СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УДК 004.9

ОПТИМИЗАЦИЯ ВХОДНЫХ ДАННЫХ В ЗАДАЧЕ ПОИСКА ШАБЛОНОВ И АССОЦИАТИВНЫХ ПРАВИЛ

Д. О. Кириченко, М. А. Артемов

Воронежский государственный университет

Поступила в редакцию 05.11.2014 г.

Аннотация. В статье дается краткий обзор существующих и наиболее распространенных алгоритмов поиска шаблонов и ассоциативных правил, рассматривается проблема высокого потребления памяти и излишних накладных расходов на вычисление при их использовании. В рамках статьи для решения обозначенной проблемы предлагается методика оптимизации входных данных, включающую многоступенчатую фильтрацию и разбиение данных на связанные группы с последующим выбором эффективного алгоритма для вычисления на каждой из групп в отдельности.
Ключевые слова: анализ данных, поиск шаблонов, поиск ассоциативных правил, оптимизация.
Annotation. The article contains short review of existing and most popular pattern mining and associative rule mining algorithms. It considers high using memory and overheads on calculation. Author offers optimization technique including several steps of filtration, data division on linked groups and choice of effective algorithm to perform separately on each group.
Keywords: data mining, pattern mining, associative rules mining, optimization.

ВВЕДЕНИЕ

Одной из задач анализа данных является задача поиска шаблонов и различных закономерностей в них. Она возникает в различных областях человеческой деятельности: выявления предпочтений покупателей, анализ запросов пользователей в системе полнотекстового поиска [9, 10], исследование ошибок в узкоспециализированных системах [11] и т. д. На сегодняшний день существует ряд алгоритмов, направленных на решение вышеописанной задачи. Однако при росте объема входных данных резко повышается время вычисления и объем потребляемой памяти. В данной статье приводится краткий обзор основных алгоритмов и предлагается методика фильтрации и преобразования входных данных при их использовании.

АЛГОРИТМЫ ПОИСКА АССОЦИАТИВНЫХ ПРАВИЛ И ШАБЛОНОВ. ОБЩИЕ СВЕДЕНИЯ

Ассоциативным правилом называется выражение вида $A \Rightarrow B$, при этом множества A и B такие, что $A \cap B = \varnothing$. Для автоматического вычисления таких правил существуют специальные алгоритмы, которые по входному набору множеств строят набор ассоциативных правил, удовлетворяющих тем или иным ограничениям.

Входными данными для таких алгоритмов является набор множеств. Этот набор будем называть «базовым набором», а каждое множество внутри него – транзакцией. Изначально алгоритмы поиска ассоциативных правил использовались для решения задачи потребительской корзины, поэтому значительная часть терминологии пришла из этой области.

Пусть $\sigma(A)$ – количество транзакций в базовом наборе, в которых присутствуют все элементы множества A, а N – общее число транзакций в базовом наборе.

Критериями оценки качественного ассоциативного правила выступают такие величины как: как поддержка и достоверность.

Поддержка – отношение количества транзакций, содержащих как условие, так и следствие к числу транзакций в базовом наборе.

$$\sup(B) = \frac{\sigma(A \cup B)}{N}. \quad (1)$$

Достоверность – отношение числа транзакций, содержащих как условие, так и следствие, к числу транзакций, содержащих только условие.

$$conf(B) = \frac{\sigma(A \cup B)}{\sigma(A)}. \quad (2)$$

Алгоритмы поиска ассоциативных правил близки по своей сути к алгоритмам поиска шаблонов, так, если найдено правило (1), то можно сказать, что набор $A \cup B$ является популярным вхождением (шаблоном). Зачастую алгоритмы поиска ассоциативных правил работают в два этапа:

- Построение всех популярных наборов
- Извлечение из полученных наборов ассоциативных правил, удовлетворяющих заранее заданным условиям достоверности.

В статье будут рассматриваться оптимизации только для этапа построения популярных наборов.

Ниже рассматриваются несколько наиболее распространенных алгоритмов. Все они требуют в числе своих параметров помимо базового набора, минимальный уровень поддержки, обозначенный как T.

