© 2021 г. И.Е. ГЕНРИХОВ, канд. физ.-мат. наук (ingvar1485@rambler.ru)
(ООО "Мобайл парк ИТ", Химки),
Е.В. ДЮКОВА, д-р физ.-мат. наук (edjukova@mail.ru)
(ФИЦ "Информатика и управление" РАН, Москва)

ПОИСК ЧАСТЫХ ЭЛЕМЕНТОВ ПРОИЗВЕДЕНИЯ ЧАСТИЧНЫХ ПОРЯДКОВ С ИСПОЛЬЗОВАНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ[1]

Рассматриваются вопросы анализа данных с элементами из декартового произведения конечных частично упорядоченных множеств. Для эффективного поиска частых элементов, порождаемых всеми возможными вариантами бинаризации исходных небинарных данных, используется модификация классического FP-дерева (Frequent Pattern Tree). Сокращение временны́х затрат достигается за счет использования параллельных вычислений на основе технологии CUDA (Compute Unified Device Architecture). Приводятся результаты тестирования построенных параллельных процедур синтеза искомых частых элементов на модельных и реальных данных.

Ключевые слова: декартовое произведение частичных порядков, база данных, частый элемент, FP-дерево, пороговое FP-дерево, параллельные вычисления, технология CUDA.

DOI: 10.31857/S0005231021100032

1. Введение

Задача поиска частых элементов в данных востребована для ряда прикладных областей, среди которых следует выделить нахождение ассоциативных правил в базах данных и машинное обучение. Приведем классическую постановку рассматриваемой задачи для наиболее исследованного случая, а именно для случая бинарных данных [1].

Дано некоторое множество P, элементы которого называются атрибутами, и дана некоторая совокупность D подмножеств множества P (не обязательно различных), называемая базой данных. Подмножества множества P называются наборами атрибутов, а те из них, которые содержатся в D, называются транзакциями. Набор атрибутов Z называется *s-частым*, если отношение числа транзакций, содержащих Z, к числу всех транзакций не менее s. Требуется найти все s-частые наборы атрибутов.

Среди алгоритмов поиска частых наборов атрибутов в бинарных данных наиболее известными являются алгоритмы Apriori, Eclat, FP-Growth, SaM, RElim, LCM, DepthProject ([2–9]). Рассматриваемый в настоящей статье алгоритм FP-Growth ([2, 5]) делает предобработку бинарной базы данных, бла-

[1] Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (проект № 19-01-00430-а).

годаря которой поиск частых наборов атрибутов сводится к анализу компактной древовидной структуры, называемой Frequent Pattern Tree или FP-деревом. В результате алгоритм FP-Growth имеет в ряде случаев преимущество перед другими алгоритмами, например, позволяет избежать слишком затратной процедуры поиска решений, характерной для алгоритмов Apriori [2] и DepthProject [9].

В случае небинарных данных каждый атрибут имеет некоторое множество числовых значений, вместо наборов атрибутов рассматриваются наборы их значений и, как правило, задача сводится к бинарному случаю путем задания для каждого небинарного атрибута некоторого числа (порога), позволяющего перекодировать исходные небинарные данные в бинарные [10, 11].

На практике возникает необходимость нахождения зависимостей в частично упорядоченных данных. В [12] рассмотрена задача поиска ассоциативных правил в данных, представленных в виде декартового произведения частичных порядков, и в связи с этим введено понятие s-частого элемента для множества $P = P_1 \times \ldots \times P_n$ при условии, что $P_1, \ldots, P_n$ – конечные частично упорядоченные числовые множества.

На множестве P устанавливается частичный порядок. Считается, что элемент $y = (y_1, \ldots, y_n) \in P$ следует за элементом $x = (x_1, \ldots, x_n) \in P$, если y_i следует за x_i при $i = 1, \ldots, n$ (запись $x \preccurlyeq y$). Элементы $x, y \in P$ называются *сравнимыми*, если один из этих элементов следует за другим, иначе x и y называются *несравнимыми*.

Предполагается, что каждое множество (атрибут) P_i содержит *наименьший элемент*, т.е. такой элемент l_i, для которого выполнено $l_i \preccurlyeq x_i$ для любого $x_i \in P_i$. Элемент $x_i \in P_i$ называется *существенным* значением элемента $x = (x_1, \ldots, x_i, \ldots, x_n) \in P$, если $x_i \neq l_i$. Допускается, что база D не содержит $l = (l_1, \ldots, l_n)$.

В случае бинарных данных $P_i = \{0, 1\}$, $i = 1, \ldots, n$, и в P_i установлен порядок $0 \preccurlyeq 1$, $0 \neq 1$ (здесь $l_i = 0, i = 1, \ldots, n$).

Пусть $x \in P$, $x \neq l$ и $S_D(x)$ – число транзакций y в D таких, что $x \preccurlyeq y$. Элемент x называется *s-частым*, если $S_D(x) / |D| \geq s$, иначе x называется *s-нечастым* (здесь и далее $|D|$ – число транзакций в базе D). Заметим, что если $x \preccurlyeq y$, то $S_D(x) \geq S_D(y)$. Элемент x называется *максимальным s-частым* элементом, если $S_D(x) / |D| \geq s$ и $S_D(y) / |D| < s$ для любого y, такого что $x \preccurlyeq y$, $x \neq y$ (т.е. из условия $x \preccurlyeq y$, $x \neq y$ следует, что y – s-нечастый элемент в P).

