МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ

о выполнении научно-исследовательской работы

магистранта направления 02.04.02 «Фундаментальная информатика и информационные технологии» (магистерская программа «Технологии разработки высоконагруженных систем»)

Автор работы:
студент группы КЭ-120
Поляков Андрей Владимирович

Научный руководитель:
кандидат физ.-мат. наук, доцент
Цымблер Михаил Леонидович

Оценка: _____

Дата: _____

Подпись: _____

Челябинск 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Временным рядом называется последовательность значений, описывающих протекающий во времени процесс, измеренных в последовательные моменты времени, обычно через равные промежутки. Временные ряды используются в самых различных областях (техника, экономика, медицина, банковское дело и др. [1]), и описывают различные процессы, протекающие во времени (ежедневные изменение цены на акции, курсы валют, изменения объемов продаж, годового объема производства и др. [17]). На данный момент хорошо изучена задача поиска тенденций, характерных для данного временного ряда. Относительно новой и мало исследованной является задача обнаружения диссонансов временного ряда.

Для анализа временного ряда используют модель, которая отражает предполагаемые особенности (компоненты) ряда. Обычно модель состоит из трех компонент:

1. Тренд — отражает общее поведение ряда в плане возрастания или убывания значений.

2. Сезонность — периодичные колебания значений, связанные, например, с днем недели или месяца.

3. Случайное значение — то, что останется от ряда после исключения других компонент.

Аномалия – это отклонение от обычного поведения системы [1]. Если анализировать только случайную компоненту, то многие аномалии можно свести к одному из следующих случаев:

1. Нахождение выбросов в данных (см. рис. 1а) — классическая задача, для решения которой уже имеется хороший набор решений (Правило трех сигм, межквартильный размах и др.).

2. Сдвиг (см. рисунок 1б). Задача обнаружения сдвига в данных неплохо исследована, поскольку встречается в обработке сигналов. Для её решения можно воспользоваться Twitter Breakout Detection.

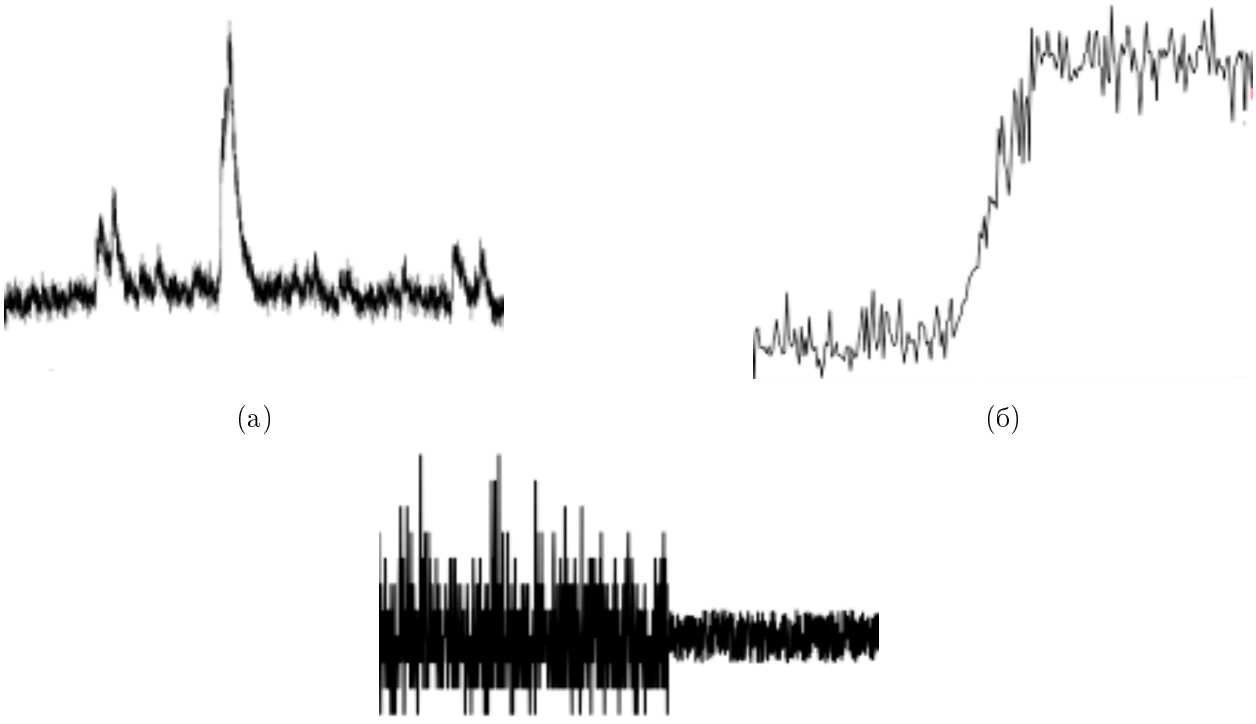
3. Изменение характера (распределения) значений (см. рисунок 1в).

4. Отклонение от «повседневного» (для данных с сезонностью).

Один из подходов поиска аномалий временного ряда, предложенный E. Keохом (E. Keogh) в 2005 году является поиск диссонансов (discords) [2]. *Диссонансом временного ряда* называется подпоследовательность временного ряда, максимально сильно отличающаяся от остальных подпоследовательностей. Диссонансы временного ряда – самые необычные подпоследовательности ряда. Алгоритм поиска диссонансов Keоха основан

на двух эвристиках, для построения которых используется алгоритм аппроксимации SAX, также предложенный Keoxom [3].

