Поиск максимальных последовательностей элементных множеств с помощью битовых матриц*

А. А. Ахрем, А. В. Янкелевич

В статье рассматривается проблема поиска максимальных последовательностей элементных множеств в хранилищах данных. Алгоритм использует битовые матрицы для хранения текущего состояния шага поиска, а также для выведения следующего поколения последовательностей на основе имеющейся информации. Целью применения данной схемы является ускорение поиска за счет минимизации обращения к хранилищу данных и использования простых структур для хранения данных в оперативной памяти.

1. Введение

За последнее десятилетие системы хранения информации вошли во все области человеческой деятельности. До недавнего времени процесс использования систем хранения можно было охарактеризовать термином «аккумулирование». Информация собиралась, вносилась в информационную систему, над ней выполнялись простые поисковые запросы. В настоящее время наступает новый этап. Растущие потребности в комплексном анализе информации заставляют использовать накопленные данные не только для простых выборок, но и для выявления схем взаимодействия, влияния друг на друга информационных потоков, классификации, прогнозирования событий.

В качестве примера областей, в которых крайне востребованы системы анализа данных, можно привести следующие:

- обработка экспериментальных научных данных, выявление связей и закономерностей;
- обработка журналов систем безопасности, выявление попыток взлома;
- анализ потребительской корзины, выявление направлений покупательской активности, составление маркетинговых планов.

* Работа выполнена при поддержке РФФИ (проект № 04–01–00386, 07–01–00572) и программы Президиума РАН «Фундаментальные проблемы информатики и информационных технологий» (проект № 2.44)

Таблица 1

Номер эксперимента	Момент времени	Наблюдаемые показатели
1	1	{a, b, d}
1	2	{b, c, d}
1	3	{b, c, d}
2	4	{b}
2	5	{a, b, c}
3	6	{a, b}
3	7	{b, c, d}

В основе решения всех вышеуказанных задач лежит единый подход — выявление набора наиболее часто встречающихся последовательностей данных. Опираясь на полученные последовательности можно строить ассоциативные правила, позволяющие выполнять прогнозирование в потоках данных.

В качестве иллюстрации рассмотрим пример. Представим базу данных, в которой хранятся экспериментальные данные. Каждая запись характеризует набор наблюдаемых показателей в определенный момент времени. Пример подобной базы данных приведен в табл. 1.

Как видно из табл. 1 подмножества наблюдаемых показателей пересекаются в экспериментах с различными номерами. Также нетрудно заметить, что в экспериментах 1 и 2 подмножества {a, b} и {b, c, d} появляются в одинаковом хронологическом порядке. Такой упорядоченный набор подмножеств называется последовательностью. Поскольку последовательность встречается неоднократно, то данные сведения могут представлять определенный интерес с точки зрения выявления определенных закономерностей в исследуемой области. Как указывалось ранее, одним из способов применения последовательностей является построение ассоциативных правил, например, $\{a, b\} \rightarrow \{b, c, d\}$. В данном случае мы можем сказать, что при появлении показателей {a, b}, мы вправе ожидать, с определенной вероятностью, появления показателей {b, c, d} в одном из следующих наблюдений. Вероятность срабатывания правила определяется как отношение количества экспериментов, в которых присутствует вся последовательность, к количеству экспериментов, в которых присутствует левая часть правила (в данном случае, {a, b}).

Безусловно, практическую ценность представляют не все возможные последовательности, а только отвечающие определенным условиям.

В данной статье рассматривается поиск максимальных последовательностей. Формальное определение этого вида дано в разделе 2.

Впервые задача поиска максимальных последовательностей данных была предложена исследователями лаборатории IBM Research [1, 2]. В своих работах они представили алгоритмы Apriori, SPADE, использующие древовидную структуру для вывода всевозможных максимальных последовательностей. К основным недостаткам, отличающим указанные алгоритмы относятся:

- использование сложных структур представления данных;
- большое количество обращение к базе данных;
- сложность адаптации алгоритма к работе в «скользящем окне» над потоком данных.

В данной работе предлагается иной подход к решению задачи. В настоящее время для вычислительных систем промышленного уровня объем оперативной памяти в несколько гигабайт не является чем-то необычным, поэтому предлагаемый алгоритм основывается на том, что структура, необходимая для вычисления максимальных последовательностей, постоянно может храниться в оперативной памяти. Таким образом, к базе данных выполняется единственное обращение в момент инициализации алгоритма. Вторым существенным моментом является использование не древовидной структуры, а битовой матрицы и, как следствие, возможность использовать быстрые битовые операции в вычислениях.