В [13–15] рассмотрены вопросы бинаризации множества $P = P_1 \times \ldots \times P_n$. Для каждого множества P_i, $i \in \{1, \ldots, n\}$, строится множество "значимых порогов" $Q_i \subseteq P_i$. Каждый набор порогов $H = \{p_1, \ldots, p_n\}$, в котором $p_i \in Q_i$ при $i = 1, \ldots, n$, порождает произведение бинарных частичных порядков P^H. Частые элементы множества P^H названы *пороговыми* частыми элементами. Поставлена задача перечисления максимальных пороговых частых элементов, порождаемых всеми возможными вариантами бинарной перекодировки. Заметим, что находить частые элементы, не являющиеся максимальными,

неэффективно как по времени, так и по памяти. Для решения поставленной задачи предложен подход, основанный на построении порогового FP-дерева (TFP-дерева).

Модель TFP-дерева является модификацией модели классического бинарного FP-дерева [2, 5]. В TFP-дереве для каждого небинарного атрибута P_i строится так называемая полная вершина, содержащая информацию о возможных вариантах бинаризации этого атрибута. При спуске из полной вершины строится либо новая полная вершина, либо корневая вершина классического бинарного FP-дерева. Бинарное FP-дерево строится тогда, когда в текущей ветви все значения небинарных атрибутов перекодированы в бинарные значения.

В реальных задачах может формироваться большое число значимых наборов порогов и для поиска максимальных пороговых частых элементов с использованием TFP-дерева требуются существенные вычислительные ресурсы. В настоящей статье с целью ускорения процедуры поиска искомых частых элементов разработаны параллельные алгоритмы на основе технологии CUDA [16, 17], позволяющей выполнить операции, не требующие больших временны́х затрат на центральном процессоре, а все сложные операции на графическом процессоре (GPU). Реализованы две схемы распараллеливания: блочная и одиночная. В блочной схеме множество всех "значимых" наборов порогов H_D разбивается на непересекающиеся подмножества, каждое из которых подается на отдельный вычислительный блок GPU для синтеза максимальных пороговых частых элементов. При одиночном распараллеливании для нахождения максимальных пороговых частых элементов, порождаемых набором порогов из H_D, используется один вычислительный блок GPU.

В разделе 2 введены основные понятия. В разделе 3 описана процедура построения TFP-дерева. В разделах 4 и 5 дано описание алгоритмов поиска максимальных пороговых частых элементов в TFP-дереве на основе параллельных вычислений с использованием технологии CUDA и приведены результаты тестирования этих алгоритмов на модельных данных и на реальных задачах из репозитория UCI [18].

2. Основные понятия

Рассмотрим случай небинарных данных. Задачу поиска частых элементов в произведении частичных небинарных порядков P сведем к бинарному случаю путем задания набора порогов $H = \{p_1, \ldots, p_n\}$, в котором $p_i \in P_i$ при $i = 1, \ldots, n$, $p_i \neq l_i$. В каждом элементе $x = (x_1, \ldots, x_n)$ множества P заменим значение x_i, $i \in 1, \ldots, n$, на 1, если $p_i \preccurlyeq x_i$, и заменим на 0 в противном случае. В результате такой кодировки получаем вместо множества P множество $P^H = P_1^H \times \ldots \times P_n^H$, где $P_i^H = \{0, 1\}$, $i \in \{1, \ldots, n\}$, и в каждом P_i^H установлен порядок $0 \preccurlyeq 1$, $0 \neq 1$. Вместо базы данных D получаем базу данных D_H.

Порогу p, $p \in P_i$, поставим в соответствие элемент $\varphi_i(p) = (x_1, \ldots, x_n)$ из P, в котором $x_i = p$ и $x_j = l_i$ при $j \neq i$. Тогда порог p называется *значимым*, если $S_D(\varphi_i(p)) \geq s$, т.е. элемент $\varphi_i(p)$ является s-частым в P. Предполагается, что каждое P_i имеет хотя бы один значимый порог.

Набор порогов $H = (p_1, \ldots, p_n)$ называется *значимым*, если для любого $i \in \{1, \ldots, n\}$ порог p_i – значимый. Для бинарной перекодировки будем выбирать только значимые наборы порогов.

Набор порогов $(p_1, \ldots, p_n)$, в котором для каждого p_i, $i \in \{1, \ldots, n\}$, выполняется условие $S_D(\varphi_i(p_i)) = \max_{p \in P_i} S_D(\varphi_i(p))$, называется *оптимальным значимым*. Оптимальный значимый набор порогов позволяет находить наибольшее число пороговых частых элементов.

Ставится задача нахождения всех максимальных пороговых частых элементов, которые порождаются всеми значимыми наборами порогов. В силу большого числа s-частых элементов имеет смысл искать и хранить только те из них, которые являются максимальными.