APRIORI

Наиболее распространенным алгоритмом поиска ассоциативных правил является алгоритм apriori [5]. Он базируется на простой идее:

$$\forall a_n : \sup(\{a_1, a_2..a_{n-1}\}) \geq \sup(\{a_1, a_2..a_n\}). \quad (3)$$

Откуда следует, что

$$\forall a_n : \sup(\{a_1, a_2..a_{n-1}\}) < T \Rightarrow \Rightarrow \sup(\{a_1, a_2..a_n\}) < T \quad (4)$$

$$\sup(\{a_1, a_2..a_n\}) \geq T \Rightarrow \forall i \in 1..n : \sup(\{a_1, a_2..a_n\} / \{a_i\}) \geq T \quad (5)$$

Для этого алгоритма характерно заранее заданное требуемое число элементов в найденных шаблонах. В рамках статьи это число для всех алгоритмов будет обозначаться как M.

Общая схема работы:

Шаг 1. Фильтрация всех a_i и оставление только таких элементов, для которых $\sup(\{a_i\}) \geq T$. Инициализация $C_i^1 = \{a_{i,1}\}$, для отфильтрованных элементов.

Шаг 2 – M. На k-м шаге происходит генерация кандидатов на включение во множество популярных наборов, основанная на правиле (5), с использованием наборов, сгенерированных на предыдущем шаге, если условие, стоящие в правой части условия (5), выполняется для всех k элементах нового кандидата, то вычисляется $\sup(\{a_{l,1}, a_{l,2}, .., a_{l,k}\})$. Если эта величина больше минимальной поддержки то инициализируется $C_l^k = \{a_{l,1}, a_{l,2}, .., a_{l,k}\}$.

Алгоритм завершается либо по завершению M шагов, либо при отсутствии на некотором шаге частичных наборов, так как дальнейшая генерация кандидатов не возможна.

Достоинства алгоритма:

- простота;
- быстрое уменьшение числа сгенерированных кандидатов, при установке высокой минимальной поддержки или относительно разреженном базовом наборе.

Недостатки алгоритма:

- многократное сканирование базового набора;
- большое число сгененированных кандидатов, при слишком большом наборе данных или при слишком низкой поддержке. Так при применении этого алгоритма к набору, содержащему шаблон из 100 элементов необходимо сгенерировать порядка 2^{100} кандидатов и выполнить над ними все необходимые проверки.

Таким образом, алгоритм эффективен только для небольших наборов, либо при высоком уровне минимальной поддержки.

Для улучшения работы используются модификации алгоритма, направленные на

уменьшение числа сканирований входного набора, уменьшения числа сгенерированных кандидатов, распараллеливание [8], но даже они позволяют исправить ситуацию далеко не во всех случаях. Одни из наиболее эффективных методов программной реализации описан в [4].

FP-GROWTH

Этот алгоритм – один из самых эффективных и позволяет избежать не только затратной процедуры генерации кандидатов, но и многократного сканирования входного набора. Метод базируется на предварительной обработке входного набора и преобразование его в специальную компактную древовидную структуру FP-tree (frequent pattern tree) и лишь затем происходит вычисление популярных наборов. В общем виде алгоритм можно записать как [6]:

Шаг 1. Производится сканирования входного набора, и все элементы каждой транзакции сортируются в порядке убывания поддержки этих элементов во всем базовом наборе.

Шаг 2. Фильтрация. Производится удаление тех элементов a, для которых $\sup(\{a\}) < T$.

Шаг 3. Построение префиксного FP-tree из оставшихся элементов.

Шаг 4. Извлечение частных предметных наборов.

Узлом FP-tree является структура, которая хранит значение узла, ссылки на все дочерние элементы и его значение поддержки для текущего узла.

Построение префиксного дерева происходит в несколько этапов:

Этап 1. Построение корневого узла.

Этап 2. Для каждого элемента каждой отсортированной транзакции из входного набора строятся узлы по следующему правилу: если для очередного элемента в текущем узле есть потомок, содержащий этот элемент, то новый узел не создается, а поддержка этого потомка увеличивается на 1, в противном случае создается новый узел-потомок с поддержкой 1. Текущим узлом при этом становится найденный или построенный узел.

Следует заметить, что размер дерева зависит от того, как элементы были отсортированы. Обычно, описанный в шаге 1 способ приводит к значительному уменьшению дерева (хотя и необязательно, что к минимально возможному), но так происходит не всегда.