В следующем разделе представлено формальное описание задачи, а так же изложен алгоритм.

2. Описание алгоритма

Вновь обратимся к базе данных D с экспериментальными данными. Данные по эксперименту снимаются в определенные моменты времени. Получаемый срез данных будем называть транзакцией T. Транзакция состоит из набора наименований показателей и в базе данных идентифицируется уникальным упорядоченным номером транзакции TID и уникальным номером эксперимента CID. Формально вводные данные к задаче представлены в табл. 2.

Обозначим X некоторое подмножество элементов транзакции T, обозначающее, что в определенный момент времени мы наблюдаем группу показателей. Обозначим через $S = \{X_1, X_2, \ldots, X_k\}$ упорядоченную последовательность подмножеств X^S, где каждое отдельное подмножество элементов X^S, принадлежит уникальной транзакции, транзакции принадлежат одному эксперименту, а условие упорядоченности говорит о том, что если один элемент X^S в описании последовательности стоит

Таблица 2

Используемые обозначения

Условное обозначение	Описание
$L = \{i_1, i_2, \ldots, i_n\}$	L — множество наблюдаемых показателей эксперимента i — элемент множества L (определенный показатель)
T	Транзакция — подмножество множества L
D	База данных, состоящая из транзакций T
TID	Уникальный упорядоченный идентификатор транзакции. Требование упорядоченности подразумевает, что транзакции с большим номером появились в базе данных позже транзакций с меньшим номером.
CID	Уникальный идентификатор эксперимента

левее другого, то соответствующая ему транзакция появилась в D раньше транзакции, соответствующей второму элементу. Длину множества $|X|$ будем вычислять как количество элементов i, его образующих. Длину последовательности S будем вычислять как

$$|S| = \sum_{i=1}^{k} |X_i|, \tag{1}$$

где k — количество подмножеств X, образующих последовательность. Последовательность длины l будем называть l-последовательностью.

Введем термин «поддержка» $Sup(S)$ последовательности S в D, как количество экспериментов, в которых наблюдались группы показателей в последовательности S. Частота последовательности S определяется как

$$F(S) = \frac{Sup(S)}{|\mathrm{D}|}, \tag{2}$$

где $|\mathrm{D}|$ — количество уникальных идентификаторов CID в D.

Обозначим $MinSup$ минимальную интересную для нас поддержку последовательности S в D. Будем называть последовательность S максимальной, если из данной последовательности невозможно получить последовательность $SExt$ длины, имеющей $Sup(SExt) \geqslant MinSup$. Следует отметить, $SExt$ можно получить из S двумя способами, за счет добавления к одному из множеств X^S нового элемента i, либо за счет добавления к последовательности нового подмножества X.

Таким образом, мы подошли к формулировке задачи: «В базе данных D найти все максимальные последовательности S, для которых $Sup(S) \geqslant MinSup$».

Прежде чем перейти к конструированию алгоритма, необходимо представить некоторые положения, используемые далее.

Рассмотрим последовательности

$$S_1 = \{X_1, X_2, \ldots, X_m\} \quad \text{и} \quad S_2 = \{Y_1, Y_2, \ldots, Y_n\}.$$

Будем считать, что S_1 является подпоследовательностью S_2, в том случае, если существуют такие числа [1] $1 \leqslant l1 < l2 < \ldots < lm \leqslant n$, что $X_1 \subseteq Y_{l1}$, $X_2 \subseteq Y_{l2}$, $\ldots$, $X_m \subseteq Y_{lm}$.

Лемма 1. *Если S_1 является подпоследовательностью S_2, то*

$$Sup(S_1) \geqslant Sup(S_2), \quad \text{см. [1].}$$

Возвращаясь к задаче поиска максимальных последовательностей, следует отметить, что в общем случае нам предстоит перебрать все возможные последовательности S и для каждой проверить условие $Sup(S) \geqslant MinSup$. Алгоритм подобного перебора обладает неопределенной сложностью, стремящейся к бесконечности.

Опираясь на Лемму 1, можно построить итерационный алгоритм конструирования последовательностей. При этом на каждом шаге мы сможем оценивать полезность последовательности в качестве строительного материала для следующего шага и, в случае необходимости, отсекать бесполезные.

