МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Мобильные и сетевые технологии
(направленность (профиль)/ специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Программный модуль для поиска оптимального алгоритма поиска частых наборов»

Студент В.В. Жупиков
(И.О. Фамилия) (личная подпись)

Руководитель А.П. Тонких
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема бакалаврской работы: «Программный модуль для поиска оптимального алгоритма поиска частых наборов».

Бакалаврская работа посвящена сравнению алгоритмов поиска частых предметных наборов, используемых при аффинитивном анализе данных с целью формирования ассоциативных правил.

Для сравнения алгоритмов на языке Python разработано программное обеспечение, загружающее заданную пользователем выборку данных, и осуществляющее поиск частых предметных наборов с использованием различных алгоритмов. Программное обеспечение визуализирует результаты вычислительных экспериментов в виде графиков.

Цель работы – разработка программного модуля для сравнения алгоритмов поиска частых предметных наборов.

Бакалаврской работа состоит из введения, трёх глав, заключения и списка литературы.

Во введении описывается актуальность проводимого исследования, дается краткая характеристика проделанной работы.

В первой главе производиться обзор существующих алгоритмов для поиска частых наборов данных.

Во второй главе описывается методология сравнительного анализа существующих алгоритмов поиска частых наборов.

В третьей главе описывается программная реализация для сравнения эффективности работы алгоритмов.

В заключении представлены выводы по проделанной работе.

В работе присутствуют 2 таблицы, 19 рисунков, 6 формул. Список литературы состоит из 20 литературных источников. Общий объем выпускной квалификационной работы составляет 42 страниц.

Abstract

The topic of the bachelor's thesis: "A software module for finding an optimal algorithm for finding frequent sets".

Bachelor's thesis is devoted to the comparison of algorithms for searching frequent subject sets used in affine analysis of data to form associative rules.

In order to compare the algorithms, software is developed in Python that loads a user-defined sample of data and searches for frequent subject sets using different algorithms. The software visualizes the results of computational experiments in the form of graphs.

The aim of the paper is to develop a software module to compare algorithms for searching frequent subject sets.

The bachelor's thesis consists of an introduction, three chapters, a conclusion and a list of references.

The introduction describes the relevance of the research, gives a brief description of the work done.

The first chapter reviews existing algorithms for searching frequent datasets.

The second chapter describes the methodology of comparative analysis of existing algorithms for finding frequent datasets.

Chapter three describes the software implementation to compare the performance of the algorithms.

The conclusion of the work is presented in the conclusion.

There are 2 tables, 19 figures, 6 formulas in the paper. The reference list consists of 20 references. The total volume of graduate qualification work is 42 pages.

Содержание

Введение

В настоящее время Big Data становятся неотъемлемой частью личной и профессиональной жизни людей. Сегодня по-настоящему успешный бизнес уже немыслим без анализа данных, а в медицине различные инструменты Data Mining используются повсеместно.

Аффинитвный анализ активно используется розничными сетями всех стран мира для поиска взаимосвязи в подкупах, совершаемых клиентами магазина [12]. Целью аффинитивного анализа является поиск правил, описывающих закономерность в совершаемых покупках. Первое математическое описание данной задачи было предложено Rakesh Agrawal в 1993 году, и она известна под названием Market Basket Analysis [11]. «Задача подразумевает выделение нестрогих правил, по которым клиенты, приобретающие некий набор товаров, зачастую покупают что-то ещё в дополнение к имеющимся покупкам. Правила такого вида именуются ассоциативными, а вероятность, с которой выбор одних товаров влечёт выбор других, называется значимостью (confidence). Задача анализа продуктовой корзины являлась предпосылкой к появлению области ассоциативных правил, однако в дальнейшем этот инструмент начали использовать для увеличения перекрёстных продаж (cross-sell) и продаж с повышением цены (up-sell), а также для более эффективной прямой адресной рассылки рекламных предложений (direct mail)» [20].

Основная часть любого алгоритма поиска ассоциативных правил состоит в поиске часто встречающихся наборов. При условии наличия решения этой подзадачи, получение ассоциативных правил осуществляется тривиальным образом, в то время как сам поиск частых наборов вычислительно сложен. В действительности, проблема, к которой сводится поиск ассоциативных правил, является гораздо более востребованной и применяется ещё и для задач классификации и прогнозирования, например, при диагностике заболеваний сердца [8].

Исходя из значительного количества исследований в области часто встречающихся наборов и наличия множества свежих публикаций, можно с уверенностью сделать вывод об актуальности проблем, сводящихся к поиску частых наборов.

Задача существует уже более 20-ти лет, и за это время появилось огромное количество алгоритмов, решающих её с той или иной степенью эффективности. Однако сложность состоит в том, что до появления данной работы не существовало полноценного сравнительного анализа одновременно всех существующих алгоритмов поиска частых наборов в равных условиях [14]. При решении конкретной задачи важно иметь возможность осознанного выбора алгоритма с наибольшей производительностью, однако это невозможно было сделать при отсутствии полномасштабного исследования.

Цель данной работы — разработка программного модуля для сравнения алгоритмов поиска частых предметных наборов. Для достижения этой цели в рамках данной работы были сформулированы следующие задачи:

- провести теоретический анализ алгоритмов;
- разработать программное обеспечение для сравнения быстродействия алгоритмов;
- провести тестирование программного обеспечения на существующем наборе данных.

1 Анализ алгоритмов для работы с предметными наборами

1.1 Обзор технологии аффинивного анализа данных

Как уже было отмечено, алгоритмы аффинитивного анализа используются для профилирования розничных сетей. Давайте формализуем задачу профилирования, обобщив её до общего случая. Пусть имеется множество объектов с общим для всех словарём упорядоченных признаков:

$$I = \{i_1, i_2, ..., i_m\}, \quad (1)$$

при этом также существует некоторое количество наборов признаков из словаря:

$$T^n = \{t_1, t_2, ..., t_n \mid t \subset I\}, \quad (2)$$

каждый из которых описывает тот или иной объект. Тогда профилем назовём такое подмножество словаря $P \subset I$, которое наиболее ёмко характеризует данный объект, то есть достаточно часто встречается в его исходных описаниях T^n. Порог частоты встречаемости будет определяться входными данными, по умолчанию будет взято наиболее часто встречающееся подмножество словаря [1].

Поскольку задача профилирования объектов не была автоматизирована ранее, рассмотрем наиболее близкая к ней задача Market Basket Analysis, сформулированную Rakesh Agrawal в 1993 году. В этой же работе была определена область ассоциативных правил [5].

- Имея упорядоченный набор признаков $I = \{i_1, i_2, ..., i_m\}$, а также набор исходных транзакций (описаний) $T^n = \{t_1, t_2, ..., t_n \mid t \subset I\}$, для каждого набора признаков $s \subset I$ положим $s(t) = 1$, если признаки из s совместно встречаются в $t \in T^n$.

- Поддержкой (support) набора s назовём $v(s) = \frac{1}{n} \sum_{i=1}^{n} s(t_i)$

• Набор $s \subset I$ будет называться частым (frequent itemset), если его поддержка превышает минимальное пороговое значение (параметр δ), то есть $v(s) \geq \delta$.

• Ассоциативное правило (association rule) $x \to y$ — это пара непересекающихся наборов $x, y \subset I$ для которых выполняются следующие условия [10]:

1. Данная пара наборов одновременно встречается часто в исходных данных, т.е. выполняется условие (3):

$$v(x \cup y) \geq \delta. \quad (3)$$

2. Если в описании встречается *x*, то с вероятностью, превышающей σ, встречается также и *y*, т.е. выполняется условие (4):

$$v(y \mid x) = \frac{v(x \cup y)}{v(x)} \geq \sigma, \quad (4)$$

под $v(y|x)$ понимается достоверность правила (confidence) с минимальным пороговым значением σ.

«Главной и наиболее вычислительно сложной частью задачи поиска ассоциативных правил является поиск часто встречающихся наборов. Найдя их, из каждого полученного частого набора путём разбиения его на два непересекающихся поднабора можно за линейное от количества признаков время выделить ассоциативное правило, отбросив те из них, чья значимость меньше заданной минимальной σ» [4]. Таким образом, имеет смысл сравнивать только алгоритмы поиска часто встречающихся наборов. В настоящее время существует 12 алгоритмов для поиска частых предметных наборов.

Существует два основных подхода к решению данной задачи:

1. Генерация решений и тестирование (candidate-generation-and-test). В данном случае алгоритмы формируют наборы длиной *k*, основываясь на наборах длинной $k - 1$.

2. Наращивание паттернов (pattern-growth). В том случае поиск

наборов осуществляется рекурсивно. Для этого база данных делится на части, и проводится поиск локальных ответов, которые в дальнейшем наращиваются до общего результата.

1.2 Алгоритмы для работы с предметными наборами

Рассмотрим особенности алгоритма Apriori. Данный алгоритм был основателем технологии поиска ассоциативных правил (Rakesh Agrawal) и является одним из первых решений данной задачи. В основе алгоритма лежит подход candidate-generation-and-test [9].

Apriori основан на иерархическом принципе обхода в ширину для перебора всех возможных комбинаций наборов разной длины:

- множество частых наборов, состоящих из одного предмета, формируется в соответствии с 1.5:

$$G_1 \leftarrow \left\{ \{i\} \mid i \in I, v(\{i\}) \geq \delta \right\} \tag{5}$$

- множество часто встречающихся наборов длины j, генерируемое из наборов длины $(j-1)$ формируется в соответствии с 6:

$$G_j \leftarrow \left\{ s \cup \{i\} \mid s \in G_{j-1}, i \in G_1 \setminus s, v(s \cup \{i\}) \geq \delta \right\} \tag{6}$$

Для снижения количества рассматриваемых комбинаций предметов в алгоритме используется принцип антимонотонности.

Рассмотрим особенности алгоритма FP-Growth. Данный алгоритм заложил основу подхода «pattern-growth», который впоследствии был использован другими алгоритмами для поиска частых наборов.

FP-Growth, как и многие другие алгоритмы, сводит задачу поиска частых наборов к задаче хранения словаря, для решения которой в данном случае используется префиксное дерево – FP-Tree. Предварительно признаки внутри каждой транзакции сортируются по убыванию значения поддержки, их порядок фиксируется. Нечастые одноэлементные наборы отбрасываются заранее. Сама структура данных строится по следующим принципам [2]:

- корень дерева v_0 содержит значение *null;*
- в каждой вершине *v* дерева *T* задаются: признак $i_v \in I$, множество дочерних вершин $C_v \subseteq T$, поддержка $p_v = v(s_v) : s_v = \{i_u \mid u \in [v_0, v]\}$, $[v_0, v]$ – путь от корня дерева v_0 до вершины *v*;
- $V(T,i) = \{v \in T \mid i_v = i\}$ – все вершины признака *i*;
- $P(T,i) = \sum_{v \in V(T,i)} p_v$ – суммарная поддержка признака i;
- дерево подразделяется на уровни, каждый из которых соответствует какому-то признаку и каждому признаку сопоставляется единственный уровень. При этом следующая на пути вершина может находиться на любом уровне ниже текущего, так как и те, и другие упорядочены по убыванию поддержки соответствующих им элементов [8].