Пример: для входного набора, состоящего из (a,b,c,d,e), (a,c,d), (c,b,a), (b,a) при минимальной поддержке $T=2$ построенное дерево принимает вид:

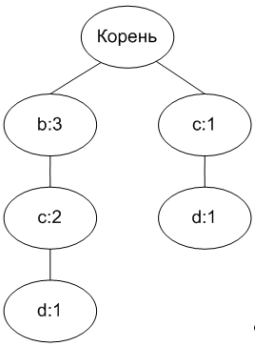

Рис. 1. Пример FP-дерева

Извлечение популярных предметных наборов:

Важной частью этого этапа является понятие условного дерева. Условное поддерево множества A – это FP-tree, содержащее только транзакции, в которые входит множество A.

Алгоритм извлечения популярных наборов имеет вид:

Для каждого элемента в дереве начиная с элемента с наименьшей поддержкой выполнить:

Шаг 1. Добавить этот элемент во множество A.

Шаг 2. Построить условное дерево по этому элементу. В случае если такое дерево оказывается пустым, то записать в результат элементы множества A (они и будут очередным популярным набором), иначе выполнить этот алгоритм для построенного условного дерева.

Шаг 3. Исключить элемент из множества A.

Шаг 4. Исключить элемент из дерева.

Таким образом, дерево проходится рекурсивно снизу вверх полностью, и при этом генерируются все возможные популярные наборы.

Достоинства алгоритма:

• позволяет избежать затратной процедуры генерации кандидатов, характерной для Apriori и Eclat.

• сжатие базового набора в компактную структуру, обеспечивающие быстрое и полное извлечение предметных наборов.

• число сканирования входного набора сокращено до двух.

• размер дерева обычно меньше размера входного набора данных.

Недостатки алгоритма:

• построение дерева – затратная по времени операция.

• в некоторых случаях, вследствие большого числа узлов и связей, размер FP-дерева может намного превышать размер входного набора данных.

Существует множество модификаций алгоритма, направленных на улучшение тех или иных его свойств. Одна из наиболее быстрых реализаций алгоритма описана в [3], а вариант реализации, направленный на уменьшение объема занимаемой памяти – в [1].

ECLAT

Оба алгоритма и Apriori и FP-growth используют так называемое, горизонтальное представление множеств:

$$\begin{aligned} S_1 &= \{a_{1,1}, a_{1,2}, ..., a_{1,n_1}\} \\ S_2 &= \{a_{2,1}, a_{2,2}, ..., a_{2,n_2}\} \\ &... \\ S_k &= \{a_{k,1}, a_{k,2}, ..., a_{k,n_k}\} \end{aligned} \quad (6)$$

Алгоритм Eclat [2], на первом шаге своей работы преобразует горизонтальное представление в вертикальное (так называемые TID-множества) и в дальнейшем ведет работу именно с ним.

$$\begin{aligned} a_1 &= \{S_{1,1}, S_{1,2}, ..., S_{1,m_1}\} \\ a_2 &= \{S_{2,1}, S_{2,2}, ..., S_{2,m_2}\} \\ &... \\ a_k &= \{S_{k,1}, S_{k,2}, ..., S_{k,m_k}\} \end{aligned} \quad (7)$$

При таком представлении поддержка равна отношению мощности множества к общему числу транзакций.

$$\sup(A) = \frac{|A|}{N}. \quad (8)$$

Шаги этого алгоритма аналогичны соответствующим шагам алгоритма Apriort, кроме функции вычисления поддержки кандидата, которая теперь не требует сканирования базы.

Достоинства алгоритма:

• простота

• поддержка для любого элемента рассчитывается без сканирования базового набора

• число сканирований базового набора сокращено до одного раза.

Недостатки алгоритма:

• TID-множества могут оказаться слишком большими, поэтому операции с ними могут занимать длительное время.

• большое число сгенерированных кандидатов, при малом уровне минимальной поддержки.

Описание одной из оптимальных программных реализаций можно найти в [4].

ОПТИМИЗАЦИЯ АЛГОРИТМОВ

В некоторых ситуациях, например, при анализе потоков текстовых данных[12], для транзакций характерно сочетание большой длины с наличием в них элементов, не образующих шаблоны с другими элементами этих транзакций. Такая ситуация приводит к излишней генерации бесперспективных кандидатов в алгоритме Eclat, а также к высокой ветвистости дерева в алгоритме FP-growth, что не позволяет применять алгоритм в полной мере в различных системах обработки и анализа текста[13]. Соответственно одной из основных задач при обработке данных с такими характеристиками – как можно сильнее снизить размер транзакций и их количество перед выполнением на них какого-либо алгоритма вычисления ассоциативных правил.

