САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

ПОИСК И ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ ПОСЛЕДОВАТЕЛЬНЫХ СОБЫТИЙ ПРИ АНАЛИЗЕ ЛОГОВ

Дипломная работа студента 545 группы

Алеева Алексея Валерьевича

Научный руководитель	…… / подпись /	Аспирант, Елизаров Е. А.
Рецензент	…… / подпись /	Ведущий инженер, ООО «Квестора», Хлуднев М. В.
“Допустить к защите” заведующий кафедрой,	…… / подпись /	д.ф.-м.н., проф. Терехов А.Н.

Санкт-Петербург
2012

SAINT PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

SEQUENTIAL PATTERN MINING AND USAGE IN LOG ANALYSIS

by

Alexey Aleev

Graduate paper

Supervisor	……	Ph. D. Student, E. A. Elizarov
Reviewer	……	Principal Engineer, LLC Questora, M. V. Khludnev
"Approved by" Head of Department	……	Professor A. N. Terekhov

Saint Petersburg
2012

Оглавление

Введение

Уже в течение многих лет неуклонно растет интерес к методам обнаружения знаний в базах данных (knowledge discovery in databases). Объемы современных баз данных, которые весьма внушительны, вызвали устойчивый спрос на новые масштабируемые алгоритмы анализа данных. Одним из популярных и весьма актуальных методов обнаружения знаний являются алгоритмы поиска шаблонов последовательных событий (sequential pattern mining или SPM).

Использование шаблонов позволяет находить закономерности между связанными во времени событиями.

Одним из примеров области применения таких знаний может послужить интернет-торговля. Поиск наиболее частых последовательностей покупок позволяет получить информацию о том, через какой промежуток времени после покупки товара “А” человек наиболее склонен купить товар “Б”, или в какой последовательности приобретаются товары. Получаемые закономерности в действиях покупателей можно использовать для персонализации клиентов, формирования более выгодного предложения, стимулирования продаж определенных товаров или управления запасами.

Другим примером области применения задачи SPM может послужить веб-аналитика. Зная наиболее популярные последовательности переходов по страницам, можно размещать на соответствующих страницах определенный контент или изменить структуру сайта с целью более быстрого и удобного доступа к некоторым страницам. Подобный анализ сайтов обычно требует больших денежных затрат (в случае привлечения сторонних организаций), либо наличия эксперта в данной области, поскольку свободные решения

(Яндекс.Метрика, Google Analytics, AWStats) не предоставляют таких возможностей.

Кроме того, поиск шаблонов последовательных событий может применяться в биоинформатике для поиска каркасов белковых последовательностей, в телекоммуникационных компаниях для анализа данных об авариях на различных узлах телекоммуникационной сети, а также при анализе последствий природных катаклизмов.

Данные о событиях могут быть записаны и храниться различными способами, поэтому процесс поиска шаблонов последовательных событий для одних систем может быть неприменим к другим системам. На данный момент не существует инструмента, позволяющего решать задачу SPM для различных форматов данных. Существуют открытые реализации некоторых алгоритмов решения задачи SPM, но для их применения требуется привести базу последовательностей к определенному виду, что не так просто. Результаты работы алгоритмов, в свою очередь, требуют обратного преобразования. При этом, как правило, любое событие, будь то покупка товара в интернет-магазине, начало сессии пользователя на сайте или авария на одном из узлов телекоммуникационной сети, каким-либо образом журналируется. В связи с этим предлагается решать задачу SPM на основе базы журналов (или логов).

Еще одной проблемой является наличие большого числа алгоритмов для поиска шаблонов последовательных событий. Они имеют разную степень эффективности, а также отличаются возможностями задания некоторых параметров. В связи с этим для решения задач, для которых требуется поиск шаблонов последовательных событий, необходимо вначале провести анализ существующих алгоритмов и определить, какой из них лучше подходит в данной ситуации.

1. Постановка задачи

Целью данной работы является создание инструментального средства для анализа журналов, способного решать задачу SPM для различных предметных областей.

Для сбора и хранения журналов о событиях будет использоваться инструментальное средство GDLogTool, в разработке которого автор принимает участие.

Для достижения цели были выделены следующие задачи:

1. Исследовать существующие методы решения задачи SPM.
2. Выбрать/разработать эффективный алгоритм решения задачи SPM.
3. Разработать механизм определения типов журналов.
4. Реализовать данный механизм и выбранный алгоритм в рамках проекта GDLogTool.

2. Обзор существующих методов решения задачи SPM

Задача поиска шаблонов последовательных событий является важной задачей в области анализа данных, существует множество алгоритмов для ее решения. В данной работе рассмотрены основные подходы и алгоритмы, а также описаны их преимущества и недостатки. На основе этого анализа было принято решение, какой алгоритм использовать для реализации в рамках проекта GDLogTool.

Сначала дадим формальное описание задачи SPM, а после этого опишем основные методы для ее решения.

2.1. Формальное описание задачи поиска шаблонов последовательных событий

Рассмотрим постановку задачи поиска последовательных шаблонов, используя задачу анализа рыночной корзины, поскольку именно так она была изначально сформулирована в основополагающей статье Агравала и Шриканта[1]. При этом будем придерживаться терминологии первоисточника.

Пусть имеется база данных D, в которой каждая запись представляет собой клиентскую транзакцию. Каждая транзакция содержит следующие поля: идентификатор клиента, дата/время транзакции и набор купленных товаров. Добавим ограничение, что ни один клиент не имеет двух или более транзакций, совершенных в один и тот же момент времени.

Введем несколько понятий.

Предметный набор – это непустой набор предметов (товаров), появившихся в одной транзакции.

Последовательность – это упорядоченный список предметных наборов.

Последовательности будем заключать в треугольные скобки, а предметные наборы - в круглые. Тогда, если обозначать предметы целыми числами, то предметные наборы будут записаны в виде *(2, 4, 5)*, *(1, 3)*, а последовательность, содержащая эти наборы - *<(2, 4, 5), (1, 3)>*. Если предметы появились в одном наборе, это значит, что они были приобретены одновременно.

Обозначим предметный набор $I = (i_1,i_2,...,i_m)$ (от англ. *itemset* - предметный набор, набор элементов), где i_j - предмет. Обозначим последовательность $S = <I_1,I_2,...,I_m>$, где I_i - предметный набор. *Длиной* последовательности будем называть количество предметов в этой последовательности, а последовательность длины k - *k-последовательностью*. Последовательность S_1 *содержится* в другой последовательности S_2, если все предметные наборы S_1 содержатся в предметных наборах S_2, при этом порядок надмножеств из S_2 соответствует порядку предметных наборов S_1. Например, последовательность *<(3),(4,5),(8)>* содержится в последовательности *<(7),(3,8),(9),(4,5,6),(8)>*, поскольку *(3)* содержится в *(3,8)*, *(4,5)* содержится в *(4,5,6)* и *(8)* - в *(8)*. Однако, *<(3),(5)>* не содержится в *<(3,5)>* и наоборот, поскольку в первой последовательности предметы 3 и 5 были куплены один за другим, а во второй – куплены совместно.

Все транзакции одного клиента могут быть показаны в виде последовательности, в которой они упорядочены по дате, времени или

номеру визита. Такие последовательности будем называть клиентскими. Формально это записывается следующим образом.

Пусть клиент совершил несколько упорядоченных во времени транзакций $T_1,T_2,...,T_k$. Тогда каждый предметный набор в транзакции T_i обозначим $I(T_i)$, а каждую клиентскую последовательность для данного клиента запишем как $<I(T_1), I(T_2),...,I(T_k)>$. Иными словами, клиентская последовательность – это последовательность предметных наборов, соответствующих транзакциям, совершенным данным клиентом.

Последовательность *S* называется *поддерживаемой* клиентом, если она содержится в клиентской последовательности данного клиента. Тогда *поддержка* последовательности определяется как число клиентов, поддерживающих данную последовательность.

Для базы данных клиентских транзакций задача поиска шаблонов заключается в обнаружении последовательностей, имеющих поддержку выше заданного порогового значения. Каждая такая последовательность является шаблоном последовательных событий.

Далее будем называть последовательности, удовлетворяющие ограничению минимальной поддержки, *частыми* последовательностями.

Последовательность *S* называется *максимальной*, если она не содержится в какой-либо другой последовательности. Иногда требуется найти не все шаблоны последовательных событий, а только те из них, которые являются максимальными.

Теперь опишем основные подходы для решения задачи поиска последовательных шаблонов и соответствующие им алгоритмы.

2.2. Алгоритмы вида «candidate generate-and-test»

Первыми алгоритмами, разработанными для решения задачи SPM, являются AprioriAll, AprioriSome и DynamicSome, представленные в работе Агравала и Шриканта[1]. Все эти алгоритмы используют подход генерации и отбора кандидатов частовстречающихся последовательностей, а также свойство *антимонотонности*, которое заключается в том, что каждая подпоследовательность частой последовательности должна также быть частой.

Работа данных алгоритмов состоит из нескольких фаз:

Фаза сортировки заключается в перегруппировке записей в таблице транзакций. Сначала записи сортируются по уникальному ключу покупателя, а затем по времени внутри каждой группы.

Фаза отбора кандидатов - в исходном наборе данных производится поиск всех частых предметных наборов. В частности, на этом этапе происходит поиск всех одноэлементных шаблонов.

Фаза трансформации. Производится для ускорения процесса проверки присутствия последовательностей в наборе транзакций покупателей. Трансформация заключается в замене каждой транзакции списком частых предметных наборов, которые в ней содержатся. При этом если в транзакции отсутствуют частые предметы, то данная транзакция не учитывается. Аналогичным образом не учитываются предметы, не являющиеся частыми, а также последовательности, транзакции которых не содержат частых предметных наборов.

Фаза генерации последовательностей - из полученных на предыдущих шагах последовательностей строятся более длинные шаблоны последовательностей.

Фаза максимизации - среди имеющихся последовательностей происходит поиск тех, что не входят в более длинные последовательности. Данный шаг является опциональным.

Алгоритмы AprioriAll, AprioriSome и DynamicSome отличаются только в четвертой фазе. AprioriAll имеет немного более высокие показатели производительности по сравнению с двумя другими алгоритмами.

Позже был разработан алгоритм GSP, в котором были введены дополнительные параметры, благодаря чему стало возможным учитывать ограничения по времени между соседними транзакциями шаблона и решать более широкий круг задач. В общем виде работа алгоритма GSP похожа на AprioriAll, при этом, как указано в работе Шриканта и Агравала [6], GSP работает до 20 раз быстрее, поэтому опишем подробно именно этот алгоритм.

При поиске зависимостей в данных нас могут интересовать только такие, где одни события наступают вскоре после других. Если же этот промежуток времени достаточно велик, то такая зависимость может не представлять значения. Проиллюстрируем сказанное на примере.

Владельца книжного магазина скорее всего не заинтересует тот факт, что человек, купивший "Основание" Азимова, спустя три года купил "Основатели и Империя". Его могут интересовать покупки, интервал между которыми составляет, например, три месяца.