В процессе работы алгоритма неоднократно формируется условное FP-дерево (CFP-Tree) - это FP-Tree, построенное только по транзакция, содержащим данный признак. Пусть дано FP-дерево T и признак $i \in I$. Conditional FP-Tree $T' = T \mid i$ будет получено, если:

- оставить в дереве только вершины на путях из вершины *v* признака *i* снизу вверх до корня v_0, то есть $T' = \bigcup_{v \in V(T,i)} [v, v_0]$;
- увеличить значения поддержек p_v вершин $v \in V(T',i)$ снизу вверх по правилу $p_u = \sum_{w \in C_u} p_w$ для всех $u \in T'$;
- удалить из *T'* все вершины, соответствующие признаку *i*, так как к текущему моменту все признаки, лежащие ниже *i*, уже будут рассмотрены.

Заметим, что *T'* формируется из дерева *T* без обращения к базе транзакций.

Формирование итогового ответа происходит посредствам рекурсивной процедуры, принимающей FP-дерево *T*, набор $s \subseteq I$ и список наборов *R*. В результате работы все частые наборы, содержащие *s*, будут добавлены в *R*.

Перебираются все признаки $i \in I \,|\, V(T,i) \neq \varnothing$ по уровням снизу вверх, и если $P(T,F) \geq \delta$, то:

- $R \leftarrow R \cup \{s \cup \{i\}\}$;
- построить $T' = T \,|\, i$;
- запустить данную процедуру от новых параметров: $T', s \cup \{i\}, R$.

Проиллюстрируем вышеописанное построение FP-Tree конкретным примером (рисунок 1). Имеется некоторое количество транзакций, которые после обозначения признаков некоторыми символами (или наборами символов) преобразуются в слова.

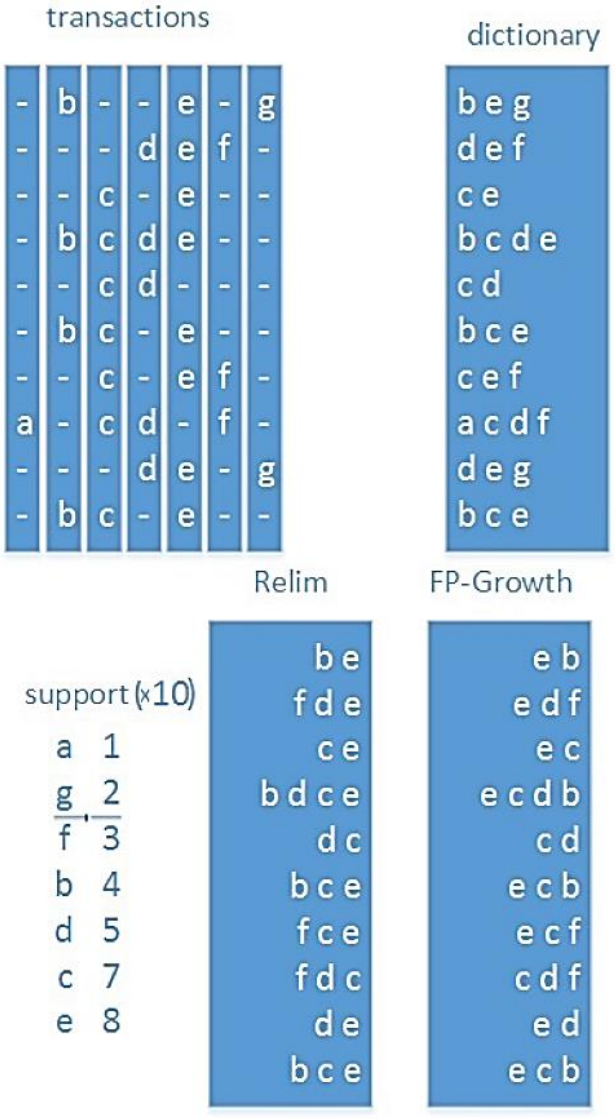


Рисунок 1 – Примеры выполнения алгоритмов FP-Growth и Relim

Предварительно подсчитывается поддержка каждого одноэлементного набора. Нечастые наборы отсеиваются, остальные упорядочиваются внутри каждой транзакции в порядке не убывания значения поддержки. Для простоты положим минимальную поддержку $\delta = 0{,}3$. За один проход по рассматриваемой базе транзакций будет построено дерево следующего вида (рисунок 2).

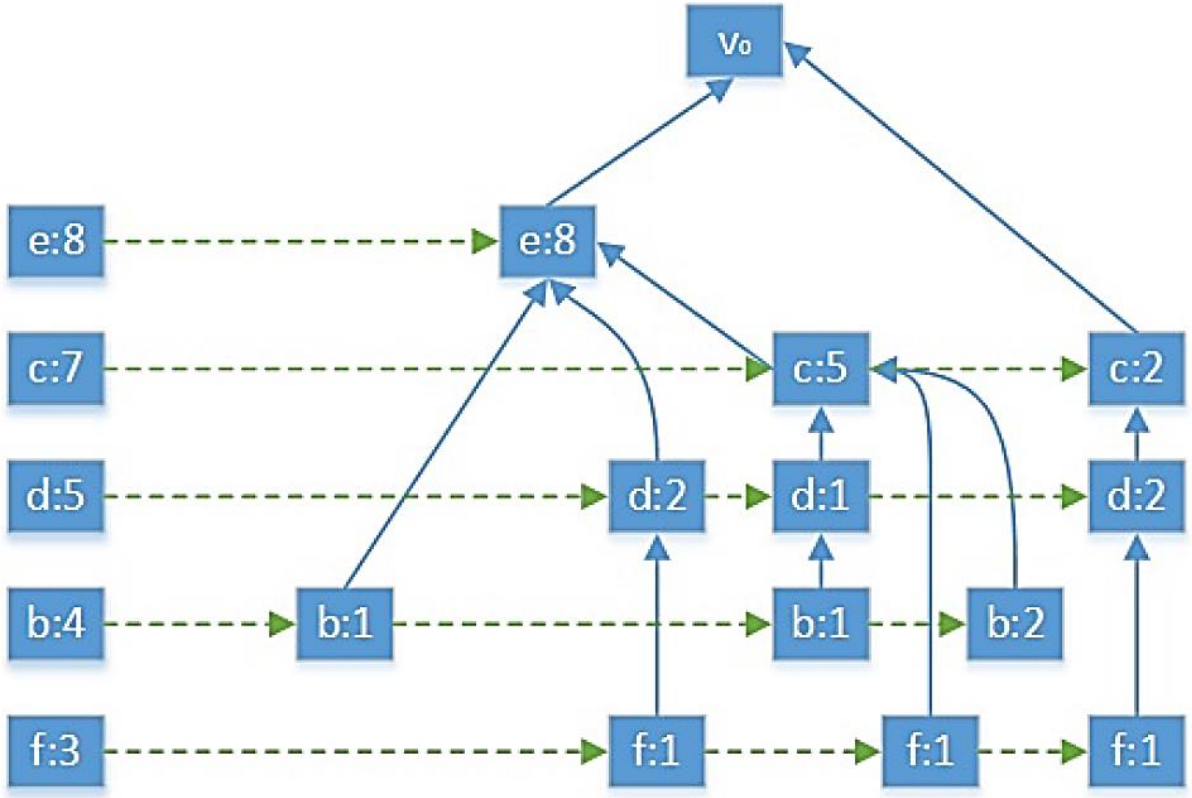


Рисунок 2 – Пример FP-дерева

Рассмотрим особенности алгоритма Relim. Данный алгоритм основан на схему рекурсивного отсеивания. Алгоритм осуществляет поиск всех частых наборов с данным префиксом, рекурсивно увеличивая его вместе с обновлением значения поддержки. Он является представителем подхода pattern-growth [13]. Основные аспекты алгоритма, проиллюстрированные на примере, представленном на рисунке 1.

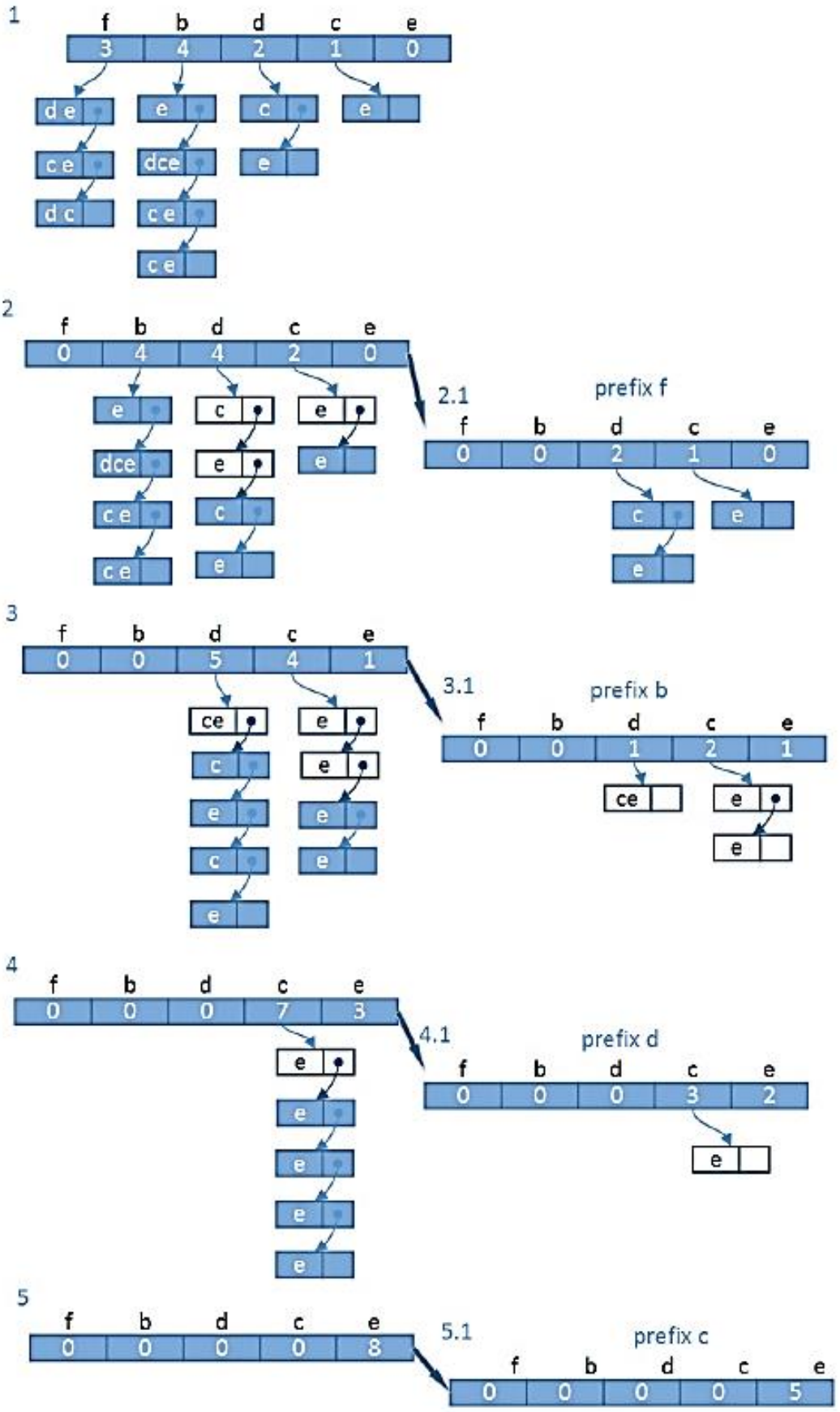


Рисунок 3 – Пример циклически изменяющейся структуры данных алгоритма Relim

Пример работы цикла обхода текущей структуры данных проиллюстрирован на рисунке 3. Здесь на каждой итерации светлым выделены перемещённые элементы списка, справа изображены заново сгенерированные массивы, от которых и будут производиться рекурсивные вызовы. На рис. 4 рассмотрено получение набора ’dc’ как часто встречающегося с поддержкой 0.3 на шаге 4.1.1 и переход к шагу 4.2, сразу после которого будет получен набор ’de’ с поддержкой 0.3.

