ отсутствует D. Аналогично d(ABE) = {} (т.е. нет таких транзакций, где при наличии АВ отсутствует Е). d(ADE) = {} (нет транзакций, у которых при наличии AD отсутствует E), в чем можно убедиться на рис.1,а.

Определим рекурсивно по формуле (2) поддержку тройных комбинаций:

 Ϭ(ABD) = Ϭ(AB) - |d(ABD| = 4 – 1 = 3 Данная комбинация действительно содержится в трех транзакциях. Ϭ(ABE) = Ϭ(AB) - |d(ABE)| = 4 – 0 = 4. Эта комбинация содержится в 4-х транзакциях. Всего на этом третьем уровне частыми оказались пять комбинаций. Помимо двух приведенных это ADE, BCE, BDE.

Комбинирование на следующем четвертом уровне требует наличия двух тройных комбинаций с необходимой поддержкой и общим префиксом из двух элементов. Рекурсивно применяя (4) к комбинациям ABD и ABE с общим префиксом AB, получим d(ABDE) = d(ABE) – d(ABD) = {} – {4} = {}. В этом случае также можно убедиться, что нет ни одной транзакции, в которой при наличии элементов ABD отсутствовал бы элемент Е. Поддержку этой комбинации с префиксом ABD определим, рекурсивно применяя формулу (2):

 Ϭ(ABDE) = Ϭ(ABD) - |d(ABDE)| = 3 – 0 = 3 (транзакции).

Общий вид формулы для вычисления разностей:

 d(PXiXjY) = d(PXiY) – d(PXiXj) (6)

где PXi –общий префикс, Y – добавляемый элемент, старший по индексу.

Формула дает множество идентификаторов транзакций, в которых при наличии PXi….Xj отсутствует элемент Y. Общий вид формулы для вычисления поддержки комбинации, выражаемой числом транзакций:

 Ϭ(PXiXjY) = Ϭ(PXiXj) - |d(PXiXjY)| (7)

Прямоугольную таблицу diffsets можно создать путем выполнения SQL-запросов к таблице “товары”, содержащей поля идентификатора товара id и наименование, и к таблице “транзакции”, содержащей поля идентификатора транзакции tid и идентификатора товара. При этом в столбце tid итоговой таблицы будут перечисляться идентификаторы тех транзакций, в которых данный товар отсутствует. Такая таблица diffsets будет иметь вид:

|  |  |
| --- | --- |
|  id | tid |
| 10 | 1 |
| 10 | 3 |
| 20 | 2 |
| 20 | 3 |
| …. | ….. |

 Это позволит просто вычислять разности, например, d(10) = {13}, d(20) = {23}, или находить индексы транзакций, в которых не содержится данная комбинация. Например, в транзакции 3 не содержится комбинация товаров с индексами 10 и 20 и т.д. Однако видно, что индекс товара id будет повторяться столько раз, сколько существует транзакций, в которых данный товар отсутствует. Это приведет к огромному числу строк. Кроме того, для получения такой таблицы потребуется реализовать множество громоздких запросов к исходным таблицам. При этом число строк таблицы diffsets заранее неизвестно.

Задача конвертирования реляционных таблиц товаров и транзакций в вертикальную форму представления diffsets оказалась не простой и потребовала проведения специальных экспериментов. На первом этапе пытались представить множество diffsets с помощью объектов на языке Java путем выполнения SQL-запросов к таблице товаров и транзакций. В результате этих запросов к таблице товаров, число которых равно числу товаров в базе, создается двумерный вспомогательный массив Rez(1…N,1..2):

 Rez = Запрос (Вернуть значения полей наименования товара и его идентификатора из <название базы>.таблицы товаров). Далее создаются в цикле новые объекты класса KombEL и заполняются его свойства Title и Id значениями из соответствующих полей массива Rez. Эти объекты образуют массив MasKombEL, хранящий множество diffsets в атрибуте (списке) MasT. Этот атрибут заполняют в цикле по числу объектов путем выполнения следующих запросов к таблице транзакций:

MasKombEl[i].MasT = Запрос (Вернуть список идентификаторов транзакций, не содержащих i-ый товар из <имя базы>.таблицы транзакций). Здесь же можно подсчитать, путем запроса к таблице транзакций, их число, исключив с помощью специального оператора повторы идентификаторов. Далее определить размер (число идентификаторов) в списках MasT и по их разности найти поддержку каждого товара. К достоинствам данного метода можно отнести создание diffsets без изменения структуры БД. Кроме того, экономится память, так как считывается не вся база, а только diffsets множества. Однако у этого метода оказались серьезные недостатки, перекрывающие его достоинства: низкая скорость, связанная с громоздкими SQL-запросами и не масштабируемость алгоритма. Однако данный подход оказалось возможным усовершенствовать.

