

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему _____

(промежуточный, этап 2)

Выполнил:

студент группы БПМИ18 _____
Подпись _____ И.О. Фамилия _____
Дата _____

Принял:

руководитель проекта _____
Имя, Отчество, Фамилия _____
Должность _____
Место работы _____

Дата _____ 2020 _____
Оценка (по 10-тибалльной шкале) _____ Подпись _____

Москва 2020

Содержание

1	Задачи второго этапа	3
2	Построение ассоциативных правил	3
2.1	Постановка задачи	3
2.2	Основные понятия	3
3	Алгоритм PrefixSpan	3
3.1	Необходимые определения	3
3.2	Характеристика алгоритма	4
3.3	Ход Алгоритма	4
3.4	Оптимизации	4
4	Алгоритм FIN	6
4.1	Характеристика алгоритма	6
4.2	Необходимые определения	6
4.3	Ход Алгоритма	6
4.4	ROC-tree	7
4.5	Нахождение паттернов старших размеров	7
5	Клиент-серверная архитектура	8
5.1	Rest	8
5.2	HTTP и URI	8
5.3	Устройство WEB-сервиса	9
6	Источники	10

1 Задачи второго этапа

Изучение и сравнение алгоритмов построения ассоциативных правил PrefixSpan и FIN. Разработка клиент-серверной архитектуры с базой данных и API для проведения экспериментов и доступа к результатам.

2 Построение ассоциативных правил

2.1 Постановка задачи

- Входные данные – массив последовательностей событий, параметр *min_support*.
- Выходные данные – набор паттернов, часто встречающихся в массиве последовательностей.

2.2 Основные понятия

- $I = \{i_1, i_2 \dots i_n\}$ – множество всех возможных событий.
- Последовательность – массив событий $[t_1, t_2, \dots t_m]$, где $t \subseteq I, \forall t$
- Пусть $\alpha = [a_1, \dots, a_n]$, $\beta = [b_1, \dots, b_m]$, α называют подпоследовательностью β если существуют целые числа $1 \leq j_1 \leq j_2 \leq \dots \leq j_n \leq m$, такие, что $a_1 \subseteq b_{j_1}, \dots a_n \subseteq b_{j_n}$
- $count(pattern)$ – количество появлений *pattern* как подпоследовательность на множестве всех последовательностей.
- $support(pattern) = \frac{count(pattern)}{|DATABASE|}$
- $min_support \in [0, 1]$ - ограничение снизу на $support()$ для паттернов, которые мы ищем.
- Назовем *sequence* частым, если $support(sequence) \geq min_support$

3 Алгоритм PrefixSpan

3.1 Необходимые определения

- Если α подпоследовательность β , то β – это *super sequence* для α .
- Пусть $\alpha = \langle a_1, \dots, a_n \rangle$, $\beta = \langle b_1, \dots, b_m \rangle$ $m \leq n$, β называют префиксом α , если $b_i = a_i$ для всех $i \leq m - 1$ и $b_m \subseteq a_m$

- Пусть β подпоследовательность α , тогда α' (подпоследовательность α) называют проекцией β , если β это префикс α' , и не существует *super sequence* для α' с таким же условием.

3.2 Характеристика алгоритма

- В результате работы получается полное множество частых паттернов.
- В отличие от Apriori-like алгоритмов нет нужды генерировать и проверять последовательности, не содержащиеся в проецированной базе данных. Все более длинные паттерны получаются из уже существующих коротких.
- Проецированная база данных постоянно уменьшается: уменьшается количество последовательностей и длины префиксов.
- Более производительный чем GSP и FreeSpan.
- Можно улучшить с помощью более эффективного проецирования.