Представим D в виде табл. 3. Таблица упорядочена по возрастанию по столбцам CID и TID.

Алгоритм начинается с определения всех *последовательностей длины* 1, для которых $Sup(S) \geqslant MinSup$. Все последовательности, не обладающие необходимой *поддержкой*, отсекаются. Обозначим полученное множество всех $S(1)$, обладающих необходимой поддержкой, $H(1)$. Далее попарно комбинируя элементы $H(1)$, сгенерируем множество кандидатов *последовательностей длины* 2. Алгоритм генерации последовательностей-кандидатов длины 2 представлен ниже:

Из 2-х элементов $S_1(1)$ и $S_2(1)$ можно образовать последовательность:
1. $\{S_1(1)S_2(1)\}$ — назовем это ***I-шаг***;
2. $\{S_1(1),\ S_2(1)\}$ — назовем это ***S-шаг***.

[1] Следует отметить, что номер набора элементов в последовательности соответствует номеру транзакции

Таблица 3

Представление D

Номер эксперимента CID	Номер транзакции TID	Состав транзакции T_CID, TID
1	1	$T_{1,1}$
1	...	...
1	Количество транзакций TCOUNT для CID = 1	T_1, TCOUNT(CID = 1)
...	...	...
N	1	$T_{N,1}$
N	...	...
N	Количество транзакций TCOUNT для CID = N	T_1, TCOUNT(CID = N)

Обозначим $H(2)$ все сгенерированные *последовательности длины* 2, обладающие требуемой поддержкой.

Уже начиная с шага генерации *последовательностей длины* 3, возникает важный вопрос, какие последовательности имеет смысл комбинировать между собой. Предлагается следующий алгоритм генерации последовательностей-кандидатов длиной, превышающей 2:

Рассмотрим 2 *последовательности* $S_1 = \{X_1, X_2, \ldots, X_m\}$ *и* $S_2 = \{Y_1, Y_2, \ldots, Y_n\}$. *Возможны* 2 *варианта объединения*:

1. ***S-шаг:*** *Если* Y_1 *является подмножеством* X_m *и* S_1 *состоит более чем из одного элементного множества, то*

$$S_{\text{кандидат}} = \{X_1, X_2, \ldots, X_m, Y_2, \ldots, Y_n\};$$

2. ***I-шаг:***

   а) Если $m = n$ *и* S_1 *и* S_2 *различаются только одним элементом в множествах* X_m *и* Y_n, *то*

$$S_{\text{кандидат}} = \{X_1, X_2, \ldots, X_m \cup Y_n / (X_m \cap Y_n)\};$$

   б) Если Y_1 *является подмножеством* X_m *и* S_1 *состоит из одно элементного множества, то*

$$S_{\text{кандидат}} = \{X_1, X_2, \ldots, X_m, Y_2, \ldots, Y_n\}.$$

После генерации кандидата S необходимо убедиться, что

$$Sup(S) \geqslant MinSup.$$

Если на определенном шаге последовательность S не участвует в генерации кандидатов следующего шага, то она добавляется в результирующий набор максимальных последовательностей.

В процессе генерации кандидатов разные пары последовательностей могут давать одинаковый результат. Каждый новый кандидат должен проверяться на уникальность по существующему набору кандидатов. Кандидаты-дубликаты должны исключаться.

Алгоритм заканчивает работу в момент, когда на определенном шаге J невозможно сгенерировать кандидатов, обладающих требуемой поддержкой.

Все последовательности с шага $J - 1$, участвующие в генерации кандидатов шага J добавляются в результирующий набор.

Поддержку кандидата можно определить, выполнив запрос (серию запросов) к базе данных, однако, это весьма неэффективно с точки зрения количества операций ввода/вывода и использования процессора на обсчет запросов. В данной статье предлагается простой алгоритм ведения перечня кандидатов и вычисления их поддержки в оперативной памяти. Для этого воспользуемся битовой матрицей.

Следует отметить, что все используемые далее матрицы, отсортированы по возрастанию по колонкам CID, TID.

На первом шаге сформируем матрицу принадлежности 1-*последовательности* $S(1)$ для каждой записи {CID, TID} в базе данных, в виде табл. 4.