 Для этого предлагается считывать таблицу транзакций не целиком в оперативную память, а блоками строк, относящимися к одной транзакции. Каждая строка будет содержать поля идентификатора транзакции и идентификатора одного товара, принадлежащего этой транзакции. Таким путем будем собирать информацию обо всех товарах одной транзакции и записывать ее в объект OneTrans (одна транзакция) класса KombEL. Вместо массива Rez из запроса к таблице “товары”, создаем массив Database, формируемый запросом к таблице транзакций. Database является объектом класса ResultSet специального буфера потока в Java, в который данные поступают по одной строке в результате SQL-запроса к таблице транзакций.

 Database = Запрос (Вернуть TID, ELEMENT из таблицы <имя базы>.транзакций).

//Далее организуется цикл по k элементам Database для считывания блока строк.

 Цикл по k элементам Database

 Если OneTrans не равно 0, то{

 Если считанная строка из буфера потока Database[k,1] принадлежит той же

 строке k, что и объект OneTrans.Title, то {

 добавить считанный элемент ELEMENT строки к списку MasT объекта

 OneTrans //В объекте OneTrans[k].MasT собираются элементы транзакции.

 Иначе //кончились строки, принадлежащие данной транзакции

 Цикл по индексам товаров id этой транзакции {

 Если объект OneTrans[k].MasT не содержит Id данного товара, то добавить в

 выходной массив VxodData.MasT класса KombEL идентификатор этой

 транзакции из объекта OneTrans.Title. //это будет diffsets.

 кесли

 Кцикл

 Создать новый объект OneTrans[k+1]

 Иначе создать новый объект OneTrans //Этот случай предусмотрен для 1-ой строки,

 если OneTrans равен 0.

Кцикл

Таким образом, мы никогда не считываем всю таблицу транзакций в оперативную память, а делаем это блоками для каждой транзакции, заполняя атрибуты объекта OneTrans, в одном из которых (MasT) содержится список идентификаторов товаров данной транзакции. Последующий цикл по товарам позволяет вычислить те из них, которые не содержатся в данной транзакции. Выходной массив таких строк VxodData[k].MasT будет логически представлять собой вертикальное множество diffsets. Однако этот массив не содержит сведений о поддержке элементов и о часто встречающихся товарах. Для начала нужно задать в % минимально допустимую поддержку элементов (KoefSup). Тогда допустимое число транзакций, в которых должен содержаться элемент (MinSup) можно найти по формуле:

 MinSup = округлить (КТ\*KoefSup/100)

Отобранные таким путем товары будут первыми частыми множествами, которые перепишем в динамический массив Result класса KombEL2, который будет отличаться по структуре от класса KombEL добавленным атрибутом S для хранения поддержки каждого товара.

Цикл по элементам i массива VxodData

 Sup = KT – (число элементов в объекте VxodData[i].MasT); //КТ – число транзакций

 Если Sup >= MinSup, то

 Добавить в выходной массив Result[i] объект VxodData[i];

 Добавить в выходной массив Result[i].S = Sup

 кесли

 Кцикл

VxodData = 0 //Удаляем уже не нужный массив. Вся нужная информация - в Result.

Если (Result.size) > 1, то вызываем процедуру комбинирования товаров Perebor. Если отобрано более одного частого товара, начинаем поиск и определение поддержки двойных и более высокого порядка частых множеств.

Комбинирование реализуется как в алгоритме Apriori рекурсивно. К исходной комбинации добавляется старший элемент Y из второй комбинации предыдущего уровня путем конъюнкции и определяется разностное множество по формуле (6). Рассмотрим алгоритм вычисления разностных множеств. Его реализует вспомогательная функция dRaznost. На ее вход подаются два объекта класса KombEL2: расширенный объект В и исходный расширяемый объект А.

 dRaznost (объект В, объект А) //В обратном порядке в соответствии с (6)

 dRaznost = 0

 Если количество\_элементов (В.MasT) = 0, то

 Вернуть новый пустой список (выход)

 Иначе

 Если (не\_содержится элемент массива (A.MasT) в массиве (B.MasT)), то

 Добавить в выходной массив разностей элемент (B.MasT)

 кесли

 Кцикл

 кесли

 Второй вспомогательной функцией является SupComb для расчета поддержки новой комбинации в соответствии с формулой (7).