3.3 Ход Алгоритма

1. Изначально алгоритм запускается для пустого паттерна и всей базы данных.
2. Алгоритм получает на вход текущую проекцию (массив событий) и текущий паттерн.
3. Просмотр последовательности на предмет частых событий (айтемов), то есть тех *item*, для которых верно: $support(current_pattern + item) \geq min_support$
4. Добавление к *current_pattern* найденных *item*. Получение новых паттернов.
5. Построение проекции массива последовательностей для каждого из полученных паттернов.
6. Для каждого из новых паттернов рекурсивно запускается алгоритм на соответствующей ему проекции массива последовательностей.

3.4 Оптимизации

Оптимизации алгоритма основываются на более эффективном построении проекций, так называемом "псевдо-проецировании":

Можно заметить что спроецированный массив содержится в исходном. Тогда вместо того чтобы каждый раз "физически" строить новый массив последовательностей, можно строить массив пар $\langle sequence_number, offset \rangle$, где:

sequence_number – номер последовательности из базы данных.

offset – сдвиг последовательности, по сути индекс начала события в данной проекции.

Стоит заметить, что использовать эту оптимизацию можно только тогда, когда вся база данных помещается в оперативную память, так как множественное обращение к разным участкам памяти ПЗУ сильно влияет на производительность.

4 Алгоритм FIN

4.1 Характеристика алгоритма

- Работает эффективнее чем PrePost и FR-growth.
- РОС-tree в отличие от PPC-tree хранит только pre-order, делая алгоритм эффективнее по памяти.
- На плотных датасетах потребляет намного меньше памяти чем PrePost.

4.2 Необходимые определения

- *pre-order* – номер вершины для обхода в порядке: (текущая вершина; левый ребенок; правый ребенок)
- *N-info* – для вершины N из РОС-tree это пара $(pre-order, count)$ (соответствующие вершине pre-order, и частота одноэлементного паттерна).
- Если отсортировать частые элементы в каждой последовательности из базы данных в порядке убывания их частоты, получится множество $L1$.
- $i_1 > i_2$, если $i_1, i_2 \in L1$, и частота i_1 больше (то есть в $L1$ он стоит раньше чем i_2).
- $Nodeset(i)$ – множество всех *N-info* для события i .
- $Nodeset(i_1, i_2)$ – если $i_1 > i_2$ - множество тех (pre_k, c_k) из $Nodeset(i_2)$, для которых верно, что существует вершина N , соответствующая i_1 и являющаяся предком (pre_k, c_k) .
- $Nodeset$ K -элементного множества: Пусть $P = p_1 p_2 \dots p_k$ – K -элементное множество ($p_j \in L1$ и $p_1 > p_2 > \dots > p_k$). Положим $P_1 = p_1 p_3 \dots p_k$ и $P_2 = p_2 p_3 \dots p_k$, тогда $Nodeset(P) = Nodeset(P_1) \cap Nodeset(P_2)$.

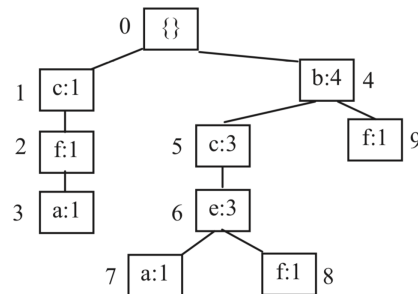
4.3 Ход Алгоритма

1. Построение РОС-tree и выявления всех частых одноэлементных множеств событий. (и их Nodesets)
2. Запускается обход РОС-tree для нахождения всех частых двухэлементных множеств и их Nodesets.
3. Построение всех частых $K (> 2)$ - элементных множеств.