Каждая совершённая покупка представляет собой элемент последовательности. Последовательность состоит из одного и более элементов. В некоторых случаях не имеет значения, если бы наборы товаров, содержащиеся в элементе последовательности, входили не в одну покупку (транзакцию), а составляли бы несколько покупок, при условии, что время

транзакций (покупок) укладывалось бы в определённый интервал времени (окно).

В случае с алгоритмом GSP требуется учитывать дополнительные условия, чтобы определить, содержит ли последовательность указанную последовательность (подпоследовательность). Добавим к описанному ранее определению понятие скользящего окна. Допускается, что элемент последовательности может состоять не из одной, а из нескольких транзакций, если разница во времени между ними меньше чем размер окна. Помимо этого, введем такие параметры как минимальное и максимальное допустимое время между транзакциями (min-gap и max-gap). Тогда последовательность $d = < d_1 ... d_m >$ содержит последовательность $s = < s_1 ... s_m >$, если существуют такие целые числа $l_1 <= u_1 < l_2 <= u_2 < ... < l_n <= u_n$, что:

1. s_i содержится в объединении d_k, где $l_i <= k <= u_i$, $1 <= i <= n$.

2. Разница во времени транзакций $d_{u[i]}$ и $d_{l[i]}$ не превосходит размера окна, $1 <= i <= n$,

3. (Время транзакции $d_{l[i]}$) - (время транзакции $d_{u[i-1]}$) >= min-gap, $2 <= i <= n$,

4. (Время транзакции $d_{u[i]}$) - (время транзакции $d_{l[i-1]}$) <= max-gap, $2 <= i <= n$.

Описание работы алгоритма GSP

Работа алгоритма GSP заключается в нескольких проходах по исходному набору данных. На первом проходе алгоритм вычисляет поддержку для каждого предмета и выделяет из них частые. Каждый подобный предмет представляет собой одноэлементную последовательность. В начале каждого последующего прохода имеется некоторое число часто встречающихся последовательностей, выявленных на предыдущем шаге

алгоритма. Из них будут формироваться более длинные последовательности-кандидаты. Каждый кандидат представляет собой последовательность, длина которой на один больше чем у последовательностей, из которых кандидат был сформирован. Таким образом, число элементов всех кандидатов одинаково. После формирования кандидатов происходит вычисление их поддержки. В конце шага определяется, какие кандидаты являются частыми. Найденные частые последовательности послужат исходными данными для следующего шага алгоритма. Работа алгоритма завершается тогда, когда не найдено ни одной новой частой последовательности в конце очередного шага, или когда невозможно сформировать новых кандидатов.

Таким образом, в работе алгоритма можно выделить две основные операции:

- генерация кандидатов;
- подсчёт поддержки кандидатов.

Рассмотрим эти операции более подробно.

Генерация кандидатов

Пусть L_k содержит все частые k-последовательности, а C_k - набор кандидатов из k-последовательностей. В начале каждого шага мы располагаем L_{k-1} - набором частых (k-1)-последовательностей. Необходимо на их основе построить набор всех частых k-последовательностей.

Введём определение смежной подпоследовательности.

При наличии последовательности $s = < s_1 s_2 ... s_n >$ и подпоследовательности *c*, *c* будет являться смежной последовательностью *s*, если соблюдается одно из условий:

1. *c* получается из *s* при удалении предмета из первого (s_1) или последнего (s_n) предметных наборов.

2. *c* получается из *s* при удалении одного предмета из предметного набора s_i, если в его составе не менее двух предметов.

3. *c* - смежная подпоследовательность *c`*, где *c`* - смежная подпоследовательность *s*.

Например, дана последовательность *s* = *<(1,2),(3,4),(5),(6)>*. Последовательности *<(2),(3,4),(5)>*, *<(1,2),(3),(5),(6)>* и *<(3),(5)>* являются смежными подпоследовательностями *s*.

А последовательности *<(1,2),(3,4),(6)>* и *<(1),(5),(6)>* таковыми не являются.

Если некоторая последовательность содержит последовательность *s*, то она также содержит и все смежные подпоследовательности *s*.

Генерация кандидатов происходит в две фазы.

Фаза объединения.

Мы создаем последовательности-кандидаты путем объединения двух последовательностей из L_{k-1}. Последовательность s_1 объединяется с s_2, если подпоследовательность, образующаяся путем удаления первого предмета из s_1 будет та же, что и в случае удаления последнего предмета из s_2. Объединение последовательностей происходит путем добавления к s_1 соответствующего предмета из последнего предметного набора s_2. Если последний предмет из s_2 составлял одноэлементный предметный набор, то при объединении он будет добавлен к s_1 как новый предметный набор. В противном случае, он будет добавлен в последний предметный набор s_1. При объединении L_1 с L_1 нужно добавить предмет к s_2 как отдельный предметный набор, а также в качестве дополнительного предмета в предметный набор последовательности s_1, так как *<(x, y)>* и *<(x),(y)>* дадут одну и ту же последовательность *<(y)>* при удалении первого предмета. Отметим, что

исходные последовательности s_1 и s_2 являются смежными подпоследовательностями для новой последовательности-кандидата.

Фаза очистки.

Удаляем последовательности-кандидаты, которые содержат смежные *(k-1)*-последовательности, чья поддержка меньше минимально допустимой. Если параметр `max-gap` не задан, то также удаляются кандидаты, содержащие последовательности (в том числе не смежные), поддержка которых меньше минимально допустимой.

Подсчёт поддержки кандидатов

Сканируя набор последовательностей, мы обрабатываем их по очереди. Для тех кандидатов, которые содержатся в обрабатываемой последовательности, увеличиваем значение поддержки на единицу.

Для уменьшения количества кандидатов, требующих проверки вхождения в обрабатываемые последовательности, а также для увеличения скорости проверки используется хэш-дерево. Работа с ним, как и его структура, детально описана в статье Шриканта и Агравала[2].

Из недостатков упомянутых выше алгоритмов можно выделить большое количество обращений к базе данных. Количество обращений соответствует длине максимального кандидата. Другим недостатком является большое число генерируемых кандидатов, что сильно сказывается на производительности для больших наборов данных.

Еще одним методом решения задачи SPM, использующим генерацию и тестирование кандидатов, является метод «вертикального» представления данных. Данное представление заключается в том, что вместо набора клиентских последовательностей рассматривается набор списков транзакций для каждого предмета. Список транзакций для предмета *i* состоит из

транзакций, в которых присутствует данный предмет. Существуют несколько алгоритмов, использующих такое представление: SPADE[8], SPAM[2] и Last Position Induction SPAM[7]. Последние два алгоритма имеют гораздо большую эффективность за счет оптимизации вычисления поддержки кандидатов, поэтому далее детально будут описаны именно эти алгоритмы.

Описание алгоритма SPAM(**S**equential **PA**ttern **M**ining algorithm)

SPAM является первым алгоритмом поиска шаблонов последовательных событий, использующим обход дерева в глубину. Для начала опишем дерево, с которым алгоритм будет работать.

Предположим, что в базе транзакций все предметы сравнимы (например, лексикографически). Если предмет i предшествует предмету j, то этот факт записывается как $i < j$. Будем считать, что предметы в одной транзакции представлены в порядке возрастания. Данное отношение порядка расширяется на последовательности. Для последовательностей S_a и S_b $S_a < S_b$, если S_a является подпоследовательностью S_b.

Рассмотрим представление всех последовательностей в виде дерева последовательностей (будем называть его просто деревом) со следующей структурой. Корень дерева обозначим как <> (пустая последовательность). Далее, структура дерева определяется рекурсивно: если n - узел дерева, то потомками n являются все такие узлы n', что $n < n'$, и для любого узла m из отношения $n' < m$ следует, что $n < m$.

Каждая последовательность в дереве относится к одному из двух видов: s-расширение или i-расширение. S-расширенной будем называть последовательность, полученную в результате добавления еще одной транзакции, состоящей из одного элемента, к последовательности-предку данной последовательности в дереве. I-расширенной будем называть

последовательность, полученную в результате добавления нового предмета в последний предметный набор последовательности-предка данной последовательности, при этом новый предмет должен превосходить все остальные предметы этого предметного набора. Например, если имеется последовательность S_a = <(a,b,c),(a,b)>, то <(a,b,c),(a,b), (a)> - *s*-расширенная последовательность, а <(a,b,c),(a,b,d)> - *i*-расширенная.

Если мы будем генерировать последовательности с помощью обхода дерева, то каждый узел в дереве может генерировать *s*-расширенных и *i*-расширенных потомков. Процесс генерации *s*-расширенной последовательности будем обозначать как S-step, а процесс генерации *i*-расширенной последовательности - как I-step. Таким образом, с каждым узлом *n* дерева ассоциируется два набора: S_n - набор предметов-кандидатов, которые рассматриваются для возможных расширений последовательностей с помощью S-step (будем называть их *s*-расширениями), и I_n - аналогичный набор для *i*-расширений.

SPAM обходит дерево последовательностей в глубину. Для каждого узла *n* проверяются значения поддержки всех *s*-расширений и *i*-расширений потомков. Если поддержка сгенерированной последовательности *s* больше либо равна заданной минимальной поддержке, то последовательность добавляется в результат, и обход продолжается рекурсивно от *s*. Если поддержка *s* меньше заданной минимальной поддержки, то запуска обхода для *s* не происходит.

Для уменьшения числа кандидатов *s*-расширений и *i*-расширений используются две техники: S-step pruning и I-step pruning. Они используются для уменьшения размеров наборов S_n и I_n для каждого узла *n*. При этом гарантируется, что все узлы, соответствующие частым последовательностям, будут посещены при обходе. Далее опишем каждую из техник.

S-step pruning.

Рассмотрим последовательность s узла n. Пусть S_a=<s, (i_j)> и S_b=<s, (i_k)> - s-расширенные последовательности для s. Предположим, что S_a - частая последовательность, а S_b не является частой. По свойству антимонотонности, <s,(i_j),(i_k)> и <s,(i_j,i_k)> не могут быть частыми, поскольку обе последовательности содержат S_b. Значит, мы можем убрать i_k из всех наборов S_m и I_m, где m - любой узел, соответствующий частой s-расширенной последовательности-потомку s.

I-step pruning.

Рассмотрим i-расширения для s=<s', (i_1,...,i_n)>. Пусть S_a=<s', (i_1,...,i_n,i_j)>, S_b=<s, (i_1,...,i_n,i_k)> и i_j < i_k. Если S_a - частая последовательность, а S_b - не частая, то последовательность <s', (i_1,...,i_n,i_j,i_k)> не может быть частой по свойству антимонотонности. Значит, можно убрать i_k из I_m, где m является частым i-расширением для s. Можно так же убрать из S_m те же предметы, что рассматривались в S-step pruning. Если по ходу S-step pruning последовательность <s,(i_l)> оказалась не частой, тогда последовательность <S_a,(i_l)> тоже окажется не частой для любого частого i-расширения S_a для s.

Псевдокод обхода дерева с учетом S-step pruning и I-step pruning представлен в Листинге 1.