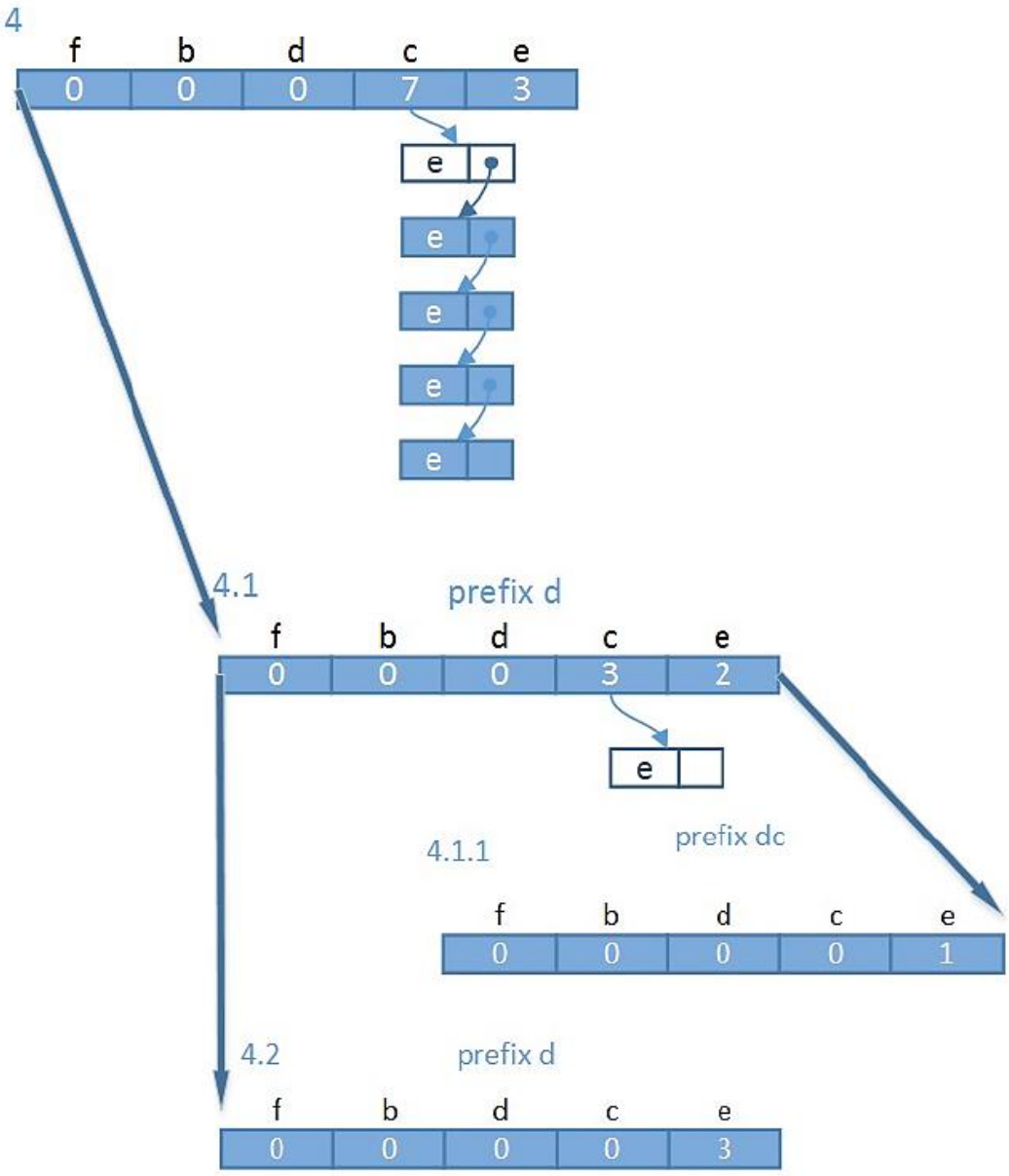


Рисунок 4 – Пример рекурсивного перехода алгоритма Relim

Рассмотрим особенности алгоритма Apriori Hybrid. Данный алгоритм основан на использовании подхода candidate-generation-and-test. Алгоритм использует структуру данных Hash-tree для хранения генерируемых кандидатов (упрощает подсчёт поддержки новых наборов). И двумерные массивы элементов числового типа для хранения часто встречающихся наборов, каждый из которых имеет свой id. По ним и происходит индексация.

Данный алгоритм работает по тем же основным принципам, что и Apriori, однако он не посещает исходную базу транзакций для подсчёта поддержки каждого рассматриваемого набора. Вместо этого используется хеширование и пересечение имеющихся наборов [16].

Рассмотрим особенности алгоритма Eclat. Данный алгоритм он основан на использовании подхода candidate-generation-and-test. При работе алгоритмы используется специальная структура данных – lattices, которая представляет собой частично уподоченные множества, в которых каждая пара элементов имеет уникальные назчения supremum и infinum.

«При работе алгоритма все кандидаты в структуре lattices, и для получения ответа алгоритм осуществляет различные обходы, основанные на поиске в ширину или в глубину. Одной из основных эвристик является разбиение наборов на подмножества для обособленного рассмотрения каждого из них. Также в процессе поиска частых наборов алгоритм Eclat пытается выделить классы эквивалентности, чтобы таким образом сузить диапазон перебора» [3].

Рассмотрим особенности алгоритма dEclat (модификацию алгоритма Elcat). Данный алгоритм основан на использовании подхода candidate-generation-and-test. При работе алгоритмы используется специальная структура данных – diffset, которая хранит в себе множества, со встроенными функциями пересечения и объединения.

Стоит отметить, что алгоритм dEclat является модификацией алгоритма Eclat тех же авторов. Основным изменением стало использование новой

структуры данных, позволяющей отсечь большее количество наборов из рассмотрения. В данном решении вновь вводится понятие эквивалентности, однако здесь оно основано на значениях некоторой функции на множестве наборов.

Рассмотрим особенности алгоритма LCMFreq. Данный алгоритм основан на использовании подхода pattern-growth. При работе алгоритма используются следующие структуры данных – бинарная матрица, префиксное дерево и массив списков.

Префиксное дерево содержит возможных кандидатов на звание частых наборов, при этом порядок признаков чётко фиксируется. Исходные транзакции хранятся в массиве списков. Бинарная матрица используется для более эффективного подсчёта поддержки. Разные версии алгоритма варьируют совместное использование этих структур данных [17].

Рассмотрим особенности алгоритма H-mine. Данный алгоритм основан на использовании подхода pattern-growth, и при работе использует структуру данных H-struct. Используемая структура данных позволяет хранить часто встречающиеся признаки вместе со ссылками на содержащие их транзакции в исходной базе данных. Способ её построения схож с FP-tree, однако вместо явного хранения транзакций алгоритм оперирует со ссылками на них. Благодаря этому H-mine превосходит многие алгоритмы по эффективности использования памяти.

Принцип работы алгоритма H-mine совпадает с FP-Growth. Отличием этих алгоритмов является выбранный подход к хранению исходных данных (что влияет на производительность алгоритма) [6].

Рассмотрим особенности алгоритма PPV. Данный алгоритм основан на использовании подхода candidate-generation-and-test. При работе алгоритма используется структура данных PPC-tree (рисунок 5). Данная структура является префиксным деревом, каждая вершина которого содержит: имя признака; счётчик, хранящий количество транзакций, лежащих на пути из корня, до данной вершины; список потомков. Для хранения рассматриваемых

наборов используется структура Node-list, состоящая из PP-кодов, извлечённых из PPC-tree.

В PPC-tree хранятся исходные транзакции. Рассматриваемые наборы содержатся в Node-list, что позволяет с быстрее подсчитывать поддержку новых наборов простым пересечениям уже имеющихся наборов. Это происходит за линейное время.

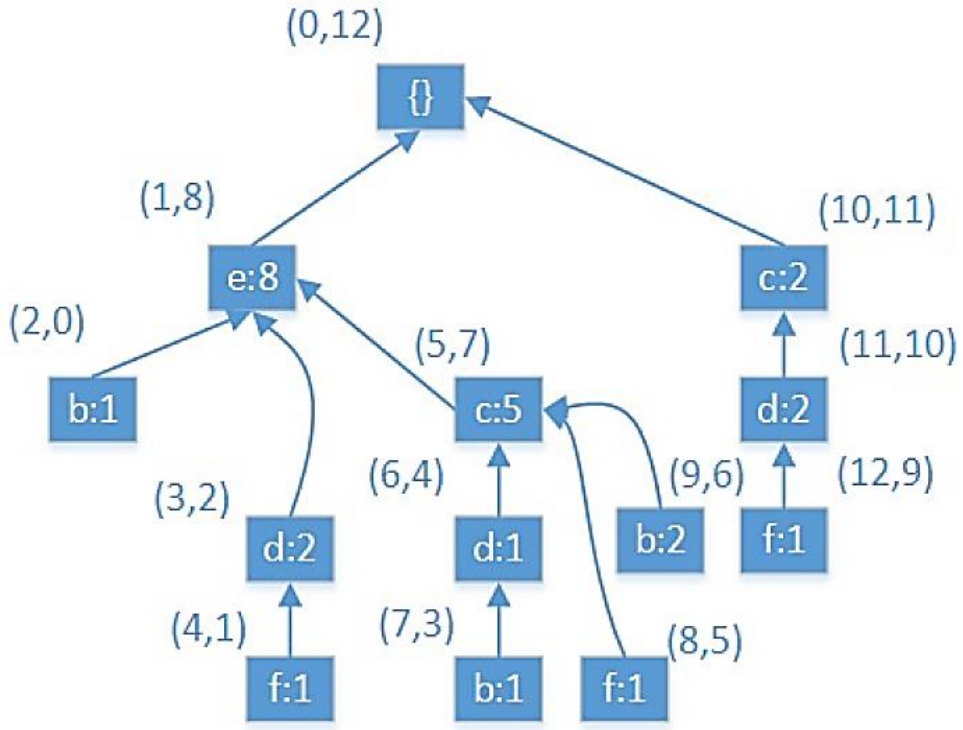


Рисунок 5 – Префиксное дерево PPC-tree

Рассмотрим особенности алгоритма PrePost. Данный алгоритм основан на использовании подхода candidate-generation-and-test. При работе алгоритма так же, как и в предыдущем случае, используется префиксное дерево PPC-tree. При переборе наборов используется структура N-list (не путать с Node-list).

Алгоритмы PrePost и PrePost+ отличаются только способом отсечения подмножеств: первый использует свойство единственности пути в N-list, а второй основан на свойстве наборов с одинаковой поддержкой [7].