Рассматриваются три способа решения поставленной задачи. Первый способ основан на последовательном поиске значимых наборов порогов и синтезе для каждого такого набора соответствующего бинарного FP-дерева. Второй способ основан на нахождении искомых элементов в TFP-дереве. Третий способ основан на параллельном поиске частых элементов в TFP-дереве с применением технологии CUDA. Результаты экспериментального сравнения по скорости счета указанных подходов приводятся в разделе 5.

3. Синтез порогового FP-дерева (TFP-дерева) и пороговых частых элементов произведения частичных небинарных порядков

Отметим, что описание варианта FP-дерева для произведения бинарных частичных порядков (далее бинарного FP-дерева) приведено в [14].

Введем обозначения: a_{ri}, $r \in \{1, \ldots, m\}$, $i \in \{1, \ldots, n\}$, – значение атрибута P_i для транзакции S_r в базе D; X – множество небинарных атрибутов в D; H – набор вида $(1, \ldots, 1)$ длины n; H_D – множество всех значимых наборов порогов. Не ограничивая общности, можно считать, что каждый небинарный атрибут имеет хотя бы один значимый порог.

Опишем алгоритм синтеза TFP-дерева. Данный алгоритм является рекурсивным. Положим на первом шаге $D' = D$, $\hat{X} = X$, $\hat{H} = H$, $H_D = \varnothing$. Далее на каждом шаге рекурсии осуществляется следующая последовательность действий 1–3.

1. Если $\hat{X} = \varnothing$, то осуществляется переход к 3, иначе осуществляется 2.
2. Выбирается атрибут $P_i \in \hat{X}$ с множеством значимых порогов Q_i и создается полная вершина с меткой (P_i, Q_i). Далее осуществляется ветвление из вершины (P_i, Q_i) по каждому порогу $p \in Q_i$. Для порога p строится одна дуга и создается база D_p путем бинарной перекодировки атрибута P_i в текущей для данной ветви базе D' по правилу: $a_{ri} = 1$, если $p \preccurlyeq a_{ri}$, иначе $a_{ri} = 0$. Полагается $D' = D_p$, $\hat{X} = \hat{X} \backslash \{P_i\}$, в наборе $\hat{H}$ координата с номером i заменяется на p и осуществляется рекурсивный переход к 1.
3. Полагается $H_D = H_D \cup \left\{\hat{H}\right\}$, и для набора порогов $\hat{H}$ строится бинарное FP-дерево. Рекурсия останавливается.

После построения TFP-дерева для каждого построенного значимого набора порогов осуществляется поиск максимальных пороговых s-частых элементов.

Пусть $H_1 = \{p_1, \ldots, p_n\}$ и $H_2 = \{q_1, \ldots, q_n\}$ – два значимых набора порогов таких, что $p_t \neq q_t$ для некоторого $t \in \{1, \ldots, n\}$ и $p_i = q_i$, $i \neq t$, $i \in \{1, \ldots, n\}$.

Пусть $W(H_1)$ – множество пороговых s-частых элементов в P^{H_1}, $W(H_2)$ – множество пороговых s-частых элементов в P^{H_2}. Имеют место приводимые далее утверждения 1 и 2.

Утверждение 1. Если $S_D(\varphi_t(q_t)) \leq S_D(\varphi_t(p_t))$, *то* $W(H_2) \subseteq W(H_1)$.

Доказательство. Пусть $S_D(\varphi_t(q_t)) \leq S_D(\varphi_t(p_t))$. Покажем, что $W(H_2) \subseteq W(H_1)$. Возьмем произвольный элемент $x = (x_1, \ldots, x_n)$ из $W(H_2)$. Заметим, что наборы H_1 и H_2 отличаются друг от друга по значению только одного порога с индексом t, поэтому база D_{H_1} отличается от базы D_{H_2} только по атрибуту с номером t. Тем самым если $x_t = 0$, то x будет также s-частым элементом и в P^{H_1} по построению базы D_{H_1}. Если же $x_t \neq 0$, то из условия $S_D(\varphi_t(q_t)) \leq S_D(\varphi_t(p_t))$ получаем, что в D_{H_1} содержится не меньше транзакций, чем в D_{H_2}, следующих за элементом x. Тем самым элемент x также принадлежит и множеству $W(H_1)$. Утверждение 1 доказано.

Пусть $W_t(H_1)$ – множество пороговых s-частых элементов в P^{H_1}, в которых значение атрибута с индексом t равно нулю, $W_t(H_2)$ – множество пороговых s-частых элементов в P^{H_2}, в которых значение атрибута с индексом t равно нулю.

Утверждение 2. Если $S_D(\varphi_t(q_t)) \leq S_D(\varphi_t(p_t))$, *то* $W_t(H_2) = W_t(H_1)$.

Доказательство. Пусть $S_D(\varphi_t(q_t)) \leq S_D(\varphi_t(p_t))$. Покажем, что $W_t(H_2) = W_t(H_1)$. Возьмем произвольный элемент $x = (x_1, \ldots, x_n)$ из $W_t(H_1)$. Заметим, что наборы H_1 и H_2 отличаются друг от друга по значению только одного порога с индексом t, поэтому база D_{H_1} отличается от базы D_{H_2} только по атрибуту с номером t. Из $x_t = 0$ и $S_D(\varphi_t(q_t)) \leq S_D(\varphi_t(p_t))$ получаем, что в базе D_{H_2} найдется $s|D_{H_2}|$ транзакций, следующих за элементом x. Тем самым элемент x также принадлежит и множеству $W_t(H_2)$. Аналогично рассматривается обратная ситуация, когда элемент x принадлежит $W_t(H_2)$. Утверждение 2 доказано.