DFS-Pruning(node $n = (s_1, \ldots, s_k)$, S_n, I_n)

```
(1)   S_temp = ∅
(2)   I_temp = ∅
(3)   For each (i ∈ S_n)
(4)       if ( (s_1, ..., s_k, {i}) is frequent )
(5)           S_temp = S_temp ∪ {i}
(6)   For each (i ∈ S_temp)
(7)       DFS-Pruning((s_1, ..., s_k, {i}), S_temp,
              all elements in S_temp greater than i)
(8)   For each (i ∈ I_n)
(9)       if ((s_1, ..., s_k ∪ {i}) is frequent)
(10)          I_temp = I_temp ∪ {i}
(11)  For each (i ∈ I_temp)
(12)      DFS-Pruning((s_1, ..., s_k ∪ {i}), S_temp,
              all elements in I_temp greater than i)
```

Листинг 1. Алгоритм SPAM

Для эффективной работы алгоритм использует представление данных в виде массива битов. Такое представление создается для всех предметов. Каждый битовый массив содержит биты для всех транзакций базы. Если предмет *i* встречается в транзакции *j*, то бит, соответствующий транзакции *j* в массиве битов для *i*, равен 1. В противном случае значение бита будет 0. Элементы массива упорядочены и разделены по клиентам, при этом если транзакция *i* произошла до транзакции *j*, то индекс бита, соответствующего транзакции *i*, будет меньше индекса бита, соответствующего транзакции *j*. Битовое представление распространяется на предметные наборы и последовательности. При наличии массивов для предметов *i* и *j*, массив для предметного набора *(i, j)* получается путем применения побитового “И” к имеющимся массивам. Для последовательностей массивы устроены следующим образом: если последний предметный набор последовательности

представлен в транзакции *j*, а все остальные наборы последовательности содержатся в транзакциях до *j*, то бит, соответствующий транзакции *j*, равен 1. В противном случае соответствующий бит равен 0. Будем обозначать массив битов для последовательности *s* как *B(s)*.

Клиентские последовательности разделяются на различные наборы в зависимости от размера. Если размер последовательности находится между 2^k+1 и 2^{k+1}, то последовательность рассматривается как 2^{k+1}-битовая. Минимальное значение *k* равно 1.

Каждый набор 2^k-битовых последовательностей соответствует определенной матрице битов, каждый столбец которой имеет длину 2^k. Вычисление поддержки для каждого клиента не составляет труда, для этого нужно проверить, все ли биты имеют значение 0 или нет. Далее будет представлен пример базы клиентских последовательностей и ее представление в виде матрицы битов.

Customer ID (CID)	TID	Itemset
1	1	$\{a, b, d\}$
1	3	$\{b, c, d\}$
1	6	$\{b, c, d\}$
2	2	$\{b\}$
2	4	$\{a, b, c\}$
3	5	$\{a, b\}$
3	7	$\{b, c, d\}$

Рис. 1. Пример базы клиентских последовательностей

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

Рис. 2. Представление базы в виде матрицы битов

Транзакция 6 в примере содержит предметы *b*, *c* и *d*, поэтому значения битов, соответствующие этой транзакции в массивах битов для *b*, *c* и *d*, равны 1, а значение бита данной транзакции для предмета *a* равно 0.

Генерация кандидатов происходит посредством процессов S-step и I-step. Опишем эти процессы.

S-step process.

Пусть даны $B(S_a)$ и $B(i)$ для последовательности S_a и предмета i соответственно, и мы хотим совершить S-step для S_a, используя предмет i.

Предметный набор (i) будет добавлен к последовательности S_a, а битовый массив $B(S_g)$ для новой последовательности должен обладать следующим свойством: если бит в массиве равен 1, то соответствующая транзакция j содержит i, а все остальные предметные наборы в S_g содержатся в транзакциях до j.

Допустим, что первый ненулевой бит в $B(S_a)$ имеет индекс k. Транзакция j, соответствующая биту k, содержит последний предметный

набор последовательности S_a, и все остальные предметные наборы S_a должны содержаться в транзакциях до *j*. Поскольку *(i)* добавлен к S_a после ее последнего предметного набора, *(i)* должен быть представлен в транзакции сразу после *j* в S_g. Таким образом, значение бита *k* в $B(S_g)$ должно быть равно 0. С другой стороны, если предмет *i* содержится в некоторой транзакции после *j*, то соответствующий бит в $B(S_g)$ должен быть равен 1. Получаем, что биты в $B(S_g)$ с индексами больше *k* имеют те же значения, что и соответствующие биты в *B(i)*. В соответствии с этими наблюдениями, процесс создания массива $B(S_g)$ происходит слудющим образом:

1. Создается массив, у которого все биты с индексами, не превосходящими *k*, равны 0, а все последующие равны 1.

2. К созданному массиву и *B(i)* применяется операция побитового “И”. Полученный в результате этой операции массив является битовым массивом для новой последовательности S_g.

Описанный процесс проиллюстрирован на рис.3:

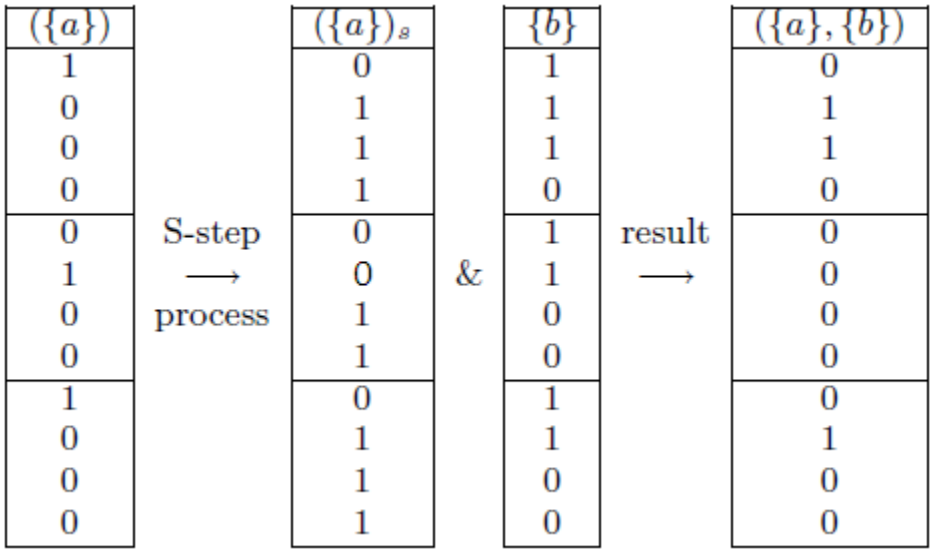

Рис. 3. S-step process для последовательности <a> и предмета b

I-step process.

Пусть имеются $B(S_a)$ и $B(i)$ для последовательности S_a и предмета i, и требуется совершить I-step для S_a, используя i. Данный процесс сгенерирует новую последовательность S_g, полученную добавлением i в последний предметный набор S_a. Битовый массив для S_g должен обладать свойством: если бит массива равен 1, то соответствующая этому биту транзакция j содержит последний предметный набор последовательности S_g, а все остальные предметные наборы S_g содержатся в транзакциях до j.

Легко убедиться, что битовый массив, полученный путем применения операции побитового “И” к $B(S_a)$ и $B(i)$, обладает описанным свойством, поэтому он является битовым массивом для S_g.

На рис.4 продемонстрирован процесс определения битового массива для последовательности, полученной в результате I-step для последовательности <(a),(b)> и предмета d.

$(\{a\},\{b\})$		$\{d\}$		$(\{a\},\{b,d\})$
0		1		0
1		1		1
1		1		1
0		0		0
0		0	result	0
0	&	0	$\longrightarrow$	0
0		0		0
0		0		0
0		0		0
1		1		1
0		0		0
0		0		0

Рис. 4. I-step process для последовательности <(a),(b)> и предмета d

Алгоритм SPAM показывает значительно лучшие результаты по сравнению со SPADE и GSP. Для больших объемов данных скорость его работы на порядок выше скорости работы других алгоритмов.

Описание алгоритма Last Position Induction SPAM

Алгоритм Last Position Induction SPAM основан на алгоритме SPAM и отличается от него способом проверки кандидатов. В алгоритме SPAM для этого приходится сначала создавать массив, применяя операции побитового “И”, которые занимают основную часть времени работы алгоритма.

В LPI SPAM используется следующий факт: если позиция последнего вхождения предмета в клиентскую последовательность меньше текущей позиции, то последовательность, дополненная этим предметом в данной позиции, не может содержаться в рассматриваемой клиентской последовательности.

При чтении базы последовательностей создаются списки позиций вхождений предметов в транзакции каждого клиента. Примеры базы последовательностей и списков позиций представлены на рис.5 и рис.6:

CID	*Customer Sequence*
10	$ac(bc)d(abc)ad$
20	$b(cd)ac(bd)$
30	$d(bc)(ac)(cd)$

Рис. 5. Пример базы последовательностей

SID	*Item Positions*
10	$a : 1 \rightarrow 5 \rightarrow 6 \rightarrow null$ $b : 3 \rightarrow 5 \rightarrow null$ $c : 2 \rightarrow 3 \rightarrow 5 \rightarrow null$ $d : 4 \rightarrow 7 \rightarrow null$
20	$a : 3 \rightarrow null$ $b : 1 \rightarrow 5 \rightarrow null$ $c : 2 \rightarrow 4 \rightarrow null$ $d : 2 \rightarrow 5 \rightarrow null$
30	$a : 3 \rightarrow null$ $b : 2 \rightarrow null$ $c : 2 \rightarrow 3 \rightarrow 4 \rightarrow null$ $d : 1 \rightarrow 4 \rightarrow null$

Рис. 6. Пример таблицы списков вхождений предметов в клиентские последовательности

Для осуществления проверки кандидатов потребуется создание специальной таблицы `ITEM_IS_EXIST_TABLE`. Она создается при чтении базы транзакций и имеет следующий вид. Каждая строка представляет собой битовый массив для некоторой позиции клиентской последовательности. Размер массива соответствует числу предметов базы последовательностей. Если бит массива равен 1, это означает, что позиция последнего вхождения соответствующего предмета в клиентской последовательности не меньше данной позиции. В противном случае значение бита равно 0. Часть таблицы `ITEM_IS_EXIST_TABLE` для базы из рис.5, соответствующая клиенту 10, показана на рис.7:

Item / Pos	a	b	c	d
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	0	0	1
6	0	0	0	1
7	0	0	0	0

Рис. 7. ITEM_IS_EXIST_TABLE для клиента 10

В качестве дополнительного параметра методу `DFS-Pruning` передается массив позиций вхождений последнего предмета текущего шаблона в клиентские последовательности `borderArray`.

Рассмотрим S-step. Для вычисления поддержки кандидата, полученного путем добавления предметного набора *(i)*, нужно всего лишь просуммировать значения битов предмета *i* на позиции `borderArray[j]+1` таблицы `ITEM_IS_EXIST_TABLE` для каждого клиента *j*.

Если требуется совершить I-step для последовательности S_a, используя предмет *i*, с помощью списка позиций вхождений для *i* для каждой клиентской последовательности (обозначим номер клиента как *j*), проверяется, может ли данный предмет быть добавлен на позицию `borderArray[j]`, образуя новую частую последовательность.