Таблица 1 – Результаты сравнительного анализа алгоритмов поиска частых предметных наборов.

Название алгоритма	**Применяемый подход**	**Используемые структуры данных**
Apriori	«candidate-generation-and-test»	Массив списков array list
FP-Growth	«pattern-growth»	Префиксное дерево FP-Tree
Relim	«pattern-growth»	Массив списков array list
PrePost+	«candidate-generation-and-test»	Список N-list, префиксное дерево PPC-tree и дерево Set-enumeration tree
Apriori Hybrid	«candidate-generation-and-test»	дерево Hash-tree
Eclat	«candidate-generation-and-test»	Структура данных Lattices
dEclat	«candidate-generation-and-test»	Множество diffset
LCMFreq	«pattern-growth»	Бинарная матрица Bitmap, префиксное дерево prefix tree и массив списков array list
H-mine	«pattern-growth»	Структура данных H-struct
PPV	«candidate-generation-and-test»	Список Node-list, префиксное дерево PPC-tree
PrePost	«candidate-generation-and-test »	Список N-list, префиксное дерево PPC-tree
FIN	«candidate-generation-and-test »	Множество Node-set, префиксное дерево POC-tree, дерево Set-enumeration tree

Рассмотрим особенности алгоритма FIN. Данный алгоритм был предложен в 2014, и он основан на использовании подхода candidate-generation-and-test. При своей работе алгоритм использует структуру данных Node-set. «Это структура аналогичная N-list и Node-list, но использующая

вдвое меньший объём памяти, так как она требует хранения только одного параметра pre-order либо post-order вместо обоих сразу» [19]. Также при работе алгоритма используется структуры POC-tree и Set-enumeration tree. Первая структура является префиксным деревом, аналогичным PPC-tree, но не хранящее параметр post-order. А вторая структура – дерево, содержащее все возможные наборы признаков, упорядоченные в порядке не убывания поддержки. Каждая вершина хранит единственный часто встречающийся признак [18].

Объединим результаты анализа алгоритмов поиска частых предметных наборов в таблицу (таблица 1).

Выводы по главе

На основе исследований был проведен сравнительный анализ алгоритмов поиска частных предметных наборов. Установлено, что существует два принципа работы алгоритмов: Генерация решений и тестирование (candidate-generation-and-test) и Наращивание паттернов (pattern-growth). Алгоритмы друг от друга также отличаются используемыми структурами данных для хранения исходных данных и промежуточных результатов работы алгоритмов.

2 Методология сравнительного анализа алгоритмов поиска частых предметных наборов

2.1 Анализ критериев для выбора оптимального алгоритма

Нам необходимо сравнить работу 12 алгоритмов поиска частых предметных наборов. Среди них 8 алгоритмов, основанных на подходе «генерация решений и тестирование» (candidate-generation-and-test), к которым относятся: Apriori, Apriori Hybrid, PrePost, PrePost+, Eclat, dEclat, PPV и FIN. А также 4 алгоритма, основанных на подходе «наращивание паттернов (pattern-growth)», среди которых: FP-Growth, Relim, LCMFreq и H-mine.

Выделим критерии, по которым алгоритмы отличаются друг от друга при анализе данных:

- асимптотика алгоритма;
- простота/сложность реализации;
- объем затрачиваемой памяти при выполнении расчетов;
- время поиска решений.

Рассмотрим критерий асимптотика алгоритма. В теории алгоритмов этот показатель играет важную роль, однако в нашем случае его использование нецелесообразно. Дело в том, что все алгоритмы пользуются некоторыми эвристиками, будь то отсечения перебора или попытки сокращения области поиска. Их эффективность напрямую зависит от конкретных данных, их плотности и равномерности распределения. В подобных случаях иногда рассматривают худший случай и выделяют порядок количества операций исходя из него, но такая ситуация возникает только на искусственных специально подобранных данных, которые не встречаются в реальных задачах.

Рассмотрим критерий простота реализации. Сам по себе данный критерий является достаточно субъективным. Этот факт находит

подтверждения даже в истории IT-индустрии, ссылаясь на тот период, когда размер жалования инженеров вычисляли, исходя из сложности их работы. Были попытки подсчёта количества строк, времени, отводимого на написание кода, но ни одна из них не принесла желаемых результатов. Поскольку всё зависит от конкретного специалиста, от манеры работы, а не только от самой задачи, критерии простоты сильно размыты. Например, в нашем случае вместо написания алгоритмов используются библиотеки с их готовыми реализациями. Поэтому, можно считать, что сложность реализации алгоритмов приблизительно равная.

Рассмотрим критерий простота реализации Объём затрачиваемой памяти. Затрачиваемая память зависит от сложности структур данных применяемых в алгоритме для поиска частых предметных наборов. При создании большого количества вспомогательных структур увеличивается объем затрачиваемой памяти. Однако дополнительные структуры данных создаются для того чтобы увеличить быстродействие алгоритма. Поэтому наиболее целесообразным решение будет являться сравнение алгоритмов по скорости поиска решений.

Рассмотрим критерий время поиска решений. Поскольку ключевой фактор - это применение решения к реальным задачам, наиболее важным является именно практический результат, а именно время работы алгоритма. Этот параметр и будет играть главную роль в сравнительном анализе.

Также стоит отметить, что существуют исследования, в которых выполнено сравнение алгоритмов поиска частых наборах на синтетических данных. Для получения синтетических данных генерируется прямоугольная матрица, в которой количество строк соответствует количеству транзакций в наборе данных, а каждый столбец соответствует своему уникальному предмету. Компоненты матрицы принимают значение либо 0, либо 1. Значение компоненты равное 1 на пересечении определённой строки и столбца означает, что данный предмет присутствует в данной транзакции, а

значение 0 – что отсутствует. Данная матрица, для получения синтетических данных, заполняется с случайным образом с помощью функции random().

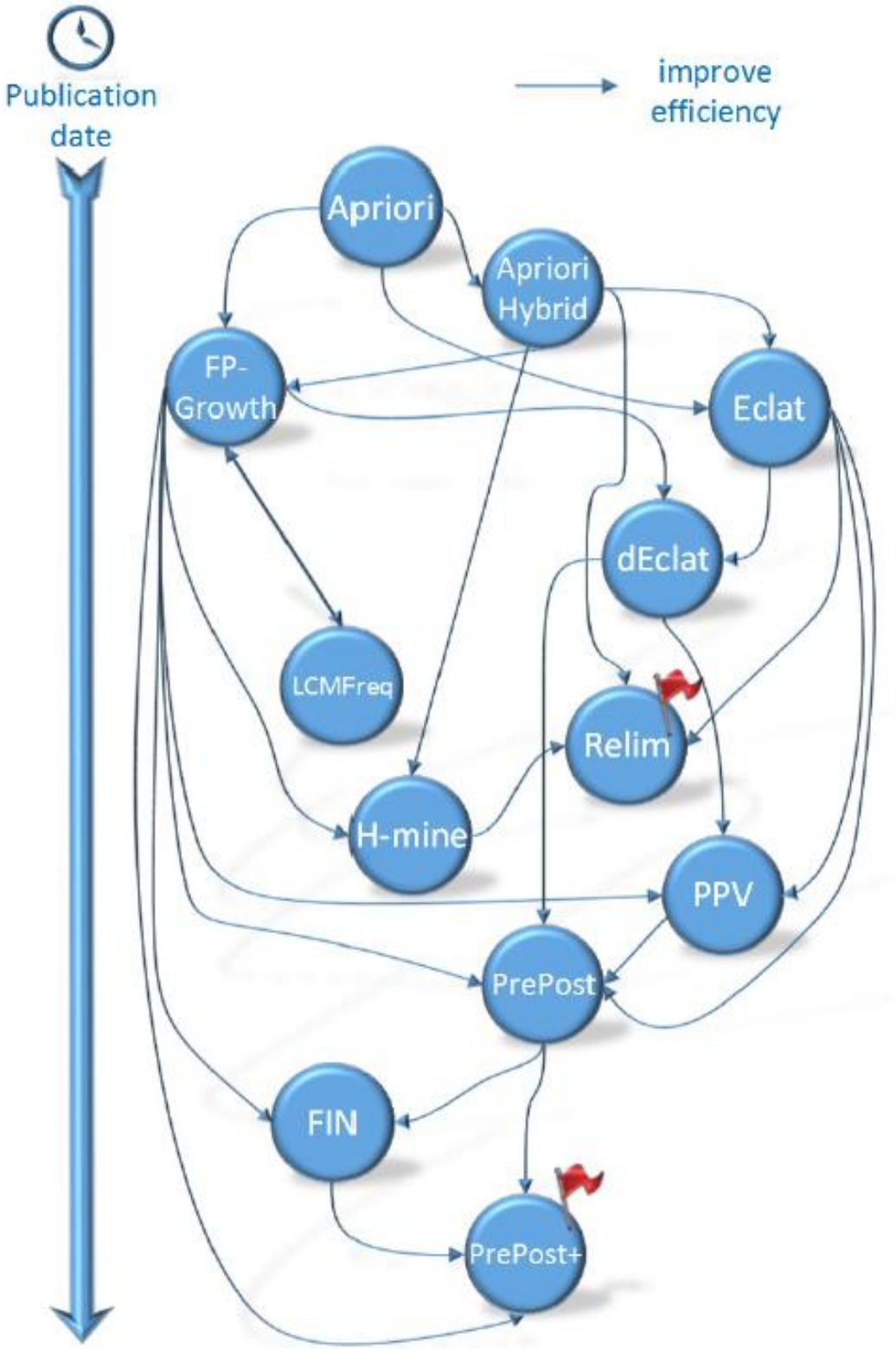


Рисунок 6 – Результат попарного сравнения алгоритмов по производительности (скорости поиска решения) на синтетических данных

Для сопоставления алгоритмов используется метод попарного сравнения. Граф результатов попарного сравнения алгоритмов производительности алгоритмов на синтетических данных показан на рисунке 6. Ребра графа, в данном случае, ориентированы от медленных алгоритмов к быстрым.