В дальнейшем в статье будут использоваться следующие обозначения

α -коэффициент поддержки, тогда $T = \alpha N$.

В работе предлагается несколько этапов фильтрации данных, перед передачей их на вход алгоритму поиска шаблонов.

Этап 1. Удаление из всех транзакций всех элементов a, для которых $\sup(\{a\}) < T$. Этот шаг аналогичен первому шагу в алгоритме FP-growth.

Этап 2. Удаление из входного набора всех транзакций, содержащих по окончании первого этапа меньше двух элементов (в данной статье полагается, что отдельный популярный объект – не является набором).

Этап 3. Так как высокая ветвистость простроенного FP дерева возникает в случае разреженных данных, а большое число кандидатов в алгоритмах Eclat и Apriori, в случае, наоборот, в компактных или имеющих много длинных транзакций. Предлагается разделить множество данных на те, которые эффективней обрабатывать с помощью алгоритма FP-Growth и на те, которые предпочтительно обрабатывать с помощью Eclat. Поэтому на этапе 3 происходит преобразование в TID-форму и фильтрация элементов по правилу $\sup(\{a\}) \geq T$, для последующей обработке.

Этап 4. Производится построение матрицы из оставшихся элементов по следующему правилу:

$$d_{i,j} = \begin{cases} 1, \sup(\{a_i, a_j\}) \geq T \\ 0, \sup(\{a_i, a_j\}) < T \end{cases}. \qquad (9)$$

Этап 5. Производится анализ матрицы, построенной на предыдущем этапе. Фактически она представляет собой описание неориентированного графа. Если в этом графе существуют отдельные связные подграфы, то элементы их образующие являются отдельными несвязанными множествами, поэтому обрабатывать их выгодно раздельно.

Пусть исходное множество S разбивается таким образом на множества $S_1, S_2 ... S_n$. Если для некоторого i, $|S_i| > E$, то имеет смысл применять алгоритм FP-growth ко множеству S_i, иначе если $|S_i| > 2$ применяется алгоритм Eclat. В случае $|S_i| = 2$, множество содержит только один популярный набор, который можно сохранить и в дальнейшем исключить S_i из рассмотрения. В последнем же случае $|S_i| = 1$, множество содержит только один элемент, не образующий ни одного популярного набора и такое множество можно также исключить из рассмотрения.

Этап 6. Снижение размерности множеств. На этом этапе проводится операция по снижению размерности множеств S_i, которые должны обрабатываться с помощью алгоритма FP-growth. Основываясь на уже построенной матрице (9) и свойстве (5), из множества S_i убираются элементы a_j, такие что $\forall k, l : d_{k,j} = 1, d_{l,j} = 1 \Rightarrow d_{k,l} = 0$, т. е. такие элементы, которые гарантированно не будут входить в шаблоны мощностью 3. Соответственно такие элементы будут образовывать максимальные шаблоны мощностью 2, которые на этом этапе сохраняются в качестве результата.

Этап 7. Применение алгоритма Eclat к оставшимся группам S_i, для которых $|S_i| \leq E$. Группы, над которыми были проведены вычисления, можно удалить из рассмотрения и оперативной памяти.

Этап 8. На этом этапе остаются только те S_i, для которых выполняется условие $|S_i| > E$. Для применения алгоритма FP-growth необходимо применить обратное TID-преобразование, которое и производится на этом этапе.

Этап 9. Для уменьшения длин транзакций, что повлечет за собой уменьшение ветвистости, высоты построенного дерева и количества шагов при его рекурсивном обходе предлагается на основании построенной матрицы (9) разделить транзакции на связные между собой части. Те части, в которых будет только 1 элемент, удаляются из рассмотрения.

Этап 10. Выполнение алгоритма FP-Growth.

ОЦЕНКА ЭФФЕКТИВНОСТИ

В работе ищутся только максимальные предметные наборы, т. е. такие наборы, что $\sup(\{a_1, a_2, .., a_n\}) \geq T$ и для $\forall b$ выполняется $\sup(\{a_1, a_2, .., a_n, b\}) < T$. Поиск именно таких наборов позволяет сократить время работы алгоритма с одной стороны и автоматически означает, что любой поднабор, входящий в максимальный набор, также является популярным набором с другой.