Таблица `ITEM_IS_EXIST_TABLE` может храниться в более компактном виде за счет того, что не все строки несут полезную информацию. Предлагается хранить только те строки, которые отличаются от предыдущей (за исключением строк из нулей). Этого достаточно для того,

чтобы знать значения всех битов во всех строках. Оптимизированная таблица проиллюстрирована на рис.8.

Item / Pos	a	b	c	d
5	1	0	0	1
6	0	0	0	1

Рис. 8. Оптимизированная часть таблицы ITEM_IS_EXIST_TABLE

Last Position Induction SPAM позволяет избежать операций побитового “И” и превосходит SPAM по скорости в 2-3 раза на любых объемах данных.

Как и в алгоритмах SPADE и SPAM, время работы LPI SPAM возрастает линейно с ростом числа клиентских последовательностей.

2.3. Алгоритмы вида «pattern-growth»

Другим подходом является поиск шаблонов последовательных событий на основе проецирования базы и увеличения длины шаблонов (projection-based sequential pattern-growth approach). Существуют два известных алгоритма, использующие данный подход: FreeSpan[3] и PrefixSpan[4].

PrefixSpan является «развитием» алгоритма FreeSpan и превосходит его по скорости исполнения, а также использует меньше памяти, поэтому далее будет детально описан только этот алгоритм.

Описание алгоритма PrefixSpan

Поскольку порядок предметов в одной транзакции не имеет значения, предполагается, что предметы расположены в лексикографическом порядке.

Введем несколько терминов.

Определение 1. Для заданной последовательности $s = <e_1, e_2, ..., e_n>$ последовательность $p = <e'_1, e'_2, ..., e'_m>$ ($m <= n$) называется префиксом s, если и только если:

1. $e'_i = e_i$ для $i <= m - 1$
2. e'_m содержится в e_m
3. Все предметы из ($e_m - e'_m$) лексикографически идут после любого элемента из e'_m.

Пример: $<a>$, $<a, (a, b)>$ являются префиксами $<a, (a, b, c), (a, c), d, (c, f)>$, в то время как $<a, b>$ - нет.

Определение 2. Рассмотрим последовательность $s = <e_1, e_2, ..., e_n>$. Пусть последовательность $p = <e'_1, e'_2, ..., e'_m>$ ($m <= n$) является префиксом s. Тогда последовательность $q = <e''_m, e'_{m+1}, ..., e'_n>$, где $e''_m = (e_m - e'_m)$, называется суффиксом s, соответствующим префиксу p, и обозначается s/p. В данном случае s обозначается как $p*q$.

Пример: для последовательности $<a, (a, b, c), (a, c), d, (c, f)>$ последовательность $<(a, b, c), (a, c), d, (c, f)>$ является суффиксом, соответствующим префиксу $<a>$.

Лемма 1. Если s – шаблон последовательных событий длины l и $\{b_1, b_2, ..., b_m\}$ – множество всех шаблонов последовательных событий длины l+1 с префиксом s, то множество всех шаблонов с префиксом s, за исключением самого s, может быть разбито на m непересекающихся подмножеств. При этом j-ое подмножество для всех j, таких что $1 <= j <= m$,

является множеством всех шаблонов последовательных событий с префиксом b_j. Данный факт доказывается в работе Пея[4].

Определение 3. Пусть *s* – шаблон последовательных событий в базе транзакций *D*. *S*-проецированной базой (*s*-проекцией), обозначающейся как $D|_s$, называется множество суффиксов в *D*, соответствующих префиксу *s*.

Определение 4. Пусть *s* – шаблон последовательных событий в базе транзакций *D*, а *q* – последовательность из *D* с префиксом *s*. Поддержкой *q* в *s*-проецированной базе $D|_s$ называется число последовательностей *p* из $D|_s$, таких что *q* содержится в *s*p*.

Идея алгоритма PrefixSpan заключается в том, чтобы найти в заданной базе все частые предметы и добавить их к текущему шаблону, получая тем самым новые частые последовательности, а после этого искать частые последовательности большей длины на основе проецированных баз. Псевдокод алгоритма представлен в Листинге 2:

PrefixSpan(s, $D|_s$):

Параметры: s – шаблон последовательных событий;

$D|_s$ – s-проекция исходной базы D, если s не является пустой последовательностью <>, иначе $D|_s$ = D.

1. Найти все частые предметы b из $D|_s$, такие что:
   А) b может быть добавлен к последнему предметному набору из s, образуя частую последовательность;
   Б) предметный набор (b) может быть добавлен в s, образуя частую последовательность.
2. Для каждого частого предмета b:
   Добавить b в s, образуя новый шаблон s';
   Добавить s' в результат;

```
3.Для каждого нового шаблона s′:
    Построить s′-проекцию D|s′;
    Вызвать PrefixSpan(s′, D|s′);
```

Листинг 2. Алгоритм PrefixSpan

Для того чтобы найти все шаблоны последовательных событий в базе D, необходимо вызвать `PrefixSpan(<>, D)`.

Создание проецированных баз может сильно сказаться на производительности при работе с большими объемами данных, поэтому вместо физического создания проекций используется псевдопроецирование (pseudoprojection). При рекурсивном вызове метода `PrefixSpan` вместо созданной проекции ему передаются указатели на минимальные позиции возможных вхождений предметов в клиентские последовательности после текущего шаблона. В качестве указателя рассматривается набор, состоящий из идентификатора клиента, номера транзакции в клиентской последовательности и позиции в транзакции. Благодаря псевдопроекции, скорость работы алгоритма значительно повышается. Кроме того, для работы алгоритма требуется значительно меньше памяти.

Как показано в работе Пея[4], алгоритм PrefixSpan с использованием псевдопроекции значительно эффективнее GSP, SPADE и FreeSpan.

2.4. Итоги обзора

В данном обзоре были рассмотрены основные методы решения задачи SPM. Среди рассмотренных алгоритмов наиболее эффективными являются

Last Position Induction SPAM и PrefixSpan. Данные алгоритмы с большим преимуществом выигрывают у своих конкурентов по скорости работы. Тем не менее, они не позволяют учитывать такие параметры как минимальное и максимальное время между транзакциями, тем самым оказываются неприменимы для решения некоторого класса задач. Одним из примеров такой задачи является ситуация с книжным магазином, описанная при рассмотрении алгоритма GSP. Помимо этого, данные алгоритмы не позволяют задать в качестве параметра максимальную длину шаблонов, а также получить в качестве результата только максимальные частые последовательности, что является важным для некоторых задач. Например, при прогнозировании аварий или последствий природных катаклизмов требуется учитывать все частые последующие события, а для этого необходимо использовать список максимальных частых последовательностей.

По результатам исследования алгоритмов решения задачи SPM было решено реализовать алгоритмы Last Position Induction SPAM и PrefixSpan в рамках проекта GDLogTool. При этом они должны быть адаптированы для возможности задания в качестве параметров алгоритма минимального и максимального времени между транзакциями, максимальной длины шаблона, а также позволять получать в качестве результата только максимальные частые последовательности.

3. Реализация и сравнительный анализ выбранных алгоритмов решения задачи SPM

Как уже было отмечено, в результате исследования алгоритмов решения задачи SPM было принято решения реализовать алгоритмы Last Position Induction SPAM и PrefixSpan, расширив перед этим области их применения.

Первым усовершенствованием алгоритмов является добавление возможности ограничения минимального и максимального интервалов времени между соседними событиями шаблонов.

Решение для LPI SPAM:

Для того чтобы учитывались ограничения по времени между соседними событиями шаблонов, необходимо несколько изменить процесс вычисления поддержки кандидатов. На каждой итерации обхода дерева нужно дополнительно хранить значение позиции транзакции, на которую был добавлен предмет на предыдущей итерации алгоритма. Далее, поддержка кандидата увеличивается лишь в том случае, если разница между временем транзакции, соответствующей предыдущей позиции, и временем транзакции, соответствующей текущей позиции, не превосходит заданного максимального значения, а также не меньше заданного минимального значения. Кроме того, добавляется возможность вводить ограничения не только по времени, но и по числу событий, прошедшим в промежутке между соседними элементами шаблона. В данном случае сравниваются не времена транзакций, соответствующие позициям, а сами позиции. Стоит отметить, что дополнительная проверка при вычислении поддержки кандидата происходит только для S-step. В случае I-step новый предмет добавляется в

последний предметный набор имеющегося шаблона, а не как отдельный предметный набор, состоящий из одного предмета, в связи с этим предыдущая позиция будет совпадать с текущей, и получившаяся последовательность будет удовлетворять заданным ограничениям, поскольку им удовлетворяет шаблон.

Решение для PrefixSpan:

Для данного алгоритма ограничения по времени между соседними событиями шаблонов вводятся аналогичным образом. При выполнении метода `PrefixSpan` происходит дополнительная проверка при вычислении поддержки предметов. Помимо позиций возможных вхождений предметов в клиентские последовательности, методу передается набор номеров транзакций, в которые входит последний предмет, добавленный в текущий шаблон. Увеличение поддержки предмета происходит лишь в том случае, если разница во времени между двумя последними транзакциями последовательности, полученной путем добавления в шаблон нового предметного набора, состоящего из этого предмета, удовлетворяет заданным временным ограничениям. Для того чтобы учитывать ограничения по числу событий, прошедшим в промежутке между соседними элементами шаблона, производится аналогичная проверка, но проверяется не разница во времени между транзакциями, а разница их позиций в клиентской последовательности.

Помимо добавления возможности задания временных ограничений, необходимо предоставлять в качестве результата работы алгоритма как все шаблоны последовательных событий, так и только максимальные из них. Данная возможность реализуется следующим образом.

Для хранения максимальных шаблонов используется дополнительный список. Изначально этот список пуст. При нахождении нового шаблона его дальнейшая обработка происходит в два шага:

1) Проверка на максимальность: если ни один шаблон из списка не содержит данный шаблон, то он является максимальным;
2) Если шаблон оказался максимальным, происходит удаление всех шаблонов из списка, содержащихся в новом шаблоне, после чего он добавляется в список.

Для сокращения числа кандидатов на добавление в список максимальных шаблонов используется тот факт, что основные методы обоих выбранных алгоритмов (`DFS-Pruning` и `PrefixSpan`) являются рекурсивными. Напомним, что в качестве результата методы возвращают объединение набора шаблонов, найденных в соответствии с заданными параметрами, и результатов последующих рекурсивных вызовов. После нахождения нового шаблона *s* происходит проверка мощности множества шаблонов, найденных с помощью рекурсивного вызова основного метода алгоритма с параметром *s*. Если полученное множество оказывается не пустым, то существует шаблон, для которого *s* является префиксом. Этот факт означает, что *s* не является максимальным шаблоном, поэтому он не проверяется дополнительно и не добавляется в список максимальных шаблонов.

После завершения поиска шаблонов последовательных событий созданный список будет содержать все максимальные шаблоны и только их.

Последним усовершенствованием выбранных алгоритмов является добавление возможности задания максимальной длины шаблонов.