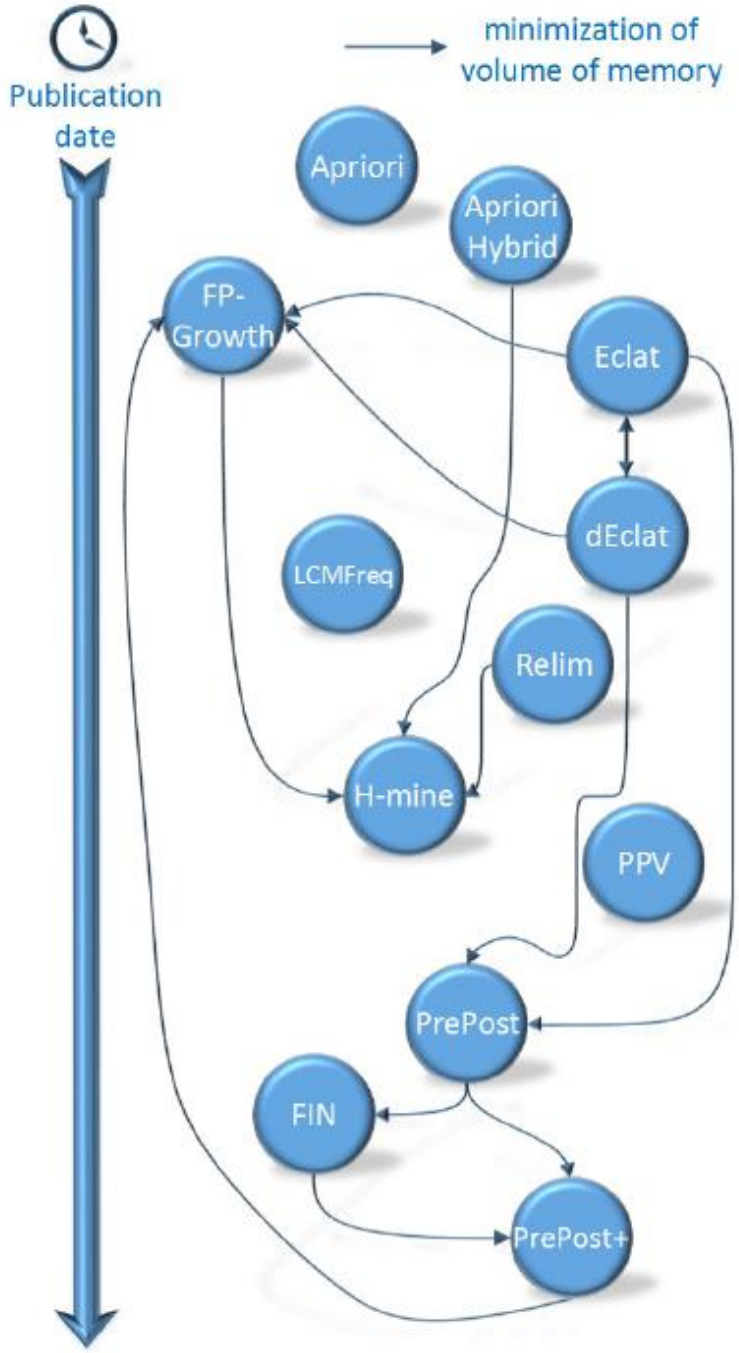


Рисунок 7 – Результат попарного сравнения алгоритмов по объему используемой памяти на синтетических алгоритмах

Аналогичный граф, построенный по итогам попарного сравнения алгоритмов, по объему используемой памяти показан на рисунке 7. Важно отметить, что оба графа построенные на синтетических данных, не гарантируют получение таких же результатов сравнения на реальных наборах данных.

2.2 Выборка данных для анализа

Для сравнения алгоритмов будет использован набор данных «Store data» размещенный в открытом репозитории по адресу https://archive.ics.uci.edu/ml/datasets.php на сайте цента машинного обучения и интеллектуальных систем (Center for Machine Learning and Intelligent Systems).

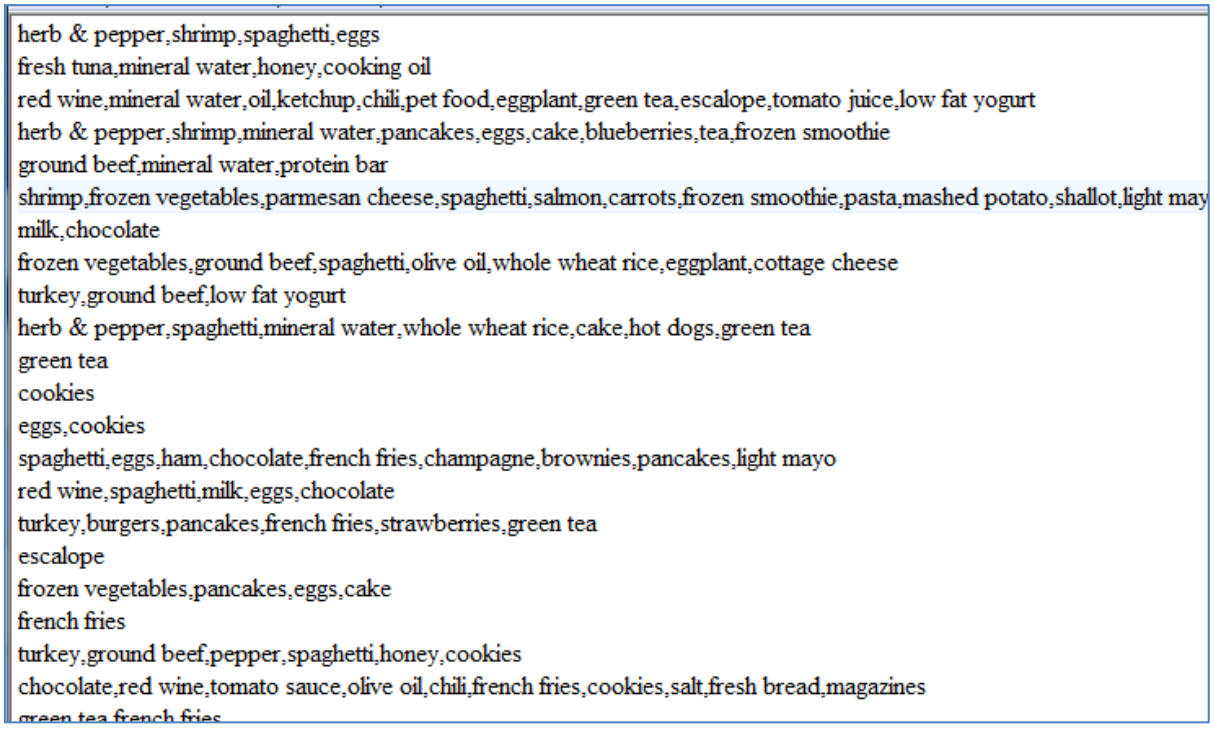


Рисунок 8 – Пример содержимого файла store_data.csv

Данная выборка данных содержит в себе информации о товарах супермаркета купленных клиентами магазина. Транзакции составлены на

основе выданных чеков. Информация в наборе данных храниться в обезличенном виде. Для товаров указана только их категория без указания производителя и марки.

Выборка данных «Store data» представляет собой текстовый файл store_data.csv, в котором предметы перечислены через запятую. Предметы одной транзакции находятся на одной строке. Пример содержимого файла представлен на рисунке 8.

Данный файл содержит себе информацию о 7501 транзакции (о товарах купленных одновременно на основе информации из 7501 чека магазина). Количество уникальных предметов – 20 штук.

Выводы по главе

Произведен анализ критериев для определения оптимального алгоритма поиска частых предметных наборов. Рассматривались такие критерии, как асимптотика, простота реализации, объем затрачиваемой памяти и время поиска решения. В ходе анализа установлено, что время поиска решения является ключевым фактором при выборе оптимального алгоритма.

Также описана выборка данных, на которых будет проводиться тестирование алгоритмов поиска частых предметных наборов.

3 Разработка программного модуля для сравнения алгоритмов

3.1 Описание программного модуля

Для сравнения алгоритмов поиска частых предметных наборов на языке Python было разработано аналитическое программное обеспечение, выполняющее тестирование быстродействия алгоритмов на выбранном пользователем наборе данных. Перечислим ключевые особенности программного обеспечения:

- возможность экспорта тестовых данных путем загрузки csv-файла
- визуализация загруженных исходных данных путем представления их в виде таблицы
- планирования вычислительных экспериментов при тестировании алгоритмов путем задания диапазона и шага изменения порогового уровня поддержки;
- представление результатов экспериментов с замерами времени поиска решений в виде таблицы
- визуализация результатов вычислительных экспериментов в виде графика
- наличия графического интерфейса, реализовано посредством виджетов.

При реализации программного обеспечения использовались следующие библиотеки:

- библиотека numpy, реализующая методы по работе с множествами (подсчет мощности множества, сортировка множеста и т.д.);
- библиотека matplotlib, предназначенная для построения графиков различных видов;

- библиотека pandas, содержащая методы для представления данных в виде таблицы;
- библиотека time для проведения временных замеров во время работы алгоритмов;
- набор библиотек, содержащие реализации алгоритмов Apriori, Apriori Hybrid, PrePost, PrePost+, Eclat, dEclat, PPV, FIN, FP-Growth, Relim, LCMFreq и H-mine.

Программный код для подключения основных библиотек к проекту представлен на рисунке 9. Остальные перечисленные библиотеки подключаются в коде ниже, по мере надобности.

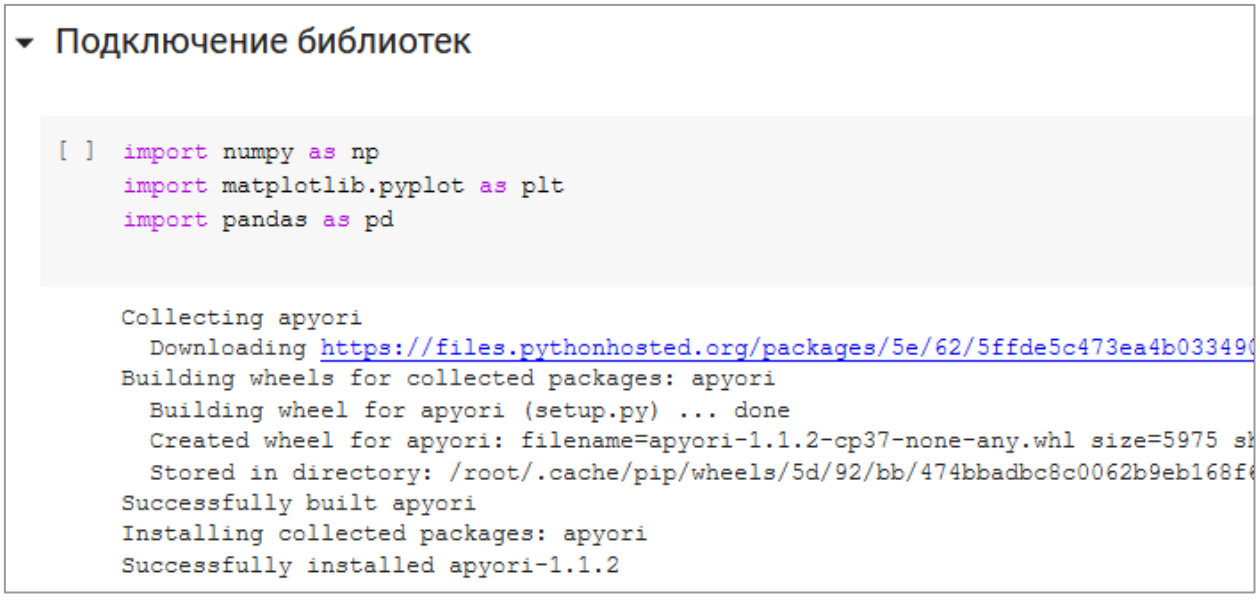


Рисунок 9 – Программный код для подключения библиотек

.На первом шаге работы программы пользователю предлагается загрузить файл с исходными данными. Для этого реализован графический интерфейс с применением стандартных виджетов. В данном случае применяется виджет, вызываемый методом files.upload(), из библиотеки google.colab. Данный метод создает кнопку, при нажатии на которую откроется стандартное диалоговое окно с выбором файла для загрузки. Выбранный файл сразу загружается в память. За процессом загрузки файла

можно следить по информации, появляющейся под кнопкой «Обзор» (рисунок 10).