Применяя утверждения 1 и 2, процесс поиска всех максимальных пороговых s-частых элементов с использованием TFP-дерева можно существенно ускорить. С этой целью в каждом Q_i, $i \in \{1, \ldots, n\}$, пороги упорядочиваются по убыванию значения величины $S_D(\varphi_i(p))$, $p \in Q_i$, и ветвление из полной вершины (P_i, Q_i) осуществляется в соответствии с выбранным порядком. Множество H_D упорядочивается в порядке построения его элементов. Нетрудно видеть, что в таком случае в H_D первым по порядку будет оптимальный значимый набор порогов H_{opt} (раздел 2).

Пусть $F_{\max}(H)$, $H \in H_D$, – множество максимальных пороговых s-частых элементов для набора порогов H; $F(H_D)$ – множество пар вида $(F_{\max}(H), H)$, $H \in H_D$. Для поиска требуемых частых элементов применяются процедуры AllMaxSets и MaxSets, результатом работы которых является соответственно построение множества $F(H_D)$ и пары $(F_{\max}(H), H)$, $H \neq H_{opt}$.

На первом шаге процедура AllMaxSets по FP-дереву, построенному для H_{opt}, классическим способом, находит множество $F_{\max}(H_{opt})$ и пару $(F_{\max}(H_{opt}), H_{opt})$ добавляет в $F(H_D)$ (первоначально $F(H_D) = \varnothing$). Далее процедура AllMaxSets осуществляет построение множества $F(H_D)$ путем последовательного вызова процедуры MaxSets для каждого $H \in H_D$, $H \neq H_{opt}$ (в порядке следования наборов в H_D).

На вход процедуры MaxSets подаются набор порогов $H = \{q_1, \ldots, q_n\}$, множество $F_{\max}(H) = \varnothing$ и текущее множество (последовательность) $F(H_D)$. На выходе из MaxSets пара $(F_{\max}(H), H)$ добавляется в $F(H_D)$. В процедуре MaxSets осуществляется следующая последовательность действий 1–3.

1. В последовательности $F(H_D)$ среди пар $(F_{\max}(H'), H')$, $H' \neq H$, $H' = \{p_1, \ldots, p_n\}$, $S_D(\varphi_i(q_i)) \leq S_D(\varphi_i(p_i))$, $i \in \{1, \ldots, n\}$, ищутся пары с минимальным числом неравенств $S_D(\varphi_i(q_i)) < S_D(\varphi_i(p_i))$. Среди этих пар берется пара $(F_{\max}(H^*), H^*)$ с максимальным номером и создается множество $T(H^*)$ индексов i, для каждого из которых при $H' = H^*$ верно указанное выше строгое неравенство. Далее осуществляется переход к 2.
2. Просматриваются элементы из $F_{\max}(H^*)$. Элемент A из $F_{\max}(H^*)$, в котором значения атрибутов с индексами из $T(H^*)$ равны 0, добавляется в $F_{\max}(H)$, иначе осуществляется переход к 3. После просмотра всех элементов в $F_{\max}(H^*)$ построение множества $F_{\max}(H)$ заканчивается и осуществляется выход из процедуры MaxSets.
3. Пусть $M(A)$ – множество различных элементов из P^H, каждое из которых получается из A заменой одного элемента 1 на элемент 0. Если A – s-частый элемент в D_H и $A \notin F_{\max}(H)$, то элемент A добавляется в $F_{\max}(H)$. Если же A – s-нечастый элемент в D_H и $|A| > 2$, то действие 3 осуществляется для каждого элемента из $M(A)$. Во всех других случаях элемент A не добавляется в $F_{\max}(H)$.

Следует заметить, что в связи с особенностью построения TFP-дерева на первом шаге работы процедуры MaxSets всегда найдется хотя бы одна пара $(F_{\max}(H^*), H^*)$. Дело в том, что спуск из полной вершины осуществляется последовательно начиная с рассмотрения наиболее значимого порога.

4. Поиск пороговых частых элементов на базе TFP-дерева и технологии параллельных вычислений CUDA

В случае больших данных для построения пороговых частых элементов произведения частичных порядков требуются существенные вычислительные ресурсы. В данном разделе с целью уменьшения временны́х затрат при поиске искомых элементов реализованы параллельные схемы вычислений на основе применения технологии параллельных вычислений CUDA (Compute Unified Device Architecture), которая позволяет использовать графический процессор [16, 17].

Графический процессор (GPU) состоит из однородных вычислительных элементов (мультипроцессоров) с общей памятью. Каждый мультипроцессор способен исполнять параллельно тысячи вычислительных “нитей”. Нити могут быть сгруппированы в вычислительные потоки, имеющие общий кэш и быструю разделяемую память для обмена данными между нитями потока.