Для ограничения максимальной длины шаблонов производится проверка в самом начале рекурсивного вызова основных методов алгоритмов. Если длина переданного в качестве параметра шаблона оказывается равной заданному граничному значению, то выполнение метода заканчивается, а в качестве результата возвращается пустой список.

В обзоре существующих решений задачи SPM было указано, что выбранные алгоритмы имеют большую степень эффективности по сравнению с остальными алгоритмами. Остается открытым вопрос о сравнении производительности выбранных алгоритмов между собой. После реализации LPI SPAM и PrefixSpan было проведено тестирование этих алгоритмов на различных объемах данных, а также с различными значениями параметров алгоритмов. Тестирование проводилось на компьютере с процессором Intel Core i5 с тактовой частотой 2.8GHz и 4Gb оперативной памяти.

Для тестирования алгоритмов были созданы базы журналов с различными показателями среднего числа транзакций(T) в клиентских последовательностях, среднего числа предметов в транзакциях(I), числа различных предметов(N), а также количества клиентских последовательностей(C). Было выделено 4 типа баз в соответствии со степенью их разреженности: сильно плотные(T=30 I=10 N=200, Very Dense), плотные(T=10 I=10 N=200, Dense), разреженные(T=10 I=5 N=1000, Sparse), сильно разреженные(T=10 I=1 N=10000, Very Sparse). Для тестирования использовались базы этих четырех типов, состоящие из 10000 последовательностей, а также базы с теми же характеристиками, состоящие из 100000 последовательностей. На рис.9 показаны результаты работы алгоритмов для четырех баз, состоящих из 100000 последовательностей, при

пороговом значении поддержки равном 1% от числа последовательностей. На рис.10 представлены аналогичные результаты для баз, включающих 10000 клиентских последовательностей.

Вертикальные оси графиков соответствуют значениям времени работы алгоритмов в секундах. Значениями на оси абсцисс являются базы журналов четырех типов.

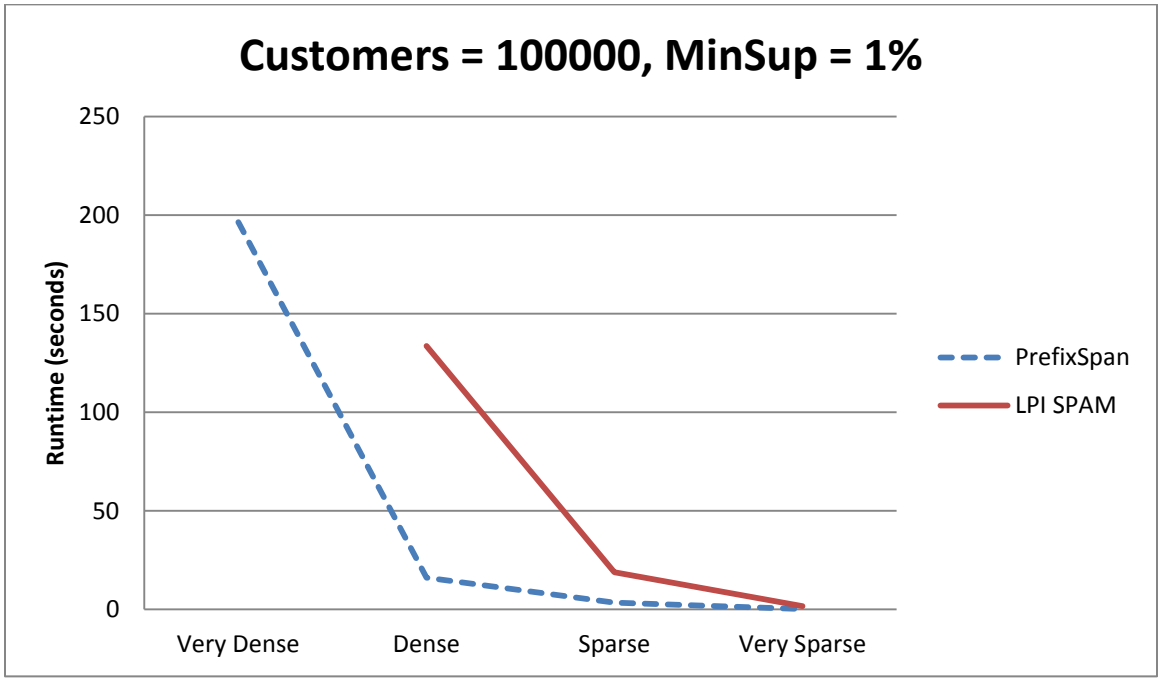

Рис. 9. Результаты работы алгоритмов для больших баз журналов

При тестировании алгоритма LPI SPAM в случае сильно плотной базы, содержащей 100000, для выполнения работы алгоритма не хватило оперативной памяти, поэтому результат не отображен на графике.

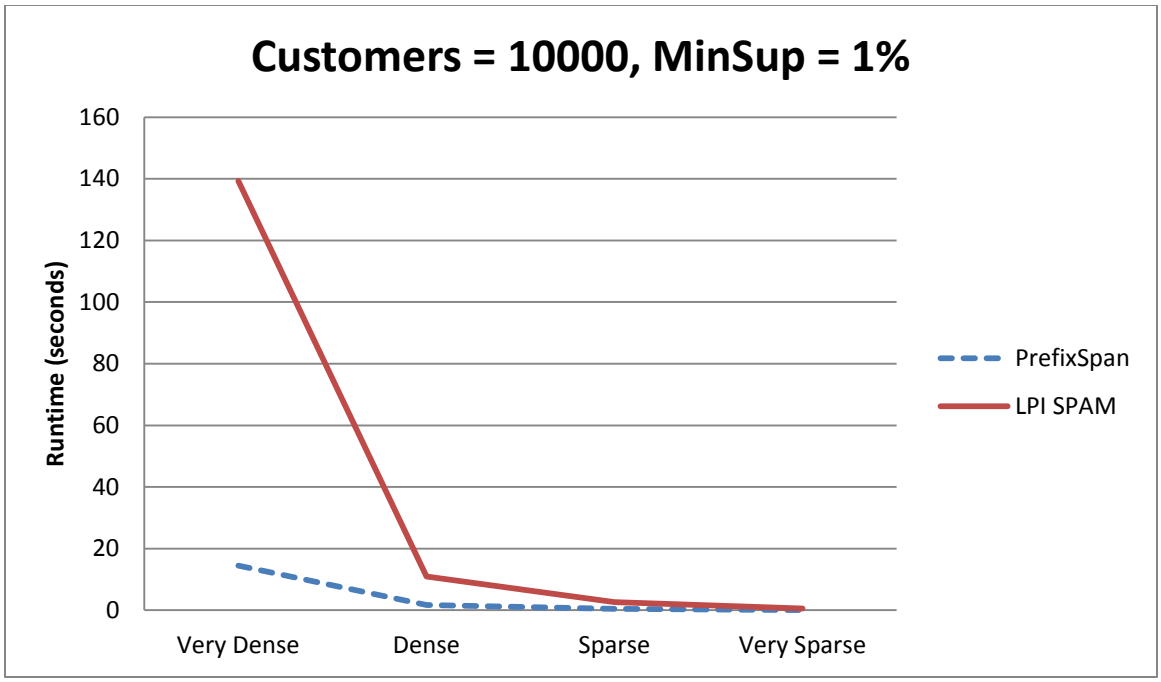

Рис. 10. Результаты работы алгоритмов для баз журналов, включающих 10000 клиентских последовательностей

Как видно из графиков, алгоритм PrefixSpan имеет лучшие показатели по сравнению с LPI SPAM на всех созданных базах, поскольку графики, соответствующие PrefixSpan, лежат ниже графиков, соответствующих LPI SPAM.

На рис.11 проиллюстрированы графики работы алгоритмов для базы журналов, содержащей 100000 последовательностей, и имеющей следующие характеристики: T=30, I=10, N=1000. Вертикальная ось обозначает время работы алгоритмов в секундах, а на горизонтальной оси расположены пороговые значения поддержки, выраженные в процентном соотношении к числу последовательностей в базе.

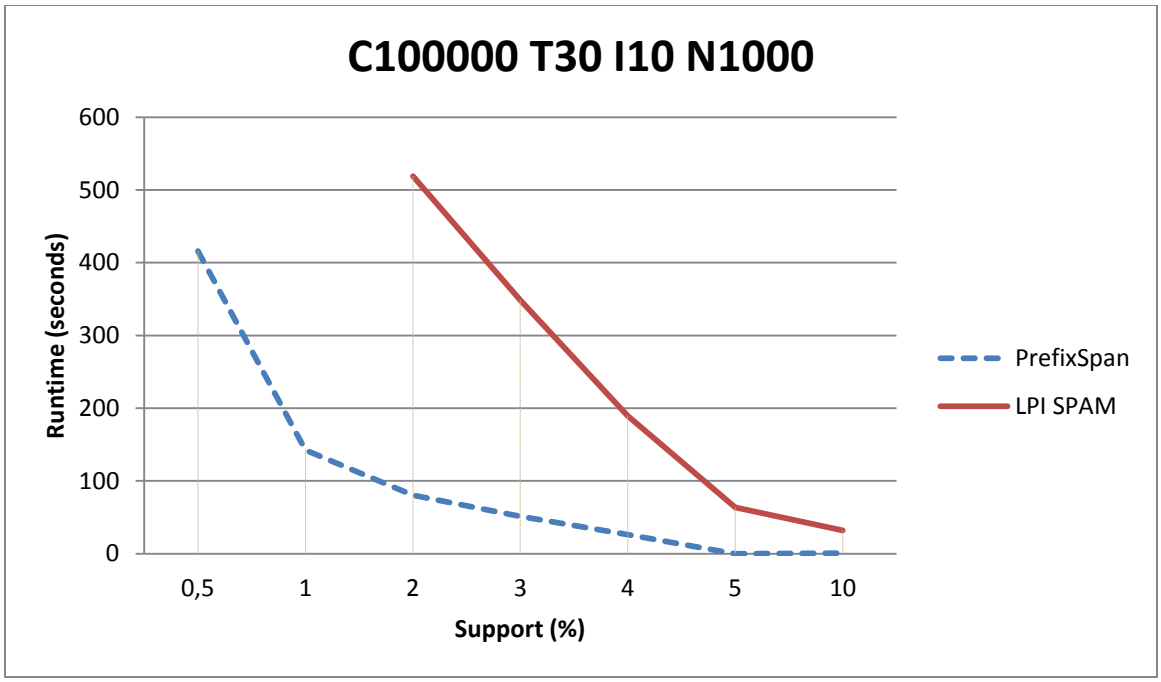

Рис. 11.Результаты работы алгоритмов для разных значений минимальной поддержки

График, изображенный на рис.11, отражает тот факт, что алгоритм PrefixSpan превосходит LPI SPAM по скорости для разных значений минимальной поддержки. Для выполнения работы алгоритма LPI SPAM для значений минимальной поддержки, соответствующих 1% от числа последовательностей в базе и менее, оказалось недостаточно оперативной памяти, поэтому данные результаты не отображены на графике.