Рисунок 10 – Программный код создания виджета для загрузки исходной выборки данных в виде csv-файла

Единственным входным параметром при работе алгоритмов поиска частых предметных наборов является – порог минимальной поддержки. Чем меньше порог поддержки, тем больше частых будет предметных наборов, и наоборот, чем выше уровень поддержки – тем меньше предметных наборов, удовлетворяющих данному уровню. Например, уровень порога поддержки 100% означает, что для того, чтобы предметный набор назывался частым, он должен присутствовать во всех транзакциях исходных данных.

В рамках вычислительных экспериментов требуется тестировать алгоритмы при различных значениях порога поддержки. Поэтому в программном обеспечении предусмотрено задание диапазона и шага изменения порога минимальной поддержки для частых наборов данных.

Программный код для отображения двух слайдеров и числового поля показан на рисунке 11. Переменная minlevel_min_support – минимальное значение порога поддержки, переменная maxlevel_min_support – максимальное значение порога поддержки, step – шаг изменения порога поддержки при проведении вычислительных экспериментов.

```
[ ]  #@title Задание парметров
     minlevel_min_support = 70 #@param {type:"slider", min:50, max:90, step:1}
     maxlevel_min_support = 90 #@param {type:"slider", min:50, max:90, step:1}

     step = 10 #@param {type:"integer"}
```

Рисунок 11 – Программный код для создания элементов управления, задающих диапазон и шаг изменения порога минимальной поддержки для частых наборов данных

Результат отображения элементов управления порогом уровня поддержки показан на рисунке 12.

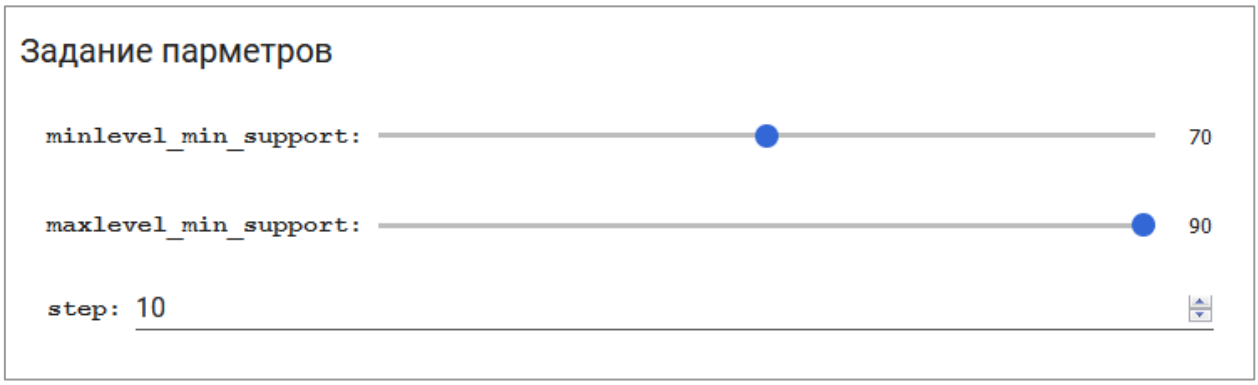
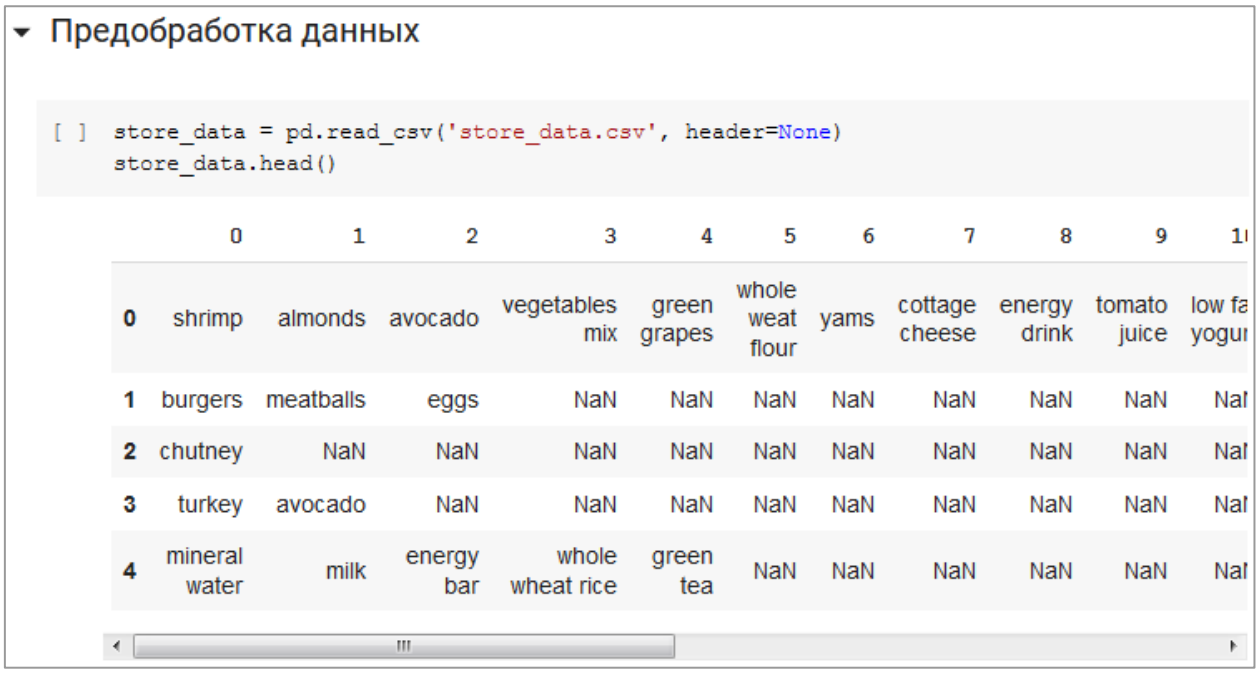


Рисунок 12 – Внешний вид элементов управления для задания диапазона и шага изменения порога минимальной поддержки

Для визуального контроля загрузки файла с данными он отображается в виде таблицы посредством метода pd.read_csv(), реализуемого библиотекой Pandas. В качестве параметров методу передается имя загруженного файла и, при необходимости, названия столбцов.

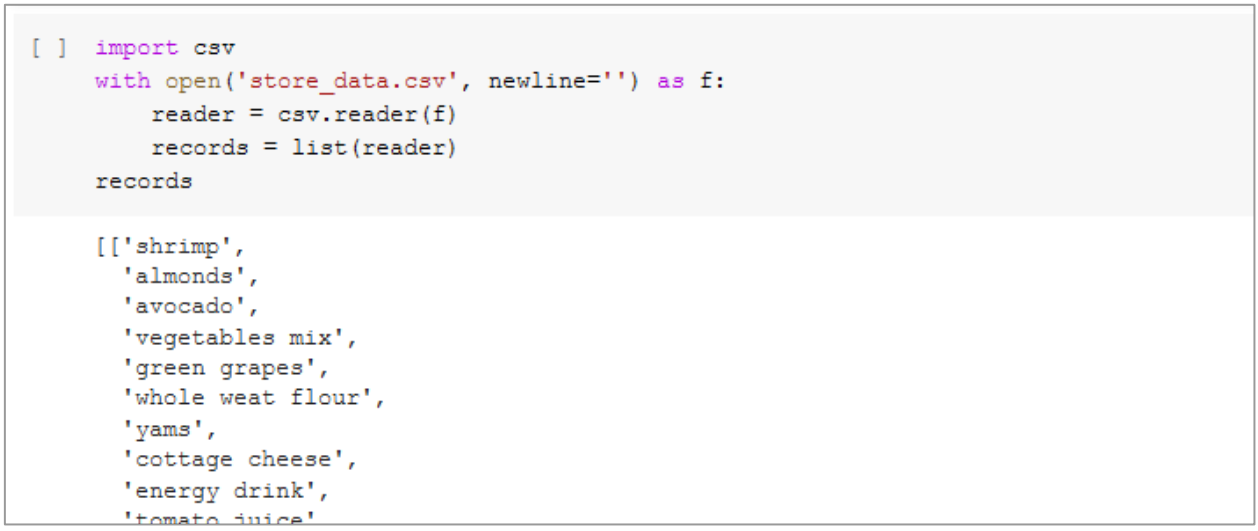
Для экономии экранного места выводится только первые 5 транзакций (из 7501), содержащихся в файле store_data.csv. Результат вывода загруженных исходных данных в виде таблицы показан на рисунке 13.



	0	1	2	3	4	5	6	7	8	9	1
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fa yogu
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	Na

Рисунок 13 – Вывод загруженных исходных данных в виде таблицы

Однако алгоритмы поиска частых предметных наборов данных требуют представления данных в своем формате, в виде – множества множеств (или массива с вложенными массивами переменной длины). Поэтому, для преобразования нужный формат, осуществляем построчное считывание данных из файла, затем конвертируем каждую строку в массив (рисунок 14).



```
[ ] import csv
    with open('store_data.csv', newline='') as f:
        reader = csv.reader(f)
        records = list(reader)
    records
```

```
[['shrimp',
  'almonds',
  'avocado',
  'vegetables mix',
  'green grapes',
  'whole weat flour',
  'yams',
  'cottage cheese',
  'energy drink',
  'tomato juice'
```

Рисунок 14 – Построчное чтение файла с исходными данными для формирования множества (массива) транзакций

Теперь, когда наше множество транзакций хранится в правильном формате (в переменной records) можно переходить к выполнению вычислительных экспериментов.

Вычислительные эксперименты проводятся последовательно для каждого алгоритма поиска частых предметных наборов. Сначала для каждого алгоритма создается пустой массив, куда будут помещаться замеры времени поиска предметных наборов. Затем, с помощью цикла for осуществляется перебор всех значения порогового уровня поддержки от minlevel_min_support до maxlevel_min_support с шагом step. На каждой итерации цикла производиться съем текущей временной метки с помощью метода time.time(). Затем запускается выполнения алгоритма. После выполнения завершения алгоритма проводится повторный съем текущей временной метки (рисунок 15).

Разница между временными метками помещается в массив (для алгоритма apriori массив храниться в переменной time_apriori). После этого осуществляется переход к выполнению следующей итерации цикла for. Как только выполнение текущего цикла завершено запускается цикла для следующего алгоритма.