Для тестирования эффективности предложенной методики фильтрации входных данных использовались два входных набора:

• корпус Retail [7], содержащий данные о покупках людей (кассовые чеки) в магазине. Этот набор содержит 88161 транзакцию, а число различных элементов в ней равно 16469;

• корпус, содержащий множество различных URL, использованный для выявления закономерности в использовании «движков» для построения сайтов. Каждый URL является отдельной транзакцией, а элементом будет являться символ строки. Строку можно записать в следующем виде:

$$S = s_{1,1}s_{1,2}..s_{1,m_1} / s_{2,1}s_{2,2}..s_{2,m_2} / .../ s_{n,1}s_{n,2}..s_{2,m_n} \quad (10)$$

где $s_{i,j}$ – i-й символ в j-м элементе пути. Далее, каждый символ кодируется по следующему правилу:

$$\underline{s}_{i,j} = (s_{i,n-j+1}, i, j), \quad (11)$$

где n – число элементов в пути URL. То есть элемент $\underline{s}_{i,j}$, является структурой хранящий номер элемента пути, начиная с конца, номер символа в этой части и значение символа. Домены в преобразованную строку не входят. Этот корпус содержит 997451 URL и 9716 различных элементов в нем.

Во всех этих тестах параметр $E = 10$.

Ниже приведены графики зависимости количества узлов в построенном FP-дереве, а также число шагов необходимых для его обработки.

Сравнительные графики зависимости числа сгенерированных кандидатов не приводится, т. к. при низком коэффициенте поддержки расчет чистым алгоритмом Eclat не может быть завершен за приемлимое время. Так для корпуса Retail data с уровнем поддержки 0.012, количество сгенерированных кандидатов для обычного Eclat алгоритма равно 13376275, а для алгоритма с предварительно отфильтрованными данными равно 1300.

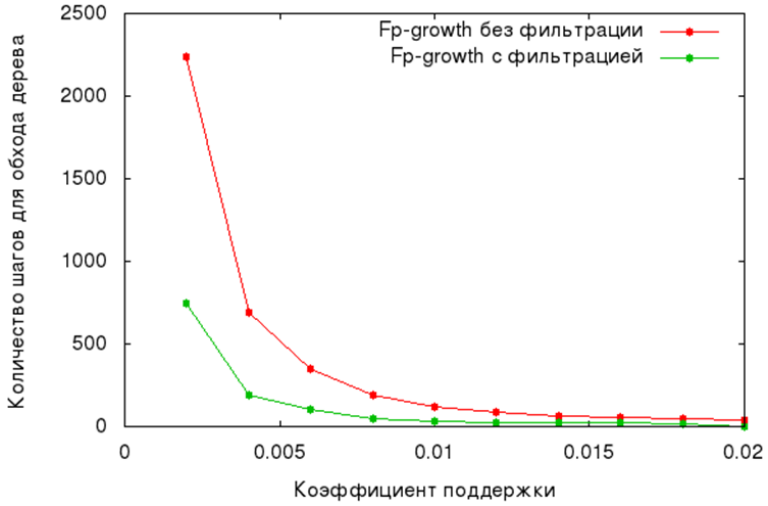

Рис. 2. Зависимость числа шагов обхода FP дерева от коэффициента поддержки для корпуса чеков

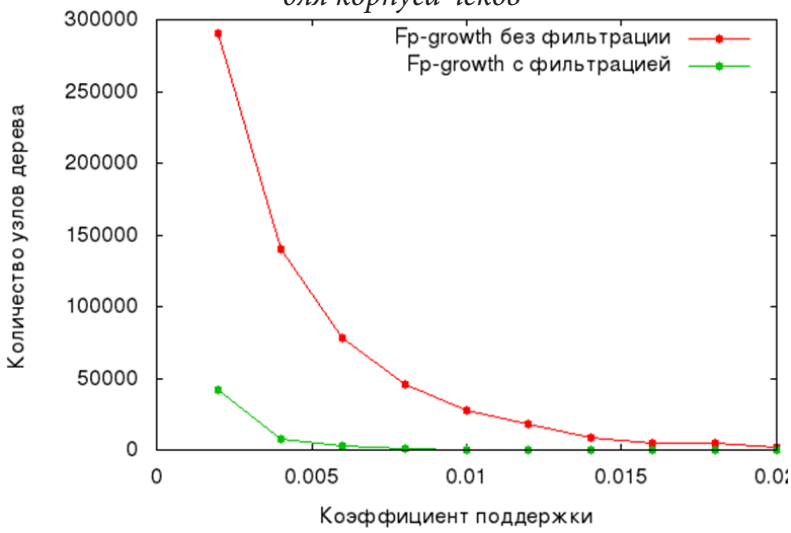

Рис. 3. Зависимость размера числа узлов в дерева от коэффициента поддержки для корпуса чеков