В результате тестирования работы алгоритмов с различными значениями минимальной поддержки для других описанных ранее баз журналов было выявлено, что PrefixSpan быстрее LPI SPAM в каждом рассматриваемом случае. Более того, алгоритм PrefixSpan потребляет меньше памяти.

Полученные результаты можно обосновать тем фактом, что LPI SPAM вначале своей работы создает таблицы списков вхождений предметов в клиентские последовательности, а также таблицу `ITEM_IS_EXIST_TABLE`,

что может занять чрезвычайно много времени и памяти при больших объемах данных. В то же время, алгоритм PrefixSpan не требует предварительных вычислений перед поиском шаблонов, а также на каждой следующей итерации поиск производится в меньшей по размерам структуре, что обеспечивает эффективную работу алгоритма.

В результате сравнения производительности алгоритмов LPI SPAM и PrefixSpan были сделаны выводы о том, PrefixSpan является наиболее эффективным алгоритмом решения задачи SPM. Он показал лучшие результаты по скорости исполнения и потребляемой памяти по сравнению с LPI SPAM. На основе полученных результатов было принято решение об использовании алгоритма PrefixSpan для поиска шаблонов последовательных событий.

4. Описание механизма определения типов журналов

Для разных приложений формат записи журналов может сильно различаться. Чтобы обеспечить возможность решения задачи SPM для различных предметных областей, нужен механизм определения типов журналов, с помощью которого будет возможно извлекать необходимую информацию из полученных сообщений. Далее будет детально описан разработанный механизм и его применение.

Во-первых, введем понятие *токена*. Токен представляет собой некоторое регулярное выражение и обозначается `@token_name;`, где `token_name` - имя данного токена. Токены необходимы для того, чтобы обозначать части сообщения о событии, содержащие информацию, которая в последствии будет обработана алгоритмом поиска шаблонов последовательных событий. Еще одно понятие, которое нам понадобится, это понятие *шаблона типа журналов*. Шаблоном типа журналов будем называть запись, представляющую собой регулярное выражение, возможно, дополненное токенами. Приведем пару примеров таких шаблонов:

а) `Error\sat\snode\s\d*`
б) `User\s@UserId;\sbought\sitem\s@ItemId;`

Первый пример задает тип журналов, содержащих информацию о номере узла, на котором произошла ошибка, в виде «Error at node N», где N – номер узла. Второй пример является упрощенным шаблоном для журналов о

покупке товара пользователем вида «User N bought item K», где N – идентификатор клиента, а K – имя или код товара.

В качестве вида регулярных выражений был выбран класс Perl5-совместимых регулярных выражений, поскольку данный вид имеет более богатый и в то же время предсказуемый синтаксис, чем POSIX и даже POSIX ERE. Для работы с регулярными выражениями использовалась библиотека Jakarta ORO[9], разработанная Apache.

Для описания шаблонов типов журналов используются два файла. В одном из них содержится определение всех токенов, а в другом - определение самих шаблонов типов журналов. В обоих файлах определение одного элемента - токена или шаблона - представляет собой строку вида:

ИмяЭлемента = ОписаниеЭлемента

Примеры файлов с определениями токенов и шаблонов типов журналов показаны на рис.12 и рис.13:

token.conf:

```
dotted_time = \d\d:\d\d:\d\d
port = \d*
file_name = "[a-z]*"
line = \d*
host = \d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}
```

Рис. 12. Пример файла, содержащего определения токенов

processors.conf:

```
connect = Connect to @host;:@port;
exception = @dotted_time;: Exception occurred in file
@file_name; at line @line;
sells = user@userId;\sbought\s@items;
webLogGet = @host;-\s@time;\s"GET\s.*"\s"@browser;.*"
```

Рис. 13. Пример файла, содержащего определения шаблонов типов журналов

На основе имеющихся файлов создается список *процессоров.* Каждому шаблону типа журналов соответствует один процессор. Процессор представляет собой объект, содержащий в качестве атрибутов имя шаблона и его определение. Кроме того, процессор содержит набор токенов соответствующего шаблона.

При поступлении нового журнала в базу начинается его обработка. Происходит проверка принадлежности журнала какому-либо типу журналов из списка процессоров. Сообщение журнала обрабатывается процессорами последовательно. Если журнал относится к типу, заданному некоторым процессором, то он помечается соответствующим *тегом* - атрибутом, характеризующим принадлежность к определенному типу журналов. Тег представляет собой имя данного процессора. Далее, для быстрого и удобного поиска все атрибуты журнала индексируются в поисковой движок Solr[13].

Рассмотрим пример обработки журнала на основе файлов, показанных на рис.14.

```
userId = \d+
items = (([a-z]|\d)+,)*([a-z]|\d)+
host = \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
browser = .*/(\d+(\.(\d)+)*)
```

token.conf

```
sells = user@userId;\sbought\s@items;
webLogGet = @host;\s-\s-\s@time;\s"GET\s.*"\s"@browser;.*"
```

processors.conf

Рис. 14. Примеры файлов token.conf и processors.conf

Допустим, в систему поступил журнал с сообщением `Log.message` = «user10 bought tv,phone».

На рис.15 проиллюстрирована обработка журнала всей цепочкой процессоров:

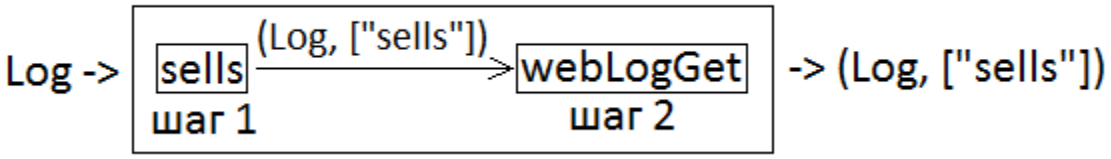

Рис. 15. Обработка журнала процессорами

На шаге 1 происходит обработка журнала процессором с именем «sells».

Поскольку сообщение журнала подходит под шаблон, заданный этим процессором, в результате обработки к журналу добавляется соответствующий тег.

На шаге 2 происходит обработка журнала процессором с именем «webLogGet».

Сообщение журнала не подходит под заданный этим процессором шаблон, поэтому список тегов журнала остается неизменным.

На этом обработка заканчивается, поскольку все процессоры закончили свою работу. В результате обработки к журналу был добавлен список из одного элемента «sells».

Для поиска шаблонов последовательных событий необходимо знать, журналы какого типа исследовать, а также какие токены соответствуют идентификатору клиента и предметному набору. Данные параметры должны быть заданы пользователем. В качестве времени транзакции (времени события) используется время отправки журнала из приложения в GDLogTool.

Если в ходе обработки оказалось, что журнал принадлежит типу, заданному пользователем, происходит извлечение данных из этого журнала. В процессоре, соответствующем указанному типу, выбираются токены идентификатора клиента и предметного набора, имена которых также задаются пользователем. Благодаря представлению в виде регулярных выражений, получение значений токенов не составляет труда. После этого полученные значения добавляются в структуру для поиска шаблонов последовательных событий.

Структура для поиска шаблонов последовательных событий проиллюстрирована на рис.16:

$UserId_1$	$Time_1$	$item_{11}, item_{12}, ..., item_{1k}$
	$Time_2$	...
	...	...
	$Time_{N1}$	...
...	...	...
$UserId_M$	$Time_{M1}$	$item_{M11}, item_{M12}, ...$
	$Time_{M2}$	...
	...	...
	$Time_{MX}$	...

Рис. 16. Структура для хранения информации журналов

Все элементы структуры хранятся в виде чисел для более эффективного использования памяти, поэтому перед сохранением информации из журнала происходит трансформация ее представления из строкового в числовое.

На рис.17 продемонстрирован процесс извлечения информации из журнала и ее последующего сохранения в структуре для поиска шаблонов последовательных событий на примере процессора «sells»:

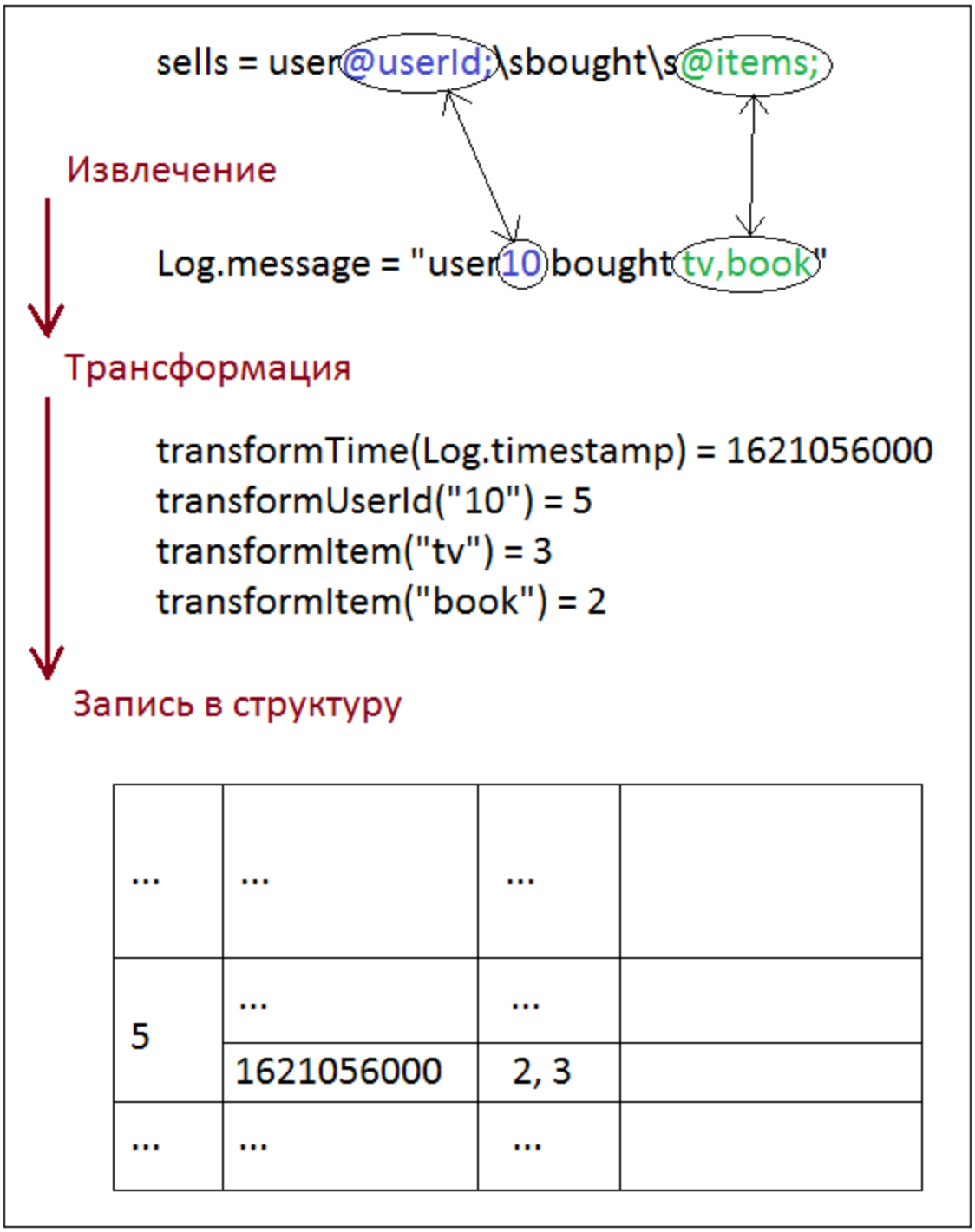

Рис. 17. Процесс извлечения информации из журнала и ее сохранения в структуру для поиска шаблонов

Отметим, что после трансформации предметов из строкового представления в числовое они записываются в структуру в порядке возрастания трансформированных значений. Данный порядок может отличаться от порядка предметов, записанных в журнале, но поскольку предметы содержатся в одной транзакции, порядок их записи не влияет на результаты поиска шаблонов последовательных событий. В то же время, запись предметов в порядке возрастания требуется для повышения эффективности работы алгоритма решения задачи SPM.