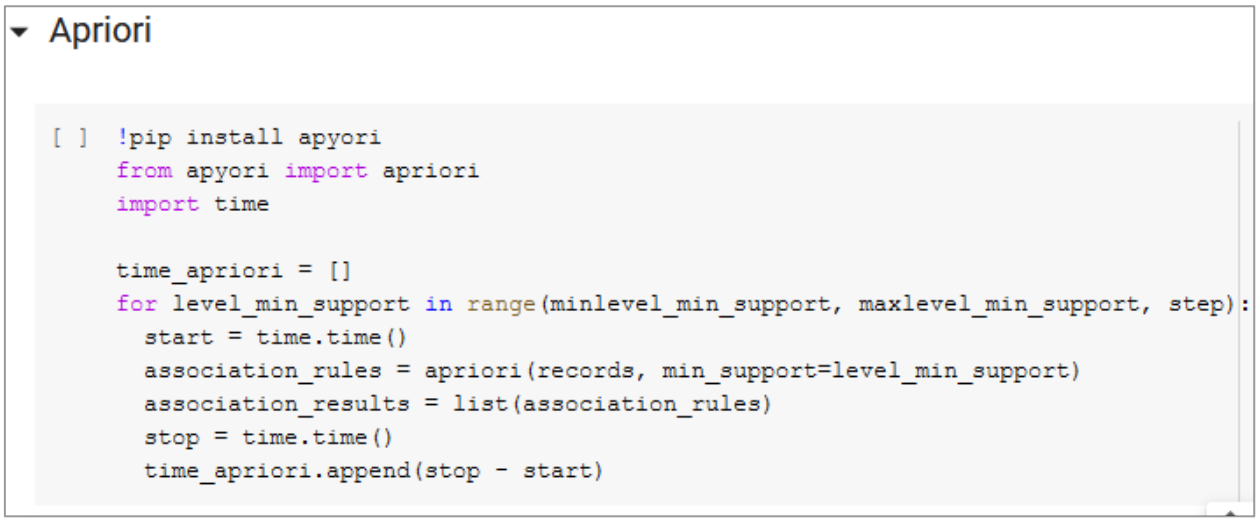


Рисунок 15 – Программный код для тестирования скорости работы алгоритма Apriori

Реализована возможность вывода ассоциативных правил, сгенерированных на основе частых предметных наборов (рисунок 16).

```
df_rules.sort_values(by=['SxC', 'Lift'])
```

	RuleL	RuleR
3	['herb & pepper']	['ground beef']
19	['shrimp', 'ground beef']	['spaghetti']
8	['cooking oil', 'ground beef']	['spaghetti']
9	['frozen vegetables', 'spaghetti']	['ground beef']
16	['herb & pepper', 'mineral water']	['ground beef']
17	['herb & pepper', 'spaghetti']	['ground beef']
11	['shrimp', 'mineral water']	['frozen vegetables']
2	['pasta']	['escalope']
5	['whole wheat pasta']	['olive oil']
4	['tomato sauce']	['ground beef']
1	['mushroom cream sauce']	['escalope']
15	['spaghetti', 'grated cheese']	['ground beef']
6	['pasta']	['shrimp']
14	['frozen vegetables', 'spaghetti']	['tomatoes']

Рисунок 16 – Вывод ассоциативных правил сгенерированных на основе частых предметных наборов

Как только циклы вычислительных экспериментов для всех алгоритмов (Apriori, Apriori Hybrid, PrePost, PrePost+, Eclat, dEclat, PPV, FIN, FP-Growth, Relim, LCMFreq и H-mine) завершаться, программа перейдет к объединению полученных результатов в сводную таблицу.

Программный код для объединения результатов вычислительного эксперимента по тестирования алгоритмов в сводную таблицу показан на рисунке 17.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.array([time_Apriori, time_AprioriH, time_Eclat, time_PrePost2,
                            time_LCMFreq, time_H_mine, time_PrePost, time_FIN,
                            time_dEclat, time_Relim, time_FP_Growth, time_PPV]),
                  columns=['70%','75%','80%','85%', '90%'],
                  index=['Apriori','AprioriH','Eclat','PrePost+','LCMFreq','H-mine',
                         'PrePost', 'FIN', 'dEclat', 'Relim', 'FP-Growth', 'PPV'])
df
```

	70%	75%	80%	85%	90%
Apriori	88.34	84.58	79.01	78.91	75.40
AprioriH	87.44	83.84	78.56	78.89	75.18
Eclat	88.06	83.21	78.24	78.05	74.05
PrePost+	78.82	73.58	68.22	67.54	64.78
LCMFreq	85.84	81.81	76.52	76.03	72.49
H-mine	84.08	78.78	74.96	72.05	70.08
PrePost	83.88	77.89	74.22	70.91	67.22
FIN	82.58	75.39	72.38	69.89	67.23
dEclat	85.32	80.44	75.96	74.62	71.05
Relim	80.72	74.06	70.37	67.81	64.82
FP-Growth	85.42	81.68	76.20	75.10	71.61
PPV	83.36	77.32	73.66	70.55	69.01

Рисунок 17 – Программное объединение результатов вычислительного эксперимента по тестирования алгоритмов в таблицу (значения таблицы – время работы алгоритма в секундах)

Как видно из рисунка 17 табличное представление результатов вычислительных экспериментов не обладает требуемой наглядностью. Поэтому в программе реализована возможность визуализации полученных результатов в виде графика. Построение графиков осуществляется с использованием библиотеки matplotlib. Для этого с помощью задания параметра figure.figsize задается размер изображения с графиками. И затем объекту plt с помощью метода plot() последовательно предаются данные с результатами экспериментов по каждому алгоритму.

```
[11] import matplotlib.pyplot as plt

     x = np.array([0,1,2,3,4])
     my_xticks = ['70%','75%','80%','85%', '90%']
     plt.rcParams['figure.figsize'] = [10, 8]
     plt.xticks(x, my_xticks)
     plt.title("Сравнение скорости поиска решения")
     plt.xlabel("Уровень минимальной поддержки")
     plt.ylabel("Время поиска решения в с")
     plt.plot(x, time_Apriori, label = 'Apriori')
     plt.plot(x, time_AprioriH, label = 'AprioriH')
     plt.plot(x, time_Eclat, label = 'Eclat')
     plt.plot(x, time_PrePost2, label = 'PrePost+')
     plt.plot(x, time_LCMFreq, label = 'LCMFreq')
     plt.plot(x, time_H_mine, label = 'H-mine')
     plt.plot(x, time_PrePost, label = 'PrePost')
     plt.plot(x, time_FIN, label = 'FIN')
     plt.plot(x, time_dEclat, label = 'dEclat')
     plt.plot(x, time_Relim, label = 'Relim')
     plt.plot(x, time_FP_Growth, label = 'FP-Growth')
     plt.plot(x, time_PPV, label = 'PPV')