Вычислительные потоки также могут быть сгруппированы в вычислительные блоки для лучшего распараллеливания вычислений. Применение вычислений на GPU наиболее эффективно в решении задач, в которых число арифметических операций велико по сравнению с операциями над памятью.

Далее описаны два параллельных алгоритма поиска максимальных пороговых частых элементов произведения частичных порядков: PTFP-tree и DPTFP-tree. В алгоритме PTFP-tree реализована одиночная схема распараллеливания, а в алгоритме DPTFP-tree применена блочная схема. В качестве базового алгоритма поиска максимальных пороговых частых элементов предлагается алгоритм, описанный в разделе 3. В этом алгоритме наиболее существенное время требуется для работы процедуры AllMaxSets. Поэтому в разработанных параллельных алгоритмах данные вычисления полностью перенесены на GPU.

В алгоритме PTFP-tree реализована процедура MaxSetsGPU, осуществляющая построение множества $F_{\max}(H)$, $H \in H_D$, $H \neq H_{opt}$, на GPU. На вход MaxSetsGPU подаются исходная база данных D, значение уровня поддержки s, число вычислительных нитей z, набор порогов $H = \{q_1, \ldots, q_n\}$, множество $F_{\max}(H) = \varnothing$ и пара $(F_{\max}(H^*), H^*) = (F_{\max}(H_{opt}), H_{opt})$. На выходе из MaxSetsGPU пара $(F_{\max}(H), H)$ добавляется в $F(H_D)$.

В процедуре MaxSetsGPU выполняются следующие действия 1–3.

1. Просматриваются элементы из $F_{\max}(H^*)$, $H^* = \{p_1, \ldots, p_n\}$. Если для $A \in F_{\max}(H^*)$, $A = \{a_1, \ldots, a_n\}$, найдется $t \in \{1, \ldots, n\}$ такое, что $a_t = 1$, $p_t \neq q_t$, то осуществляется переход к 2. В противном случае элемент A добавляется в $F_{\max}(H)$. Данная проверка элемента A выполняется параллельно z вычислительными нитями GPU. После просмотра всех элементов в $F_{\max}(H^*)$ происходит переход к 3, после которого построение множества $F_{\max}(H)$ заканчивается и осуществляется выход из процедуры MaxSetsGPU.
2. Пусть $M(A)$ – множество различных элементов из P^H, каждое из которых получается из A заменой одного элемента 1 на элемент 0. Если A – s-частый элемент в D_H и $A \notin F_{\max}(H)$, то элемент A добавляется в $F_{\max}(H)$. Если же A – s-нечастый элемент в D_H и $|A| > 2$, то действие 2 осуществляется для каждого элемента из $M(A)$. Во всех других случаях элемент A не добавляется в $F_{\max}(H)$.
3. Из множества $F_{\max}(H)$ удаляются элементы, которые предшествуют другим элементам из $F_{\max}(H)$. Данный блок выполняется параллельно z вычислительными нитями GPU.

В алгоритме DPTFP-tree в отличие от алгоритма PTFP-tree построение множества $F_{\max}(H)$ осуществляется параллельно для всех $H \in H_D$, $H \neq H_{opt}$. Для этого создается $|H_D| - 1$ вычислительных потоков, в каждом из которых вызывается процедура MaxSetsGPU. Также для лучшей загрузки мультипроцессоров GPU в алгоритме DPTFP-tree в действиях 1 и 2 процедуры MaxSetsGPU активно применяется динамический параллелизм (возможность динамически порождать внутри потока новые вычислительные потоки без возврата к коду, исполняемому на CPU [16]).

Дополнительное отличие состоит в том, что каждый вычислительный поток "сообщает" свой результат всем вычислительным потокам, которые находятся в очереди на выполнение, и те из них, которые запустятся далее, перед каждым вызовом процедуры MaxSetsGPU действуют следующим образом. Среди пар $(F_{\max}(H'), H')$, $H' \neq H$, $H' = \{p_1, \ldots, p_n\}$, $H = \{q_1, \ldots, q_n\}$, $S_D(\varphi_i(q_i)) \leq S_D(\varphi_i(p_i))$, $i \in \{1, \ldots, n\}$, множества $F(H_D)$ ищутся пары с минимальным числом неравенств $S_D(\varphi_i(q_i)) < S_D(\varphi_i(p_i))$, $i \in \{1, \ldots, n\}$, из которых выбирается пара $(F_{\max}(H^*), H^*)$ с максимальным номером. Поиск пары $(F_{\max}(H^*), H^*)$ полностью выполняется на GPU с применением динамического параллелизма.

5. Результаты численных экспериментов на реальных и модельных данных

Выполнены эксперименты на случайных модельных данных, содержащих как бинарные атрибуты, так и небинарные целочисленные атрибуты. Число небинарных атрибутов составляло 10 % от общего числа атрибутов. Значение бинарного атрибута в транзакции полагалось равным 0 с вероятностью v, $v \in [0{,}5; 0{,}9]$, и равным 1 с вероятностью $1 - v$. Число v выбиралось с применением датчика случайных чисел. Значения небинарного атрибута выбирались из интервала $[0; 9]$ с равной вероятностью. Число транзакций m изменялось от 60 до 1 800 000, число атрибутов n изменялось от 30 до 40. Максимальные пороговые s-частые элементы искались при $s = 0{,}3$. Время счета измерялось в миллисекундах.