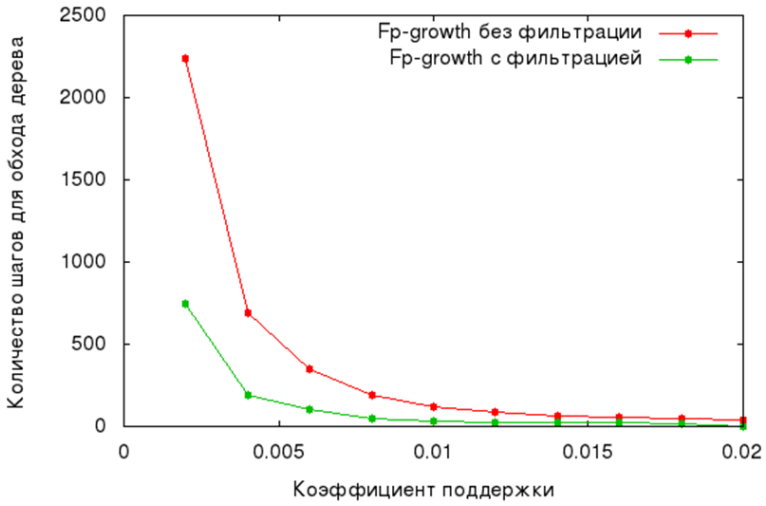

Рис. 4. Зависимость числа шагов обхода FP дерева от коэффициента поддержки для корпуса URL

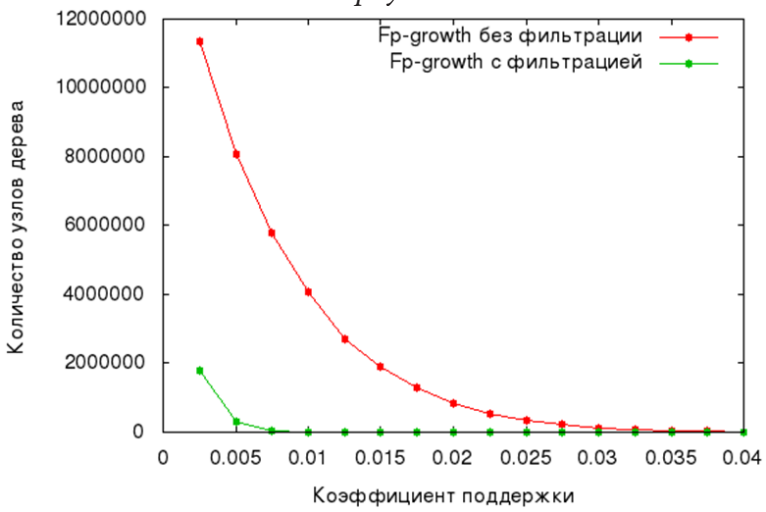

Рис. 5. Зависимость размера числа узлов в дерева от коэффициента поддержки для корпуса URL

ОБСУЖДЕНИЕ РЕЗУЛЬТАТОВ

Из графиков видно, что предлагаемая методика позволяет сократить объем построенного дерева в десятки, а иногда и в сотни раз, однако этот эффект уменьшается при стремлении коэффициента поддержки α к нулю. Это объясняется тем, что по мере уменьшения α общее число связей в матрице (9) увеличивается и количество несвязанных элементов и отдельных групп уменьшается.

При сравнении числа шагов при поиске популярных наборов в алгоритме FP-Growth наблюдается аналогичная ситуация (хоть и проявляющаяся в меньшем масштабе: оптимизация дает выигрыш в 2–4 раза).

ЗАКЛЮЧЕНИЕ

В данной статье был произведен краткий обзор наиболее распространенных алгоритмов поиска ассоциативных правил с заранее заданной минимальной поддержкой, а так же предложен алгоритм оптимизации входных данных для алгоритмов поиска ассоциативных правил и шаблонов и продемонстрирована его эффективность на ряде примеров.