Таблица 4

Матрица принадлежности *1-последовательностей*

CID	TID	Бит принадлежности
<CID>	<TID>	1 — если $S(1)$ принадлежит транзакции 0 — в противном случае

На втором и последующем шагах сгенерируем матрицы для последовательностей-кандидатов, полученных с помощью S- и I-шагов. Матрица и правила вычисления битов представлены в табл. 5.

В процессе формирования матрицы для кандидатов достаточно легко определить их поддержку. Для этого достаточно подсчитать количество CID, для которых в столбце кандидата есть хотя бы одна единица. Кандидаты, не обладающие требуемой поддержкой, исключаются из перечня

Таблица 5

Битовая матрица кандидата на шаге J

CID	TID	$S_1(J-1)$	$S_2(J-1)$	Кандидат на шаге J	
				I-шаг	S-шаг
<CID>	<TID>	b_1	b_2	$b_1 \cap b_2$	если для одного и того же CID, существует $\mathrm{TID}_{S_2} > \mathrm{TID}_{S_1}$ такой, что $b_2(\mathrm{TID}_{S_2}) = 1$ и $b_1(\mathrm{TID}_{S1}) = 1$ 0 — в противном случае

участвующих в генерации следующего шага. Также, если после генерации шага J в матрице, состоящей из столбцов всех кандидатов, появились строки $\{\mathrm{CID}, \mathrm{TID}\}$, содержащие только нули, то их можно исключить, так как они больше не несут полезной информации для $J + 1$ шага.

3. Пример использования алгоритма

Рассмотрим работу алгоритма на примере из табл. 1. Установим

$$MinSup = 2.$$

1) Сформируем битовую матрицу для первого шага алгоритма:

CID	TID	a	b	c	d
1	1	1	1	0	1
1	2	0	1	1	1
1	3	0	1	1	1
2	4	0	1	0	0
2	5	1	1	1	0
3	6	1	1	0	0
3	7	0	1	1	1

2) На втором шаге построим матрицу для всех 2-последовательностей. Приведем матрицу для комбинаций, начинающихся с элемента “a”:

		I-шаг			*S*-шаг			
0 CID	TID	ab	ac	ad	a → a	a → b	a → c	a → d
1	1	1	0	1	0	1	1	1
1	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0
2	5	1	1	0	0	0	0	0
3	6	1	0	0	0	1	1	1
3	7	0	0	0	0	0	0	0

Для вычисленного блока можно выполнить отсечение ненужных столбцов. Все столбцы, имеющие поддержку меньше 2, не представляют интереса. К числу таких столбцов относятся: ac, ad, a → a, a → d. Приведем результирующую матрицу для 2-последовательностей:

CID	TID	ab	a → b	a → c	a → d	bc	bd	b → b	b → c	b → d	cd
1	1	1	1	1	1	0	1	1	1	1	0
1	2	0	0	0	0	1	1	1	1	1	1
1	3	0	0	0	0	1	1	0	0	0	1
2	4	0	0	0	0	0	0	1	1	0	0
2	5	1	0	0	0	1	0	0	0	0	0
3	6	1	1	1	1	0	0	1	1	1	0
3	7	0	0	0	0	1	1	0	0	0	1

3) На третьем шаге сформируем матрицу 3-последовательностей:

CID	TID	ab → b	ab → c	ab → d	bcd	b → bc	b → bd	b → cd
1	1	1	1	1	0	1	1	1
1	2	0	0	0	1	1	1	1
1	3	0	0	0	1	0	0	0
2	4	0	0	0	0	1	0	0
2	5	0	0	0	0	0	0	0
3	6	1	1	1	0	1	1	1
3	7	0	0	0	1	0	0	0

Данная матрица требует дополнительных комментариев относительно применения S- и I-шагов.

Последовательность b → b → c можно получить за счет S-шага, объединяющего b → b и b → c. Однако $Sup(\text{b} \rightarrow \text{b} \rightarrow \text{c}) = 1$, поэтому данная последовательность нами исключается.

Последовательность bcd получается за счет выполнения I-шага, объединяющего bc и bd (указанные последовательности различаются только одним элементом). $Sup(\text{bcd}) = 2$, поэтому данная последовательность будет принимать участие в дальнейшей генерации кандидатов.