На рис. 1–4 приведено время поиска всех максимальных пороговых s-частых элементов алгоритмом FP-tree, основанным на последовательном синтезе для каждого значимого набора порогов классического бинарного FP-дерева, алгоритмом TFP-tree, основанным на построении TFP-дерева, и параллельными алгоритмами, описанными в разделе 4. Для каждого алгоритма приведено среднее время поиска при анализе 20 случайных баз данных размера $m \times n$. Дополнительно указаны число значимых наборов порогов $|H_D|$ и число найденных максимальных пороговых s-частых элементов $|F_D|$. На рис. 4 представлены результаты работы алгоритмов DPTFP-tree и TFP-tree при числе небинарных атрибутов k, равном $1, \ldots, 7$.

Из полученных результатов можно сделать следующие выводы.

Более быстрым из двух разработанных параллельных алгоритмов является алгоритм DPTFP-tree (рис. 1). При $n = 40$ алгоритм DPTFP-tree в среднем в 6 раз работает быстрее алгоритма PTFP-tree.

Алгоритм TFP-tree работает существенно быстрее алгоритма FP-tree. Например, при $n = 40$ время счета TFP-tree в 20–40 раз меньше времени счета FP-tree (рис. 2). Этот вывод ранее был получен в [13–15].

При большом числе транзакций (более 30 000 транзакций) и относительно небольшом числе значимых наборов порогов (не более 2500) алгоритм DPTFP-tree работает быстрее алгоритма TFP-tree в среднем в четыре раза (рис. 3). Заметим, что при $n = 40$, $m \in \{100, 200, 300\}$ время счета TFP-tree меньше времени счета DPTFP-tree в 3–10 раз (рис. 1–2) за счет того, что копирование данных между GPU и CPU занимает слишком большое время.

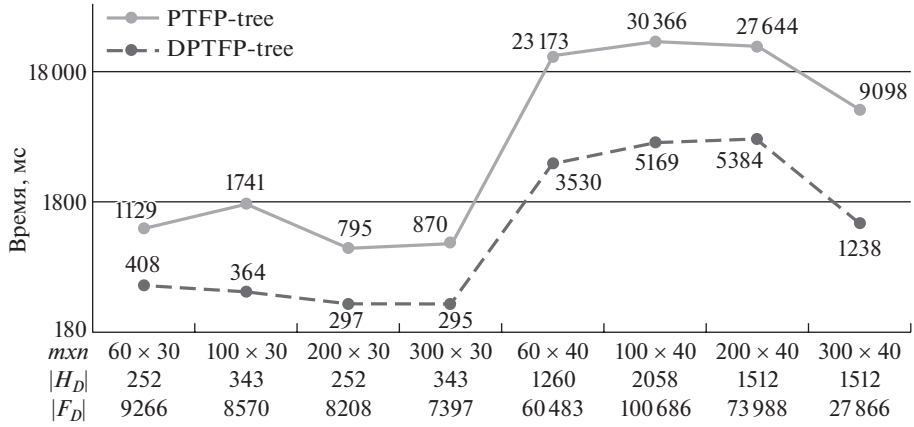

Рис. 1. Зависимость времени поиска максимальных пороговых s-частых элементов параллельными алгоритмами от размерности базы данных.

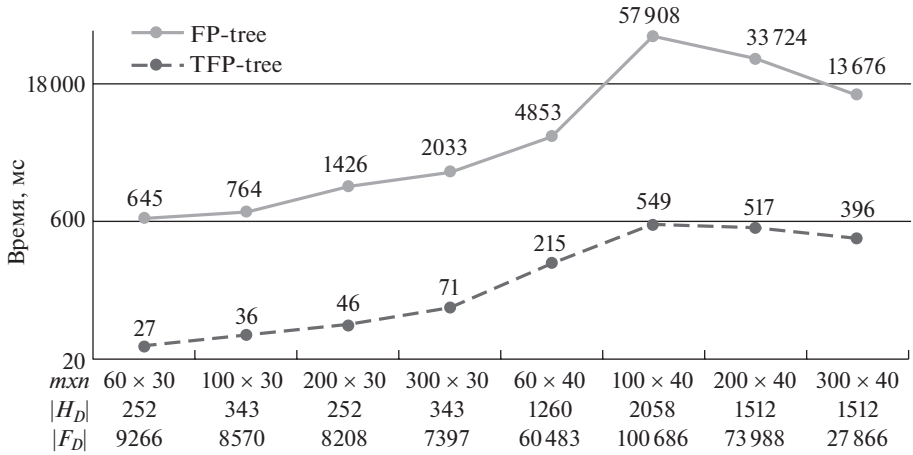

Рис. 2. Зависимость времени поиска максимальных пороговых s-частых элементов последовательными алгоритмами от размерности базы данных.

При большом числе значимых наборов порогов (более 600 000) и небольшой размерности базы данных ($n = 40$, $m = 300$) алгоритм DPTFP-tree работает быстрее алгоритма TFP-tree в среднем в три раза (рис. 4).