 SupComb (расширяемый объект А класса KombEL2, Y - число элементов разностного множества в расширенном объекте В того же класса); S - поддержка

 SupComb = A.S – Y;

В заключении рассмотрим упрощенный алгоритм функции Perebor, создающей новые комбинации, вычисляющей их разностные множества и поддержку. Вызов этой функции осуществляем с массивом частых элементов Result.

 Perebor (P – динамический массив Result объектов класса KombEL2)

 Цикл по количеству элементов Р.

 //Создаем новый массив Р1 для комбинаций следующего уровня;

 P1 = new ArrayList <Komb>( );

 Цикл от j = i + 1 до (количество элементов в (Р)) {

 //создаем новую комбинацию с префиксом P[i]

 BufTitle = P[i].Title + “&” + P[j].Title;

 //Вычисляем разностное множество этой комбинации:

 BufMas = dRaznost (P[j], P[i]);

 //Вычисляем поддержку этой комбинации:

 BufSup = SupComb (P[i], число элементов множества BufMas);

 Если (BufSup) >= MinSup, то {

 //Добавляем в Р1 новую комбинацию

 P1.add(new Komb(BufTiTle, BufMas)

 //Добавляем в результат Result ссылку на новую комбинацию и ее поддержку

 Result.add (P1.get(P1.size( ) – 1);

 Result.get(Result.size( ) – 1).s = BufSup

 }

 }

 //Если размер массива Р1 больше единицы, то рекурсивно вызываем функцию Perebor с новым массивом:

 Perebor (P1).

Использование операции конъюнкции при создании новой комбинации позволяет присоединять к исходной комбинации с общим префиксом только старший элемент присоединяемой комбинации, что соответствует рассмотренной теории.

**Реализация и тестирование. (**В работе принимал участие Кызылов А.В.)

Для реализации алгоритма разработана программа на языке Java с графическим интерфейсом. Для запуска поиска частых множеств нужно ввести в интерфейс: 1) название ODBC драйвера с созданным подключением; 2) название базы поиска; 3) коэффициент минимальной поддержки в процентах. Существующие наборы тестовых баз являются файлами (“Голосование в Конгрессе США”, “Грибы” и пр.) не пригодны для решения поставленной задачи. Поэтому необходимо было разработать программный генератор тестовых баз. Мы воспользовались готовой программой такого генератора “GeneratorDB” из дипломной работы Анваера А.Е. “Разработка алгоритма извлечения ассоциативных правил из множества категориальных данных” за 2008 год, внеся в нее небольшие изменения. Однако вначале необходимо было убедиться в корректной работе основного разработанного алгоритма. Для этого использован один из инструментов аналитической платформы Deductor Studio 4.4 [5]. Этот инструмент предназначен для поиска ассоциативных правил в данных. Его основу составляет алгоритм Apriori поиска частых множеств, но он может быть настроен на выдачу “Популярных наборов”, т.е. частых множеств с заданными уровнями минимальной и максимальной поддержки. При корректной работе нашего приложения искомые частые множества должны совпадать с множествами, выдаваемыми Deductor.

Приложение “GeneratorDB” позволяет варьировать количество исходных элементов и количество транзакций (максимально генерировали 100000 транзакций). Можно также изменять минимальное и максимальное количество элементов в транзакции. От этого параметра зависит плотность базы. При увеличении обоих параметров плотность возрастает. Кроме того, можно добавлять, удалять и редактировать элементы. В приложении заложены три шаблона исходных элементов (товаров). По умолчанию основной шаблон состоит из 26 символов английского алфавита. Другой шаблон включает в себя 1000 элементов с названиями от e1 до e1000. Третий шаблон состоит из 20 названий реальных продуктов. Создавая генератором реляционная база содержит две таблицы: “товары” и “транзакции”, что соответствует рассмотренной технологии последующего преобразования таблиц в объектное множество diffsets. Сгенерированные транзакции отображаются в форме таблицы из двух столбцов. Первый столбец содержит идентификаторы транзакций, второй – содержащийся в ней предмет (товар). Если транзакция содержит множество разных предметов, то ее идентификатор повторяется в соответствующих строках первого столбца.

 Таким образом, в отличие от исходной таблицы транзакций вместо идентификатора товара в поле Product указано его наименование. Транзакция с идентификатором Tid =1 состоит из одного элемента “w”. Транзакция с Tid = 2 состоит из 7 элементов: n,i,g,c,m,b,o. При нажатии на кнопку “Скомпоновать” все элементы сгруппируются по транзакциям (рис.2).