Аналогичным образом обрабатываются все собираемые журналы. Помимо этого, при первом запуске процесса поиска шаблонов последовательных событий, а также при запуске с изменившимися пользовательскими параметрами происходит обработка всех журналов заданного типа, имеющихся в базе.

После завершения обработки и формирования структуры для поиска шаблонов последовательных событий алгоритм решения задачи SPM может быть применен к данной структуре.

5. Выбранные алгоритмы и результаты их работы в проекте GDLogTool

5.1. Описание проекта GDLogTool

GDLogTool представляет собой веб-сервис для работы с журналами. Журналы могут приниматься с различных источников по протоколам TCP и UDP. На текущий момент реализованы возможности просмотра, сортировки и поиска по журналам, подсчет некоторых статистик, а также уведомления о поступлении журналов в соответствии с заданными фильтрами. Основным языком программирования, на котором ведется разработка GDLogTool, является Java. Веб-интерфейс системы реализован с использованием JavaScript библиотеки ExtJS.

GDLogTool позволяет принимать журналы, записанные с помощью SocketAppender и SyslogAppender. Помимо этого, реализованы два агента: file-tailer agent, который отправляет журналы, записываемые каким-либо приложением в опредленный файл, и ssh-tailer agent, который выполняет ту же работу для удаленного хоста. Архитектура GDLogTool представлена на рис.18.

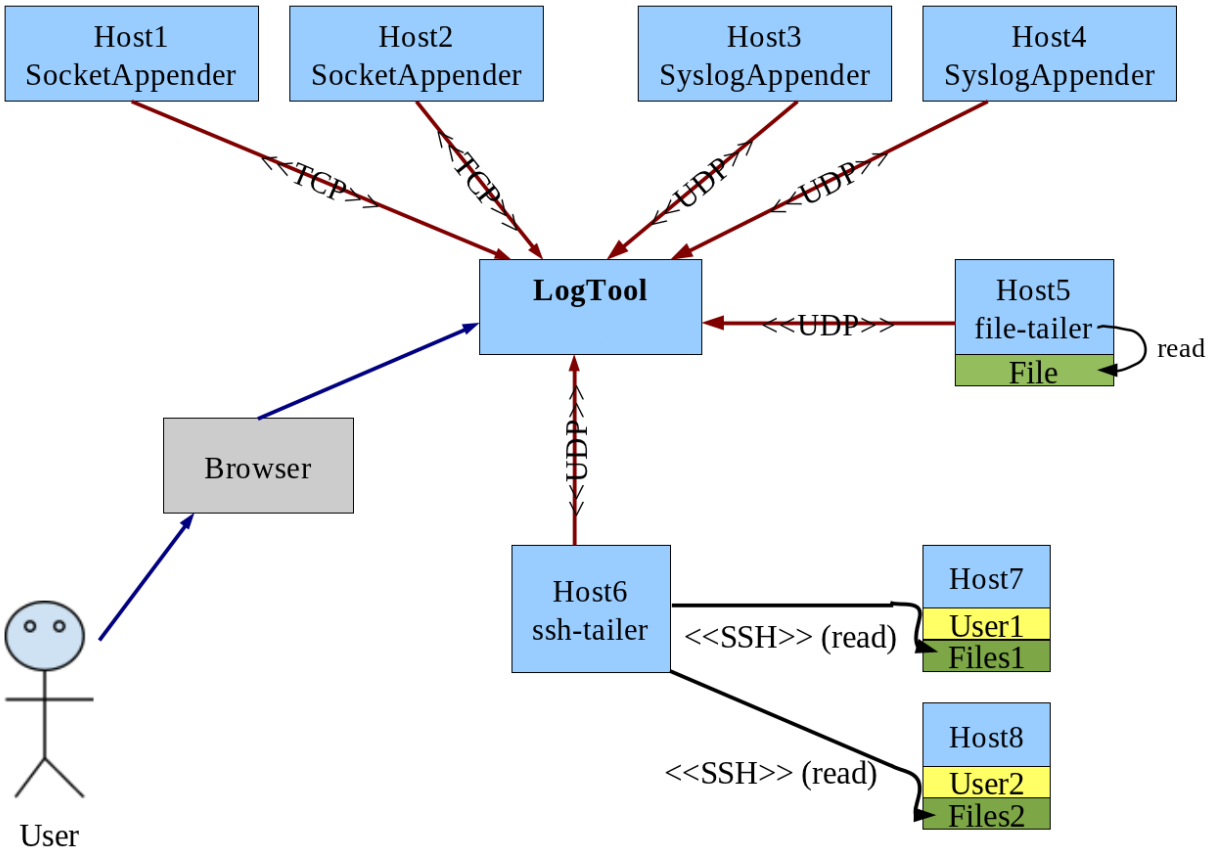

Рис. 18. Архитектура GDLogTool

Собранные журналы хранятся в файловой системе в виде дерева со структурой application/host/instance/date. Предоставляется возможность задания размера хранилища. При достижении размера хранилища заданного значения последующие поступающие в GDLogTool журналы будут записаны вместо наиболее старых.

С помощью графического интерфейса пользователь может просматривать и удалять журналы, а также производить некоторые поисковые запросы. В интерфейсе присутствует навигация по базе журналов и по результатам поисковых запросов. Присутствуют возможности разбиения на страницы, сортировки и ограничения по количеству журналов, полученных в результате поискового запроса, а также сохранения ссылки на

поисковой запрос и его выполнения с помощью REST API. Интерфейс GDLogTool представлен на рис.19:

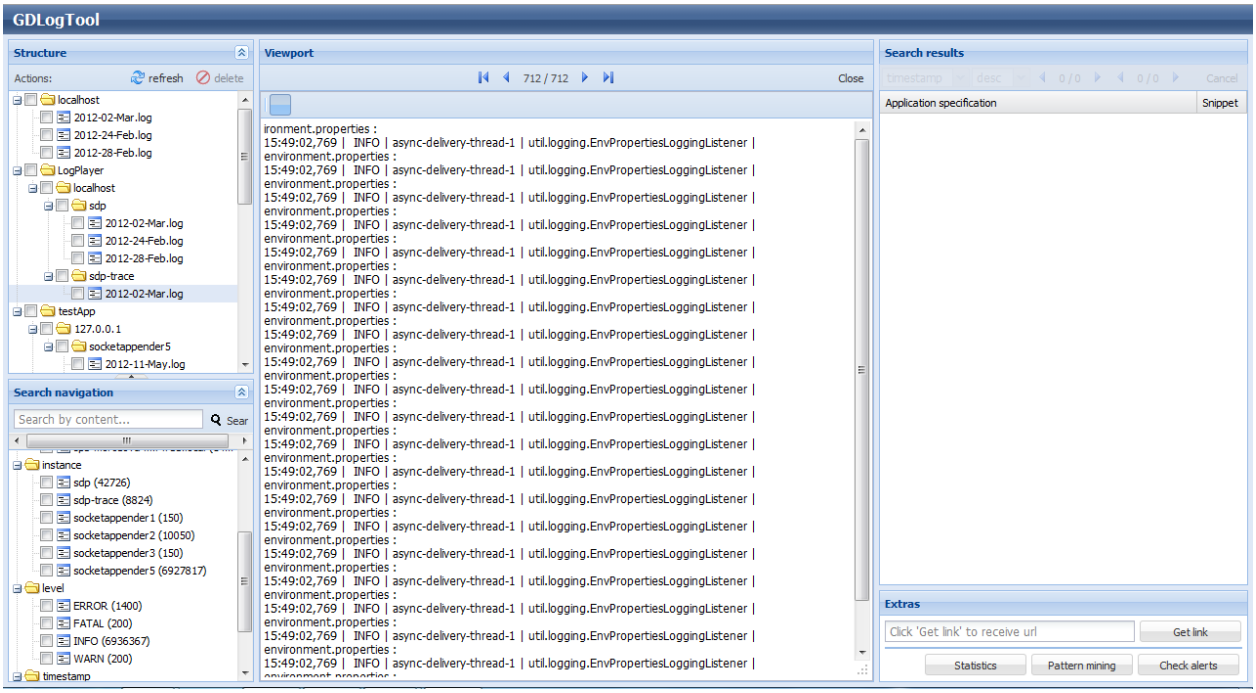

Рис. 19. Интерфейс GDLogTool

Помимо описанных возможностей, GDLogTool позволяет получить некоторую статистику относительно количества журналов с заданными параметрами. Данная функция проиллюстрирована на рис.20:

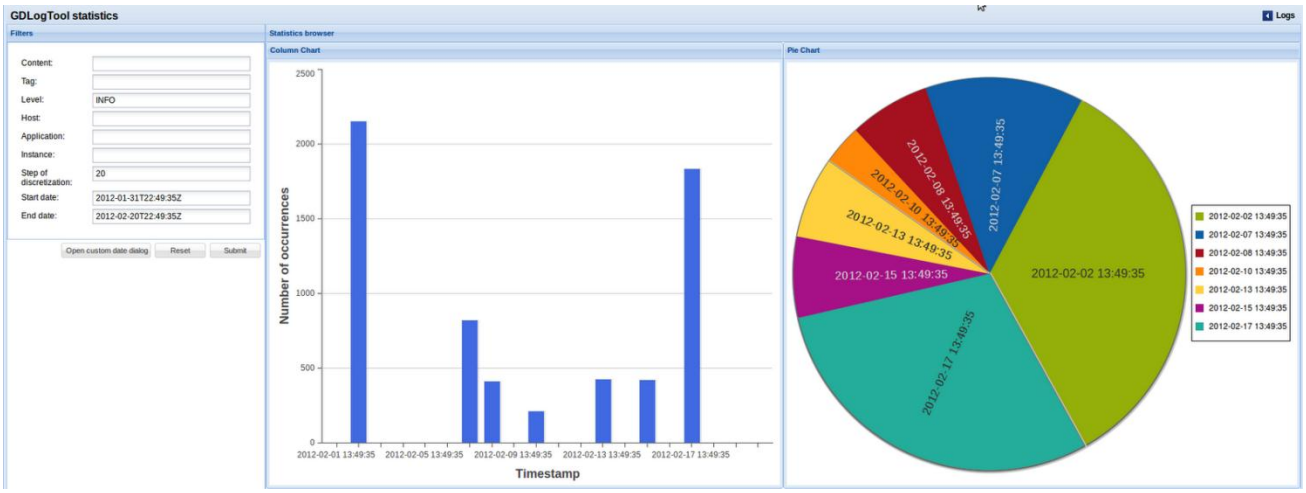

Рис. 20. Расчет статистик в GDLogTool

Из описанной функциональности лично автором были реализованы: расчет статистик, механизм сбора и обработки журналов, веб-интерфейс системы, а также некоторая часть функционала, связанная с поисковыми запросами.