     plt.legend()
     plt.show()
```

Рисунок 18 – Программный код для визуализации данных вычислительных экспериментов в виде таблицы

Дополнительно с помощью методов xlabel() и ylabel() задаются обозначения осей, а с помощью метода title() задается название отображаемое над графиком. Программный код для визуализации данных вычислительных экспериментов в виде таблицы показан на рисунке 18.

Результатом выполнения данного программного кода будет являться график, показанный на рисунке 19.

3.2 Результаты тестирования алгоритмов

С помощью разработанного программного обеспечения на наборе данных «Store data» были протестированы 12 алгоритмов поиска частых предметных наборов. При проведении вычислительных экспериментов производилось измерение времени, затрачиваемое алгоритмом на поиск всех частных наборов. Вычислительные эксперименты повторялись для одного и того же набора данных при разных значениях порога уровня поддержки (от 70% до 90% шагом 5%). Полученные результаты экспериментов показаны в таблице 2.

Таблица 2 – Результаты вычислительных экспериментов по замеру времени (время указано в секндах) поиска решений в секундах на наборе данных «Store data»

Название алгоритма	Порог минимальной поддержки в %				
	70%	75%	80%	85%	90%
Apriori	88.34	84.58	79.01	78.91	75.40
Apriori Hybrid	87.44	83.84	78.56	78.89	75.18
Eclat	88.06	83.21	78.24	78.05	74.05
PrePost+	78.82	73.58	68.22	67.54	64.78
LCMFreq	85.84	81.81	76.52	76.03	72.49
H-mine	84.08	78.78	74.96	72.05	70.08
PrePost	83.88	77.89	74.22	70.91	67.22
FIN	82.58	75.39	72.38	69.89	67.23
dEclat	85.32	80.44	75.96	74.62	71.05
Relim	80.72	74.06	70.37	67.81	64.82
FP-Growth	85.42	81.68	76.20	75.10	71.61
PPV	83.36	77.32	73.66	70.55	69.01

Для наглядности полученных результатов отобразим график, построенный с помощью разработанного программного обеспечения (рисунок 19)

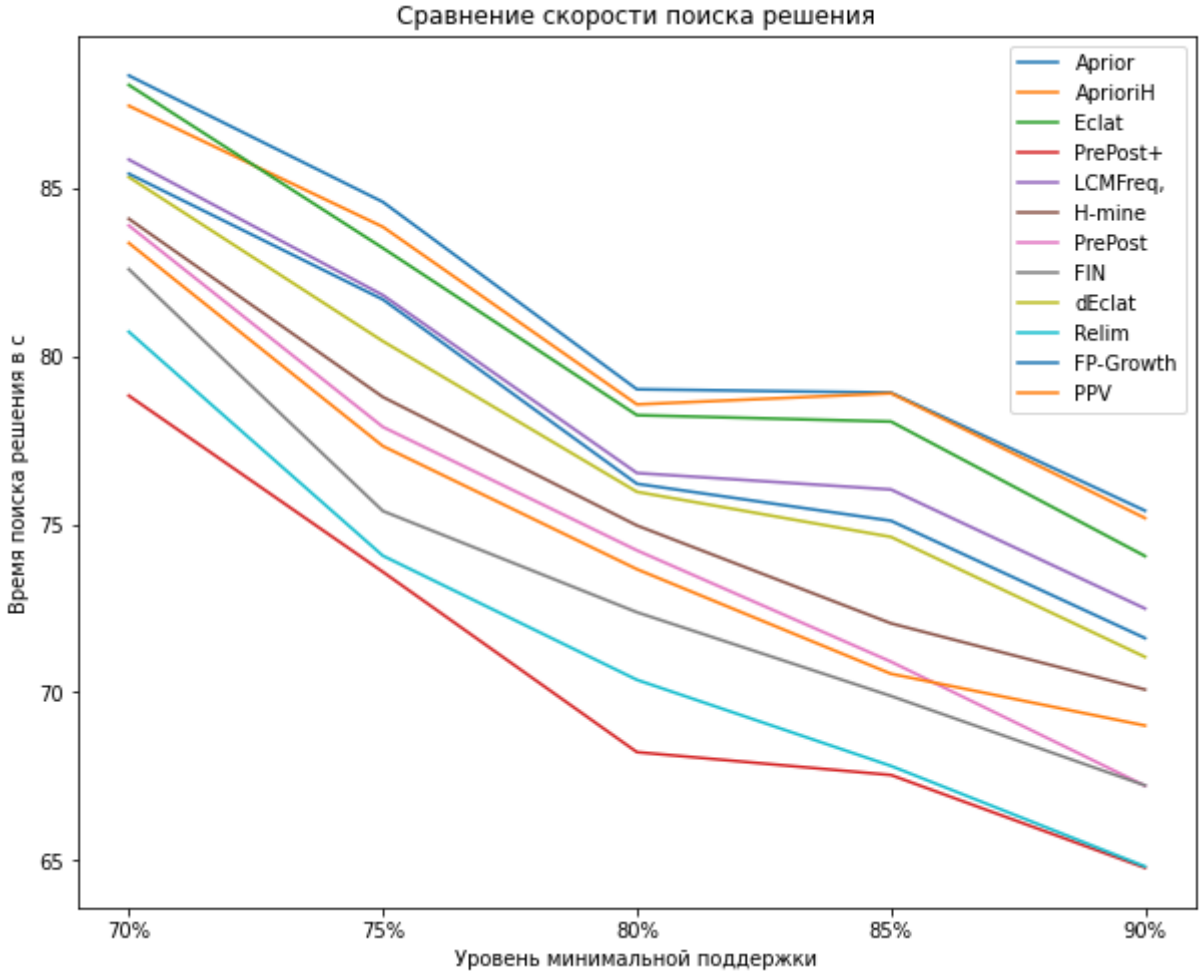


Рисунок 19 – Визуализация результатов вычислительного эксперимента

Как видно из графиков, чем меньше пороговое значение поддержки, тем больше времени требуется алгоритму для поиска всех частых предметных наборов. Это связано с тем, что чем меньше пороговый уровень поддержки, тем больше наборов соответствует критерию.

Из всех алгоритмов наибольшую скорость работы показал PrePost+. Следует отметить, что для другого набора данных лучший алгоритм следует определять отдельно. Но благодаря разработанному программному обеспечению этот процесс автоматизирован.

Выводы по главе

На языке программирование Python было разработано программное обеспечение, производящее сравнение быстродействия алгоритмов поиска частых предметных наборов на загруженных пользователем данных. Реализована поддержка следующих алгоритмов: Apriori, Apriori Hybrid, PrePost, PrePost+, Eclat, dEclat, PPV, FIN, FP-Growth, Relim, LCMFreq и H-mine. В программном обеспечении реализована возможность визуализации результатов вычислительного эксперимента по замеру быстродействия алгоритмов в виде графиков. Программное обеспечение работает на облачном сервере Google colab.

Работа программного обеспечения была протестирована на данных о покупке товаров в супермаркете. На имеющихся данных наилучшие результаты по быстродействию показал алгоритм PrePost+, а наихудшие – алгоритм Apriori. По результатам вычислительных экспериментов построены графики времени поиска алгоритмами решения в зависимости от уровня минимальной поддержки.

Заключение

Выделим основные результаты работы:

1. На основе исследований был проведен сравнительный анализ алгоритмов поиска частных предметных наборов. Установлено, что существует два принципа работы алгоритмов: Генерация решений и тестирование (candidate-generation-and-test) и Наращивание паттернов (pattern-growth). Алгоритмы друг от друга также отличаются используемыми структурами данных для хранения исходных данных и промежуточных результатов работы алгоритмов.

2. Произведен анализ критериев для определения оптимального алгоритма поиска частых предметных наборов. Рассматривались такие критерии, как асимптотика, простота реализации, объем затрачиваемой памяти и время поиска решения. В ходе анализа установлено, что время поиска решения является ключевым фактором при выборе оптимального алгоритма.

3. Описана выборка данных, на которых проводится тестирование алгоритмов поиска частых предметных наборов. Результаты тестирования и графики приведены в третье главе.

4. На языке программирование Python было разработано программное обеспечение, производящее сравнение быстродействия алгоритмов поиска частых предметных наборов на загруженных пользователем данных. Реализована поддержка следующих алгоритмов: Apriori, Apriori Hybrid, PrePost, PrePost+, Eclat, dEclat, PPV, FIN, FP-Growth, Relim, LCMFreq и H-mine. В программном обеспечении реализована возможность визуализации результатов вычислительного эксперимента по замеру быстродействия алгоритмов в виде графиков. Программное обеспечение работает на облачном сервере Google colab.

Список используемой литературы

1. Жуков, Д.А. Формирование контрольных выборок при технической диагностике объекта с применением машинного обучения / Д.А. Жуков, А.С. Хорева, Ю.Е. Кувайскова, В.Н. Клячкин // Математические методы и модели: теория, приложения и роль в образовании – международная научно-техническая конференция : сборник научных трудов, 28–30 апреля 2016 года. – Ульяновский государственный технический университет (Ульяновск), 2016. – с. 44-48. – Текст : непосредственный.

2. Иванников Ю.Ю. Применение методов машинного обучения для выявления бот-трафика среди запросов к веб-приложению / Ю.Ю. Иванников, Е.Ю. Митрофанова // Сборник студенческих научных работ факультета компьютерных наук ВГУ, Факультет компьютерных наук, 2017. – ФГБОУ ВО «Воронежский государственный университет», 2017. – с. 119-123. – Текст : непосредственный.

3. Клячин В.Н. Использование агрегированных классификаторов при технической диагностике на базе машинного обучения / В.Н. Клячин, Ю.Е. Кувайскова, Д.А. Жуков // Информационные технологии и нанотехнологии (ИТНТ-2017) – сборник трудов III международной конференции и молодежной школы. Самарский национальный исследовательский университет имени академика С.П. Королева. 2017. – Предприятие "Новая техника" (Самара), 2017. – с. 1770-1773. – Текст : непосредственный.

4. Кононова, Н.В. Исследование подсистемы контентной фильтрации с использованием методов машинного обучения / Н.В. Кононова, Ю.А. Андрусенко, Т.А. Самокаева // Студенческая наука для развития информационного общества – сборник материалов VI Всероссийской научно-технической конференции. 22–26 мая 2017. – Северо-Кавказский федеральный университет (Ставрополь), 2017. – с. 268-270. – Текст : непосредственный.

5. Мелдебай, М.А. Анализ мнений покупателей на основе машинного обучения / М.А. Мелдебай, А.К. Сарбасова // Прикладная математика и информатика: современные исследования в области естественных и технических наук – материалы III научно-практической всероссийской конференции (школы-семинара) молодых ученых. 24–25 апреля 2017 года. – Издатель Качалин Александр Васильевич, 2017. – с. 360-363. – Текст : непосредственный.

6. Наумов, Д.П. Регулятор САР на основе машинного обучения / Д.П. Наумов, Д.П. Стариков // Информационные технологии в управлении, автоматизации и мехатронике – сборник научных трудов Международной научно-технической конференции. 06–07 апреля 2017 года. – ЗАО "Университетская книга" (Курск), 2017. – с. 106-114. – Текст : непосредственный.

7. Осколков, В.М. Использование метода машинного обучения для повышения продуктивности на предприятии / В.М. Осколков, Н.И. Шаханов, И.А. Варфоломеев, О.В. Юдина, Е.В. Ершов // Автоматизация и энергосбережение машиностроительного и металлургического производств, технология и надежность машин, приборов и оборудования – материалы XII Международной научно-технической конференции, 21 марта 2017. – Вологодский государственный университет (Вологда), 2017. – с. 177-180. – Текст : непосредственный.

8. Осколков, В.М. Применение параллельных вычислений для прогнозирования на основе алгоритма машинного обучения Random Forest / В.М. Осколков, Н.И. Шаханов, И.А. Варфоломеев, О.В. Юдина, Л.Н. Виноградова, Е.В. Ершов // Сборник трудов конференции Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации. Распознавание, Курск, 16–19 мая 2017 года. – Юго-Западный государственный университет (Курск), 2017. – с. 267-269. – Текст : непосредственный.

9. Соловьев, А.Ю. Применение машинного обучения для прогнозирования неблагоприятных исходов в ургентной хирургии / Соловьев А.Ю., Берегов М.М., Вахеева Ю.М., Баутин А.Н., Гусев А.В. // Медико-биологические, клинические и социальные вопросы здоровья и патологии человека – материалы III Всероссийской образовательно-научной конференции студентов и молодых ученых с международным участием в рамках XIII областного фестиваля "Молодые ученые - развитию Ивановской области". 2017. – Ивановская государственная медицинская академия (Иваново), 2017. – с. 129-130. – Текст : непосредственный.

10. Ткач, Т.Ч. Машинное обучение и обработка больших данных - обучение в основной и средней школе / Т.Ч. Ткач // Актуальные проблемы методики обучения информатике и математике в современной школе – материалы международной научно-практической интернет-конференции. Московский педагогический государственный университет, Москва, 24 апреля 2020 года. – Московский педагогический государственный университет (Москва), 2020. – с. 217-223. – Текст : непосредственный.

11. Agarwal, J. Mining Frequent Quality Factors of Software System Using Apriori Algorithm / Jyoti Agarwal, Sanjay Kumar Dubey, Rajdev Tiwari // Proceedings of the International Conference on Data Engineering and Communication Technology (ICDECT 2016). – Springer Science+Business Media Singapore, 2017. – pp. 481-490. – Текст : непосредственный.

12. Arora, P. Design and Performance Analysis of Distributed Implementation of Apriori Algorithm in Grid Environment / Priyanka Arora, Sarbjeet Singh // ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India. – Springer International Publishing Switzerland, 2014. – pp. 653-661. – Текст : непосредственный.

13. Benhamouda, N.C. Meta-Apriori: A New Algorithm for Frequent Pattern Detection / Neyla Cherifa Benhamouda, Habiba Drias, Célia Hirèche // Asian Conference on Intelligent Information and Database Systems: Intelligent

Information and Database Systems (ACIIDS 2016). – Springer-Verlag Berlin Heidelberg, 2016 – pp. 277-285. – Текст : непосредственный.

14. Child, Ch. The Apriori Stochastic Dependency Detection (ASDD) Algorithm for Learning Stochastic Logic Rules / Christopher Child, Kostas Stathis // International Workshop on Computational Logic in Multi-Agent Systems (CLIMA 2004). – Springer-Verlag Berlin Heidelberg, 2004. – pp. 234-249. – Текст : непосредственный.

15. Choo, Y.H. A Rough-Apriori Technique in Mining Linguistic Association Rules / Yun-Huoy Choo, Azuraliza Abu Bakar, Abdul Razak Hamdan // International Conference on Advanced Data Mining and Applications (ADMA 2008). – Springer-Verlag Berlin Heidelberg, 2008. – pp. 548-555. – Текст : непосредственный.

16. Dahbi, A. Using Multiple Minimum Support to Auto-adjust the Threshold of Support in Apriori Algorithm / Azzeddine Dahbi, Youssef Balouki, Taoufiq Gadi // Proceedings of the Ninth International Conference on Soft Computing and Pattern Recognition (SoCPaR 2017). – Springer International Publishing AG 2018. – pp. 111-119. – Текст : непосредственный.

17. Dhanya, S. An Enhancement of the MapReduce Apriori Algorithm Using Vertical Data Layout and Set Theory Concept of Intersection / S. Dhanya, M. Vysaakan, A. S. Mahesh // Intelligent Systems Technologies and Applications. – Springer International Publishing Switzerland, 2016. – pp. 225-233. – Текст : непосредственный.

18. Djenouri, Y. GA-Apriori: Combining Apriori Heuristic and Genetic Algorithms for Solving the Frequent Itemsets Mining Problem / Youcef Djenouri, Marco Comuzzi // Pacific-Asia Conference on Knowledge Discovery and Data Mining: Trends and Applications in Knowledge Discovery and Data Mining (PAKDD 2017). – Springer International Publishing AG, 2017. – pp. 138-148. – Текст : непосредственный.

19. Fernández-Baizán, M.C. Using the Apriori Algorithm to Improve Rough Sets Results / María C. Fernández-Baizán, Menasalvas Ruiz,

José M. Peña Sánchez, Juan Francisco Martínez Sarrías, Socorro Millán // International Conference on Rough Sets and Current Trends in Computing (RSCTC 2000). – Springer-Verlag Berlin Heidelberg, 2001. – pp. 291-295. – Текст : непосредственный.

20. Inokuchi, A. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data / Akihiro Inokuchi, Takashi Washio, Hiroshi Motoda // European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2000). – Springer-Verlag Berlin Heidelberg, 2000. – pp. 13-23. – Текст : непосредственный.