4.4 ROC-tree

ROC-tree отличается от PPC-tree только тем что для каждой вершины хранит лишь ее pre-order. Для каждой последовательности из базы данных отсортируем ее частые элементы в порядке убывания частоты. Получим упорядоченные последовательности частых элементов ($L1$), по сути множество слов. По нему построим префиксное дерево и посчитаем pre-order порядок каждого элемента. Пример построения:

ID	Items	Ordered frequent items
1	a, c, g, f	c, f, a
2	e, a, c, b	b, c, e, a
3	e, c, b, i	b, c, e
4	b, f, h	b, f
5	b, f, e, c, d	b, c, e, f



4.5 Нахождение паттернов старших размеров

Утверждается, что если $Nodeset(P) = \{(pre_1, c_1), \dots, (pre_m, c_m)\}$, где P – K-элементное множество, то $count(P) = c_1 + c_2 + \dots + c_m$. На использовании этого свойства и строится алгоритм FIN. Построение ROC-tree позволяет рассчитать $Nodesets$ одноэлементных множеств. Далее по нему, пользуясь определением, можно рассчитать все $Nodesets$ двухэлементных множеств. Заметим, что зная их можно рекурсивно посчитать $Nodesets$ трехэлементных множеств. Таким образом, увеличивая размер рассматриваемых множеств можно найти все часто встречаемые паттерны.

5 Клиент-серверная архитектура

5.1 Rest

Rest – Representational State Transfer («передача состояния представления»), набор архитектурных принципов создания сети, не задает строгих деталей реализации но накладывает ряд ограничений:

- Система состоит из клиентов и серверов.
- Отсутствие состояния – сервер и клиент не отслеживают состояния друг друга.
- Единообразие (унифицированность) интерфейса. В идеологии Rest ресурсом может быть что угодно, у каждого ресурса есть постоянный идентификатор. Для идентификации ресурсов обычно используется URI, а в качестве протокола взаимодействия – HTTP. Клиент управляет ресурсами отправляя серверу их представления (например, в формате JSON) и получая в ответ гипермедиа. Все отправляемые сообщения самодостаточные, то есть содержат всю информацию, необходимую для их понимания.
- Клиенты и промежуточные узлы могут кэшировать полученную от сервера информацию, что избавляет от полного прохода данных по сети, в случае, когда они снова понадобятся.
- Иерархическая структура сетей образует слои: каждый компонент ограничен возможностями видеть и взаимодействовать только с соседним слоем.

Архитектура Rest проста по своему устройству и легко масштабируется, поэтому идеально подойдет для создания необходимой клиент-серверной инфраструктуры.

5.2 HTTP и URI

HTTP – HyperText Transfer Protocol («протокол передачи гипертекста») – протокол передачи данных в клиент-серверной архитектуре. Основными объектами манипуляции HTTP являются ресурсы, для их идентификации используется *URI* - унифицированный идентификатор ресурсов. URI устроен следующим образом:

$URI = scheme : [//authority]path[?query][fragment]$, где:

- *scheme* – схема обращения к ресурсу, в нашем случае это сетевой протокол HTTP.
- *authority* – иерархическая часть, в нашем случае адрес сервера.
- *path* – путь к ресурсу.
- *query* – строка-запрос, обычно содержит последовательность вида атрибут–значение
- *fragment* – позволяет идентифицировать вторичный ресурс посредством ссылки на первичный.

5.3 Устройство WEB-сервиса

Для реализации серверного бекэнда был выбран фреймворк Flask и библиотека Flask-REST-JSONAPI. Для базы данных используются библиотеки SQLAlchemy и Marshmallow. Для взаимодействия с сервером используется следующий интерфейс:

- `http://localhost/experiment/get_ids` – получить список идентификаторов экспериментов.
- HTTP запрос `POST experiment/{experiment_id}` – создать эксперимент с заданными параметрами.
- `http://localhost/experiment/{experiment_id}/start` – запустить созданный эксперимент.
- `http://localhost/experiment/{experiment_id}/get_results` – получить результаты эксперимента.

6 Источники

PrefixSpan:

- [PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth](#) (Jian Pei, Jiawei Han, Behzad Mortazavi-asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu. (2002))

FIN:

- [Fast mining frequent itemsets using Nodesets](#)(Expert Systems with Applications 41 (2014), Zhi-Hong Deng, Sheng-Long Lv)

Rest:

- Roy Fielding. [Architectural Styles and the Design of Network-based Software Architectures](#) (2000)