СПИСОК ЛИТЕРАТУРЫ

1. *Стокипный А. Л.* Способ эффективного представления исследуемого набора данных в методах поиска ассоциативных правил / А. Л. Стокипный // Кібернетика та системний аналіз. – Харьков. – 2009. – № 3. – С. 153–161.

2. *Zaki M.* Scalable Algorithm for association mining / M. Zaki // IEEE Transac¬tions on Knowledge and Data Engineering. – 2000. – № 12. – С. 372–390.

3. *Borgelt C.* An Implementation of the FP-growth Algorithm [Электронный ресурс] / C. Borgelt // Workshop Open Source Data Mining Software. – New York: ACM Press. – 2005. – Режим доступа: http://www.osdm.ua.ac.be/papers/p1-borgelt.pdf

4. *Borgelt C.* Efficient Implementations of Apriori and Eclat [Электронный ресурс] / C. Borgelt // Workshop on Frequent Itemset Mining Implementations. – New York: ACM Press. – 2003. – Режим доступа: www.intsci.ac.cn/shizz/fimi.pdf

5. *Agrawal R.* Fast algoritms for missing association rules in large databases / R. Agrawal, R. Srikant // Proceedings of the 20th International Conference on Very Large Data Bases. — Santiago de Chile. – 1994. – С. 487–499.

6. *Han J.* Mining of frequent patterns without candidate generation: a frequent-pattern tree approach / J. Han, J. Pei, Y. Yin, R. Mao // Data mining and analysis discovery. – 2004. – Т. 8 – № 1. – С. 53–87.

7. Frequent Itemset Mining Implementations Repository. Retail [Электронный ресурс]. – (URL: http://fimi.ua.ac.be/data/retail.dat/):

8. *Pol U.* Design and Development of Apriori Algorithm for Sequential to concurrent mining using MPI / U. Pol // International journal of Computers & Technology. – 2013. – Т. 10. – № 7. – С. 1785–1790.

9. *Кириченко Д. О.* Разработка системы полнотекстого поиска с поддержкой поиска обфусциорованных слов. / Д. О. Кириченко, К. Е. Селезнев // Сборник трудов восьмой международной научной конференции «Актуальные проблемы прикладной математики, информатики и механики». – Воронеж, 2013 г. – С. 45–49.

10. *Кириченко Д. О.* Поддержка вариативных запросов в системе полнотекстового поиска / Д. О. Кириченко, К. Е. Селезнев // Сборник трудов XIV Международной конференции «Информатика: проблемы, методология, технологии». – Воронеж, 2014 г. – Т. 2. – С. 282–286.

11. *Кириченко Д. О.* О проблеме интерпретации текстовой информации при анализе заболеваемости населения / Д. О. Кириченко, М. А. Артемов, М. В. Киргинцев // Сборник трудов XIV Международной конференции «Информатика: проблемы, методология, технологии». – Воронеж, 2014 г. – Т. 2. – С. 250–252.

12. *Артемов М. А.* Проблемы классификации потоков текстовых документов / М. А. Артемов, Д. О. Кириченко, М. В.Киргинцев, В. М. Мельников // Современное общество, образование и наука, сб. научных трудов по материалам Международной научно-практической конференции. – Тамбов, 2014 г. – Ч. 7. – С. 15–17.

13. *Артемов М. А., Владимиров А. Н., Селезнев К. Е.* Обзор систем анализа естественного текста на русском языке / М. А. Артемов, А. Н. Владимирова, К. Е. Селезнев // Системный анализ и информационные технологии. – Воронеж. – 2013 г. – № 2. – С. 189–194.

Кириченко Денис Олегович – аспирант кафедры Программного обеспечения и администрирования информационных систем, Воронежский государственный университет.
E-mail: Yalo55@yandex.ru

Артемов М. А. – д.ф.-м.н, заведующий кафедрой Программного обеспечения и администрирования информационных систем, Воронежский государственный университет.
Тел.: (473) 220-83-37
E-mail: Artemov_m_a@mail.ru

Denis Olegovich Kirichenko – graduate student, Chair of Programming Software and Administration of Information Systems, Voronezh State University.
E-mail: Yalo55@yandex.ru

Artemov M. A. – Doctor of Phys. And Mathem. Science, Professor, Head of the Chair of Programming Software and Administration of Information Systems. Voronezh State University.
E mail: Artemov_m_a@mail.ru
Tel.: (473) 220-83-37