4) Четвертый шаг:

CID	TID	ab → bc	ab → bd	ab → cd	b → bcd
1	1	1	1	1	1
1	2	0	0	0	1
1	3	0	0	0	0
2	4	0	0	0	0
2	5	0	0	0	0
3	6	1	1	1	1
3	7	0	0	0	0

5) Пятый шаг:

CID	TID	ab → bcd
1	1	1
1	2	0
1	3	0
2	4	0
2	5	0
3	6	1
3	7	0

В результате, получили последовательность ab → bcd с поддержкой 2.

4. Экспериментальные данные

Экспериментальная проверка алгоритма подтвердила теоретические расчетные характеристики:

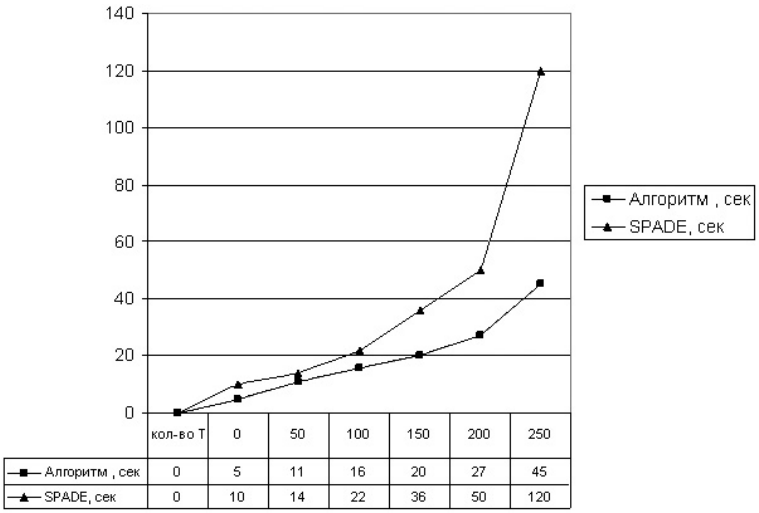
- на больших объемах данных алгоритм превосходит SPADE в силу того, что при увеличении количества транзакций решающую роль в общей производительности играет компонент вычисления кандидатов очередного шага;
- побочным эффектом использования битовых матриц является большая, нежели у SPADE, потребность в памяти.

На рис. 1 представлено сравнение данного алгоритма (на рисунке обозначается названием «Алгоритм») и SPADE. Ось X отражает количество транзакций в базе данных, ось Y время выполнения в секундах.

Эксперимент проводился на машине с процессором Athlon 1,2 ГГц и 512 Мб оперативной памяти.

5. Заключение

В данной статье представлена модификация алгоритма поиска максимальных последовательностей в базе данных с использованием битовых матриц. Предложенные подходы позволяют существенно снизить количество операций ввода/вывода, избежать представления данных



кол-во T	0	50	100	150	200	250
Алгоритм , сек	0	5	11	16	20	27	45
SPADE, сек	0	10	14	22	36	50	120

Рис. 1

в виде сложных иерархических структур и использовать высокоэффективные битовые операции в итерационной процедуре генерации и отбора последовательностей-кандидатов.

6. Дальнейшие исследования

Основными направлениями дальнейшего исследования являются следующие:

- разработка метода компактного хранения битовых матриц, без существенной потери вычислительных преимуществ;
- экстраполяция метода использования битовых матриц на задачу поиска максимальных последовательностей в потоке данных (вычисление в «скользящем окне»).

Литература

1. *Agrawal R. and Srikant R.* Mining Sequential Patterns // In ICDE 1995, Taipei, Taiwan. March 1995.
2. *Zaki M. J.* Spade: An efficient algorithm for mining frequent sequences. Machine Learning, 42(1/2):31–60, 2001.
3. *Srikant R. and Agrawal R.* Mining Sequential Patterns: Generalizations and Performance Improvements // In EDBT 1996, Avignon, France. March 1996.

4. *Mannila H.*, *Toivonen H. and Verkamo A. I.* Discovering frequent episodes in sequences // In KDD 1995. Montreal, Quebec, Canada. 1995. P. 210–215.
5. *Pei J.*, *Han J.*, *Mortazavi-Asl B.*, *Pinto H.*, *Chen Q.*, *Dayal U. and Hsu M.-C.* PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth // In ICDE 2001. Heidelberg, Germany. Apr. 2001. P. 215–226.
6. *Garofalakis M.*, *Rastogi R. and Shim K.* SPIRIT: Sequential pattern mining with regular expression constraints // In VLDB 1999. San Francisco, Morgan Kaufmann. Sept. 1999. P. 223–234.