GDLogTool является бесплатным инструментальным средством для работы с журналами. Среди аналогичных средств можно выделить Loggly[10], LogStash[11] и Splunk[14], однако, GDLogTool имеет некоторые преимущества по сравнению с ними. Во-первых, стоит отметить, что Loggly и Splunk являются коммерческими проектами, а их бесплатные версии накладывают ограничения на размер базы журналов. Loggly является cloud-based сервисом, что негативно сказывается на безопасности хранения журналов. Кроме того, проблемы с соединением могут привести к потере сообщений, а загрузка большого количества журналов может занять длительное время. Бесплатная версия Splunk не предоставляет возможность уведомлений о поступлении журналов, соответствующих заданным фильтрам, а коммерческая версия является дорогостоящей. LogStash удобен в

использовании, но позволяет только хранить и просматривать журналы, а также производить поисковые запросы. Данный инструмент не предоставляет возможности расчета статистик и задания размера хранилища, для освобождения памяти требуется удалять журналы вручную.

5.2. Применение выбранных алгоритмов и результатов их работы в проекте GDLogTool

Алгоритмы LPI SPAM и PrefixSpan были реализованы в рамках проекта GDLogTool. Для поиска шаблонов последовательных событий используется PrefixSpan, поскольку данный алгоритм показывает лучшие результаты по скорости исполнения и потребляемой памяти по сравнению с LPI SPAM. В качестве параметров алгоритма пользователь может задать имя процессора шаблона типов журналов, разделитель для корректного разбиения предметных наборов на предметы, значение минимальной поддержки, временные ограничения или ограничения на число событий, произошедших в промежутках между соседними элементами шаблонов, а также максимальную длину шаблонов. Кроме того, пользователю предоставляется возможность просмотра всех или только максимальных шаблонов. Интерфейс для поиска шаблонов последовательных событий представлен на рис.21:

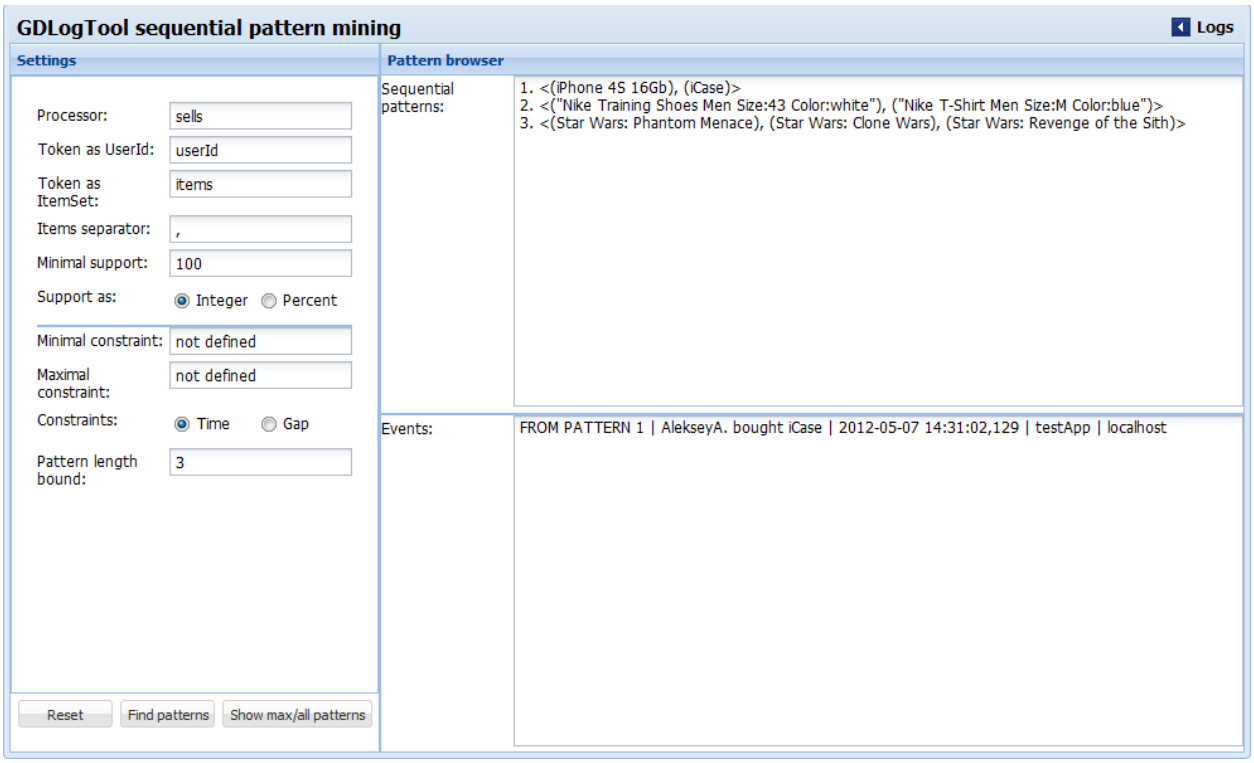

Рис. 21. Поиск шаблонов последовательных событий в GDLogTool

После завершения поиска шаблонов GDLogTool производит уведомления о поступивших событиях, соответствующих какому-либо шаблону. События отображаются в форме «Events» с указанием номера шаблона и дополнительной информацией. Благодаря уведомлениям, становится возможным быстро реагировать на поступившие события, входящие в шаблоны, а также использовать полученную информацию для создания поисковых запросов и расчета статистик.

Кроме того, для автоматизации процесса поиска шаблонов и использования результатов поиска в других приложениях была добавлена поддержка REST API[5, 12]:

Путь: /rest/mining
Тип: GET

Тело запроса представляется в виде JSON-объекта и содержит параметры, описанные в таблице 1:

Имя	***Тип***	***Описание***	***Является ли обязательным***
processor	String	Имя процессора для формирования базы журналов	Да
userIdToken	String	Имя токена, соответствующего идентификатору клиента	Да
itemsetToken	String	Имя токена, соответствующего предметному набору	Да
Separator	String	Разделитель предметов в предметном наборе	Да
minSup	Integer	Пороговое значение поддержки	Да
supportAsInteger	Boolean	Считать пороговое значение поддержки как число клиентов (если значение `true`) или как их процентное соотношение (если значение `false`)	Да
isTimeConstraint	Boolean	Использовать ограничения по времени (`true`) или по числу	Нет

		событий (`false`)	
minConstraint	String	Значение минимального ограничения	Нет
maxConstraint	String	Значение максимального ограничения	Нет
maxPatternLength	Integer	Максимальная длина шаблонов	Нет
showAllPatterns	Boolean	Отображать все шаблоны (`true`) или только максимальные (`false`)	Да

Таблица 1.Параметры запроса

Для задания ограничений по времени используются следующие обозначения: `y` – год, `mon` – месяц, `d` – день, `h` – час, `m` – минута, `s` – секунда, `ms` – миллисекунда. Перед соответствующим идентификатором записывается его числовое значение. Идентификаторы вместе со значениями отделяются друг от друга пробелами. Например, задание ограничения в 3 месяца 2 дня и 5 секунд выглядит следующим образом: «3`mon` 2`d` 5`s`». Для задания ограничений на число событий между соседними транзакциями используются числовые значения.

В качестве ответа на запрос возвращается массив строк, состоящий из шаблонов, найденных в соответствии с заданными параметрами.

Заключение

По итогам данной дипломной работы были получены следующие результаты:

1. Исследованы существующие методы решения задачи SPM.
2. Выбраны два наиболее эффективных алгоритма решения задачи SPM, их области применения были расширены за счет добавления возможностей задания временных ограничений для последовательностей и ограничений на количество событий, произошедших в промежутке между соседними элементами последовательностей, ограничений длины шаблона, а также возможностью получить только максимальные последовательности.
3. Разработан механизм определения типов журналов.
4. Данный механизм и алгоритмы реализованы в рамках проекта GDLogTool; для поиска шаблонов последовательных событий используется алгоритм PrefixSpan, поскольку по итогам тестирования он оказался более эффективным по сравнению с LPI SPAM.

Благодаря механизму определения типов логов, задача SPM может быть решена в контексте различных предметных областей и для разных целей. Функциональность GDLogTool позволяет использовать результаты работы алгоритма поиска шаблонов последовательных событий для вычисления некоторых статистик и создания поисковых запросов. Кроме того, реализована поддержка уведомлений о поступивших событиях, входящих в найденные шаблоны, что позволяет быстро реагировать на соответствующие события, а также REST API для автоматизации процесса поиска шаблонов и использования результатов поиска в других приложениях.

На данный момент GDLogTool активно развивается. Разрабатываемое инструментальное средство может быть использовано как отдельное приложение, а также как часть системы. Планируется внедрение GDLogTool в одну из систем электронной коммерции.

Список литературы

[1] *R. Agrawal, R. Srikant*. Mining Sequential Patterns. In Proc. of the 11th Int'l Conference on Data Engineering, 1995.

[2] *J. Ayres, J. Flannick, J. Gehrke, T. Yiu*. Sequential Pattern Mining using a Bitmap Representation. In ACM SIGKDD Conference, 2002, P.429-435.

[3] *J. Pei, J. Han, Q. Chen, U. Dayal, H. Pinto.* FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. In Proc. of the Int. Conf. Knowledge Discovery and Data Mining, 2000.

[4] *J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu*. PrefixSpan: mining sequential patterns efficiently by prefix projected pattern growth. In ICDE, 2001, P. 215–226.

[5] *L. Richardson, S. Ruby*. RESTful Web Services: Web Services for the Real World. O'Reilly Media, 2007, 454 p.

[6] *R. Srikant, R. Agrawal*. Mining Sequential Patterns: Generalizations and Performance Improvements. In EDBT, 1996.

[7] *Z. Yang, M. Kitsuregawa*. LPI-SPAM: An Improved Algorithm for Mining Sequential Pattern. In Proc. of Int'l Special Workshop on Databases for Next Generation Researchers in conjunction with ICDE'05, 2005, P. 8-11.

[8] *M.J. Zaki*. SPADE: An Efficient Algorithm for Mining Frequent Sequences. In Machine Learning Journal, Vol. 42(1/2), 2001, P. 31-60.

[9] Jakarta ORO, Apache Project. URL: http://jakarta.apache.org/oro/

[10] Loggly. URL: http://www.loggly.com/

[11] LogStash. URL: http://logstash.net/

[12] RESTful Web services: The basics. URL: http://www.ibm.com/developerworks/webservices/library/ws-restful/

[13] Solr, Apache Project. URL: http://lucene.apache.org/solr/

[14] Splunk. URL: http://www.splunk.com/