**

 Рис.2. Представление таблицы транзакций после компоновки.

Для тестирования сгенерировали малую тестовую базу SmallD из 1000 транзакций и 26 элементов. При минимальной поддержке 36% приложением diffsets найдено 48 частых множеств.

На той же тестовой базе с использованием приложения Deductor при минимальной поддержке 36% получены те же 48 частых множеств (рис.3). Это подтверждает корректность работы нашего приложения.



Рис.3. Часто встречающиеся множества в тестовой базе SmallD в результате работы приложения Deductor.

К сожалению, генератор случайных транзакций реализует только равномерное распределение элементов, поэтому поддержка всех одноэлементных множеств оказалась примерно 50%, а двухэлементных – чуть выше 36%.

В работе также проведено сравнительное тестирование с самым быстрым на сегодня алгоритмом нахождения частых множеств FP-growth[4], реализованным в работе Анваера А.Е. Однако приложение “FP-growth” работает с базами данных в формате MS Excel. Поэтому были заменены процедуры, отвечающие за чтение базы из файла на процедуры чтения данных через ODBC-драйвер.

Для тестирования было создано шесть комплектов баз разного объема: 3000, 11000 и 100000 транзакций с двумя уровнями плотности каждая. Плотность регулировалась изменением соотношения минимального и максимального количества элементов в транзакции от 1:15 для неплотных баз до 500:1000 для плотных баз. В результате экспериментов оказалось, что для поиска в неплотных базах небольшого объема diffsets затрачивает примерно в два раза меньше времени, чем FP-growth, независимо от величины поддержки в пределах 15 – 40 %. При увеличении объема и плотности баз алгоритм FP-growth обгоняет алгритм diffsets примерно в три раза за счет меньшего времени работы алгоритма, но при практически равном времени составления разностных множеств и времени чтения базы этими алгоритмами. Возможно, это связано с использованием алгоритмом FP-growth деревьев (JavaTree), по сравнению с массивами, используемыми в diffsets, работающими медленнее деревьев. Среднее время работы алгоритма diffsets для больших плотных баз составило около 50 минут, для алгоритма FP-growth – около 17 минут.

**Выводы.** Разработан, реализован и протестирован объектно-ориентированный алгоритм diffsets для нахождения частых множеств в транзакционных данных вертикального формата, который по скорости мало уступает самому быстрому на сегодня алгоритму FP-growth. Некоторое отставание на больших плотных базах связано, по-видимому, с использованием в случае FP-growth деревьев, работающих быстрее массивов в объектах diffsets, без использования которых, однако, невозможно получение разностных множеств вертикального формата.

Литература.

1. В.Ганти, И.Герке, Р.Рамакришан. Добыча данных в сверхбольших базах данных. – “Открытые системы”, № 9 -10, 1999.
2. R.Agrawal, T.Tmielinski, A.Swami. Mining association between sets of items in massive databases. – In Proc. of the 1993 ACM-SIGMOD Int’l Conference on management of data.
3. R.Agrawal, HManilla, R.Srikant. Fast discovering of association rules. – In Proc.of 20th Int’l Conference on VLDB, Santiago, Chile, 1994.
4. C.Borgelt. An implementation of the FP-growth algorithm. – Dep. Of knowledge processing and language engineering school of computer science. Otto-von- Guericke-University, Magdeburg, 2003.
5. M.Zaki and K.Gouda. Fast vertical mining using Diffsets. – Computer Science Dept.,Rensselaer Institute, Troy, NY, 12180 USA, 2001.
6. P.Shenoy, J.Harista, S.Sudarsham. Turbo-charging vertical mining of large databases. – In ACM SIGMOD Int’l Conf. Management of data, 2000.
7. Cайт компании BaseGroup Labs. [www.basegroup.ru](http://www.basegroup.ru). Deductor Studio.

ABSTRACT

**Frequent sets mining from transaction data of vertical format**

Frequent sets mining from very large databases underlies fast discovering of association rules, web-technologies, classification and other management of data problems. The article is describes an implementation of proposed frequent sets mining algorithm from transaction data by means converting their in vertical format and finding difference identifier sets (diffsets) in which specified element is absent. The method uses object-oriented approach. The comparative survey proved that velocity data converting in vertical difference format approximately the same as velocity reading data of fastest algorithm FP-growth.