Тестирование алгоритмов DPTFP-tree и TFP-tree также осуществлялось на 7 реальных задачах из репозитория UCI [18]: Stone Flakes (задача № 1), Cloud (задача № 2), Wine Quality White (задача № 3), Clickstream data for online shopping (задача № 4), Wholesale customers (задача № 5), Adult (задача № 6), Heart Disease (задача № 7).

В таблице представлены результаты счета для задач 1–7. В задачах 4–7 варьировались параметры: s (значение уровня поддержки) и h (максимальное число значимых порогов для небинарного атрибута). При отсутствии ограничения на значение параметра h в соответствующей ячейке таблицы поставлен

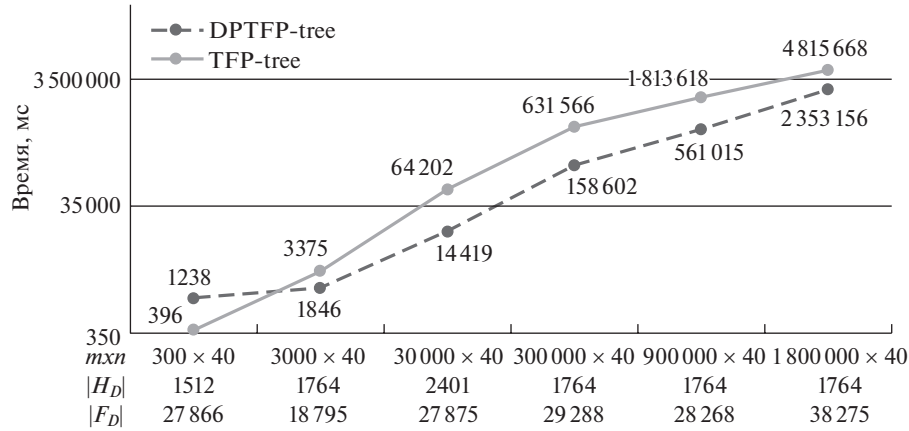

Рис. 3. Зависимость времени поиска максимальных пороговых s-частых элементов от m и при $n = 40$.

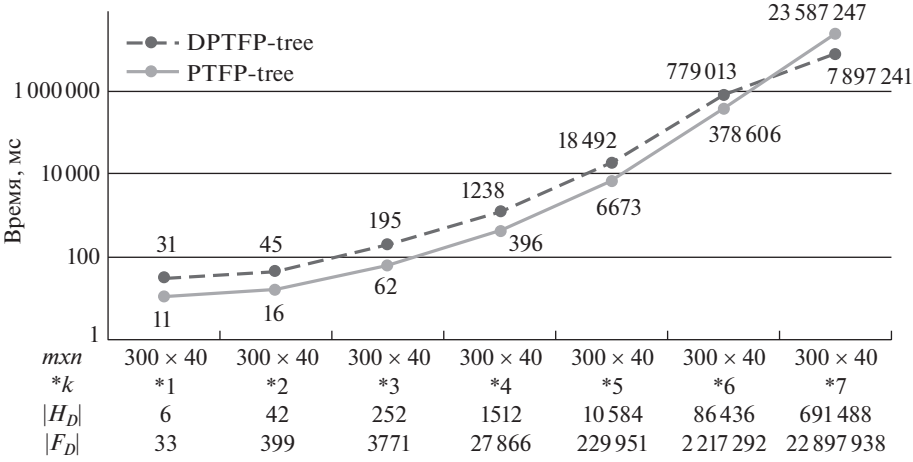

Рис. 4. Зависимость времени поиска максимальных пороговых s-частых элементов от числа небинарных атрибутов k при $n = 40$ и $m = 300$.

знак *. Для наглядности результатов счета приведены значения величины r, равной $(-t_G/t_C)$, если $t_G > t_C$, иначе равной t_C/t_G, где t_C – время работы алгоритма TFP-tree, t_G – время работы алгоритма DPTFP-tree. Серым цветом выделены значения величины r в случае, когда достигается ускорение алгоритма DPTFP-tree относительно алгоритма TFP-tree.

Из результатов, представленных в таблице, следует, что при большом числе значимых наборов порогов и при большом числе транзакций алгоритм DPTFP-tree до 3,6 раз работает быстрее алгоритма TFP-tree. На задачах с небольшим числом транзакций или с небольшим числом значимых наборов порогов алгоритм DPTFP-tree до 2,3 раз работает медленнее алгоритма TFP-tree.

Эффективность алгоритмов TFP-tree и DPTFP-tree на реальных задачах

Описание задачи				$\lvert H_D\rvert$	$\lvert F_D\rvert$	TFP-tree, с	DPTFP-tree, с	r
№	$m \times n$	s	h					
1	79×8	90	7	84 672	695 634	119	167	–1,4
2	2048×10	90	3	786 432	786 432	22 740	6400	3,6
3	4898×12	90	2	354 294	354 294	4520	1609	2,8
4	165474×12	90	*	768	2501	41	14	2,9
4	165474×12	80	*	20 300	103 052	1159	372	3,1
4	165474×12	70	*	216 384	1 511 248	16 442	4884	3,4
5	440×8	90	5	46 656	46 656	25	31	–1,2
5	440×8	90	6	117 649	117 649	180	186	–1,0
5	440×8	90	7	262 144	262 144	1087	864	1,3
6	32560×15	90	60	42 090	186 900	504	398	1,3
6	32560×15	90	70	48 990	220 480	548	496	1,1
6	32560×15	90	80	55 890	254 532	840	588	1,4
7	303×14	90	10	15 488	94 696	6	9	–1,5
7	303×14	80	10	29 282	123 022	18	42	–2,3
7	303×14	70	10	197 568	787 141	727	1230	–1,7

Таким образом, показана целесообразность применения в случае данных большого размера параллельного поиска максимальных пороговых s-частых элементов, осуществляемого на базе TFP-дерева и технологии CUDA. Установлено, что время счета заметно снижается при использовании динамического параллелизма в вычислениях на графическом ускорителе GPU.

6. Заключение

Рассмотрена актуальная задача логического анализа данных – задача поиска (максимальных) пороговых частых элементов произведения частичных порядков. Для ее решения на основе использования TFP-дерева (порогового FP-дерева) и технологии CUDA (Compute Unified Device Architecture) построены и исследованы параллельные алгоритмы. Конструкция TFP-дерева является модификацией конструкции классического бинарного FP-дерева на случай небинарных данных и ранее была предложена в [13–15].

Приведены результаты тестирования построенных алгоритмов на модельных и реальных данных. Показано, что в случае больших данных использование TFP-дерева заметно снижает время синтеза искомых частых элементов по сравнению с их последовательным поиском на базе классического FP-дерева, и установлена целесообразность применения технологии CUDA для решения поставленной задачи.

СПИСОК ЛИТЕРАТУРЫ

1. *Agrawal R., Imielinski T., Swami A.* Mining Association Rules between Sets of Items in Large Databases // Proc. 1993 ACM SIGMOD Int. Conf. on Management of Data. 1993. P. 207–216.

2. *Aggarwal C., Jiawei H.* Frequent Pattern Mining. Springer, 2014.
3. *Agrawal R., Srikant R.* Fast Algorithms for Mining Association Rules in Large Databases // VLDB Conf. 1994. P. 487–499.
4. *Zaki M.J.* Scalable Algorithms for Association Mining // IEEE Trans. Knowledge and Data Engineering. 2000. V. 12. No. 3. P. 372–390.
5. *Han J., Pei H., Yin Y.* Mining Frequent Patterns without Candidate Generation // Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX). 2000. P. 1–12.
6. *Borgelt C., Wang X.* SaM: A Split and Merge Algorithm for Fuzzy Frequent Item Set Mining // IFSA/EUSFLAT Conf. 2009. P. 968–973.
7. *Borgelt C.* Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination // OSDM. 2005. P. 66–70.
8. *Takeaki U., Masashi K., Hiroki A.* LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets // FIMI '04. 2004.
9. *Agrawal R., Aggarwal C., Prasad V.* Depth First Generation of Long Patterns // KDD '00: Proc. 6th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining. 2000. P. 108–118.
10. *Imberman S.P., Domanski B.* Finding Association Rules from Quantitative Data Using Data Booleanization // AMCIS Proc. 2001. P. 369–375.
11. *Angiulli F., Ianni G., Palopoli L.* On the Complexity of Inducing Categorical and Quantitative Association Rules // Theoretical Comput. Sci. 2004. V. 314. No. 1. P. 217–249.
12. *Elbassioni K.M.* On Finding Minimal Infrequent Elements in Multi-dimensional Data Defined over Partially Ordered Sets // arXiv:1411.2275. 2014.
13. *Генрихов И.Е., Дюкова Е.В.* О поиске ассоциативных правил в небинарных данных // 19-я Всеросс. конф. с междунар. участием "Математические методы распознавания образов". М.: РАН, 2019. С. 15–19.
14. *Генрихов И.Е., Дюкова Е.В.* Поиск частых элементов произведения частичных порядков и ассоциативные правила // Информационные технологии и нанотехнологии (ИТНТ-2020). Сб. тр. по матер. VI Междунар. конф. и молодежной школы (Самара, 26–29 мая). Самара: Изд-во Самар. ун-та, 2020. Том 4. Науки о данных. С. 620–629.
15. *Genrikhov I.E., Djukova E.V.* Finding Frequent Elements for a Product of Partial Orders and Association Rules // Int. Conf. on Information Technology and Nanotechnology (ITNT), Samara, Russia. 2020. P. 1–5.
16. *Боресков А.В., Харламов А.А., Марковский Н.Д.* Параллельные вычисления на GPU. Архитектура и программная модель CUDA. М.: Изд-во Моск. ун-та, 2012.
17. *Cheng J., Grossman M., McKercher T.* Professional CUDA C Programming. N.Y.: Wrox, 2014.
18. *Lichman M.* UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. 2013. http://archive.ics.uci.edu/ml.

Статья представлена к публикации членом редколлегии А.А. Лазаревым.

Поступила в редакцию 24.01.2021

После доработки 05.04.2021

Принята к публикации 30.06.2021