Как отмечается в работе E. Keoxa и др. [2], диссонансы временного ряда для интеллектуального анализа данных особенно привлекательны как детекторы аномалий, т.к. требуют только один интуитивный параметр (длина подпоследовательности), в отличие от большинства других алгоритмов поиска аномалий, требующих много параметров (от 3 до 7 неинтуитивных параметров).



(а) (б)

(в)

Рисунок 1 — Типы аномалий временного ряда: (а) Выброс; (б) Сдвиг; (в) Изменение характера (распределения) значений.

Процесс выявления аномалий в важнейших системах настоятельно рекомендуется автоматизировать, чтобы в процессе мониторинга больших систем была возможность автоматически выполнять действия, корректирующие состояние системы при выявлении аномалий, предотвращая тем самым, например, аварийные ситуации в системе. Также поскольку реальные данные могут быть огромного объема, необходимо задействовать все ресурсы вычислительной системы, на которой они обрабатываются (параллельное выполнение обработки, использование общей и дисковой памяти и др.). Актуальными являются исследования, посвященные использованию параллельных вычислений для решения данной задачи на кластерных системах, многоядерных процессорах и GPU. Для параллельной реализации алгоритма поиска диссонансов ряда перспективным является использование сопроцессоров Intel Xeon Phi.

Нахождение диссонансов временного ряда находят свое применение в следующих областях:

1. Медицина (электрокардиограммы, и др.) – отслеживание изменений состояния пациента и поиск патологий.

2. Дата-центры и компьютеры – мониторинг состояния памяти, загрузки ресурсов ВС, отслеживание сбоев.

3. Отслеживание изменений температуры и климата, погодных аномалий.

4. Оптимизация алгоритмов кластеризации, для «сложных случаев» (когда точки кластеризации в k-мерном пространстве сложно отнести к какому-то из кластеров).

Работа состоит из пяти разделовб заключения и списка литературы. В разделе 1.1 дается формальное описание задачи поиска диссонансов временного ряда. В разделе 1.2 приводятся основные особенности диссонансов, затрудняющие применение известных алгоритмов поиска аномалий. В разделе 1.3 дается обзор научных исследований в области последовательных алгоритмов поиска диссонансов временного ряда. В разделе 1.4 дается обзор современных подходов интеллектуального анализа данных для поиска диссонансов. Раздел 1.5 посвящен параллельным алгоритмам поиска диссонансов временного ряда.

1. Методы поиска диссонансов временного ряда

В данном разделе рассмотрены современные методы поиска диссонансов временного ряда. Материал главы организован следующим образом. В разделе 1.1 дается формальное описание задачи поиска диссонансов временного ряда. В разделе 1.2 приводятся основные особенности диссонансов, затрудняющие применение известных алгоритмов поиска аномалий. В разделе 1.3 дается обзор научных исследований в области последовательных алгоритмов поиска диссонансов временного ряда. В разделе 1.4 дается обзор современных подходов интеллектуального анализа данных для поиска диссонансов. раздел 1.5 посвящен параллельным алгоритмам поиска диссонансов временного ряда. Список литературы содержит 19 наим.

1.1. Термины и определения

Временной ряд (time series) T представляет собой хронологически упорядоченную последовательность вещественных значений $t_1, t_2, ..., t_N$, ассоциированных с отметками времени, где N – длина последовательности.

Подпоследовательность (subsequence) T_{im} временного ряда T представляет собой непрерывное подмножество T из m элементов, начиная с позиции i, т.е. $T_{im} = t_i, t_{i+1}, ..., t_{i+m-1}$, где $1 \leq i \leq N$ и $i + m \leq N$.

Расстояние (distance) между подпоследовательностями C и M представляет собой функцию, которая в качестве аргументов принимает C и M, и возвращает неотрицательное число R. Для подпоследовательностей функция расстояния является симметричной, т.е. $Dist(C, M) = Dist(M, C)$.

Пусть имеется временной ряд T, подпоследовательность C длины n, начинающаяся с позиции p, и подпоследовательность M длины n, начинающаяся с позиции q. Подпоследовательности C и M называются *несамопересекающимися (non-self match)*, если $\mid p - q \mid \geq n$.

Подпоследовательность D длины n, начинающаяся с позиции l называется *диссонансом* временного ряда T, если D находится на наибольшем расстоянии от ближайшей несамопересекающейся с D подпоследовательности, чем любые другие подпоследовательсти временного ряда, т.е. $\forall C \in T$, несамопересекающихся с D M_D и с C $M_C : min(Dist(D, M_D)) > min(Dist(C, M_C))$.

Подпоследовательность D длины n, начинающаяся с позиции p называется *K-м диссонансом временного ряда*, если D имеет K-е по размеру расстояние от ближайшей несамопересекающейся подпоследовательности, при этом не имея пересекающихся частей с i-ми диссонансами, начинающимися на позициях p_i, для всех $1 \leq i \leq K$, т.е. $\mid p - p_i \mid \geq n$.

Евклидово расстояние между двумя временными рядами Q и C длины n вычисляется по формуле 1:

$$Dist(Q, C) = \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \tag{1}$$

1.2. Особенности диссонансов временного ряда

Как отмечается в статье Е. Кеоха и др. [1], диссонансы временного ряда обладают некоторыми особенностями, которые затрудняют их поиск с помощью алгоритмов, показавших свою эффективность для поиска характерных особенностей ряда:

1. **Диссонансы не обязательно можно найти в разреженном пространстве.** Идея рассмотрения подпоследовательностей временных рядов как точек в пространстве уже давно используется десятками методов индексирования, поэтому можно представить, что такое представление было бы полезно для задачи. Мы могли бы просто проецировать наши временные ряды в n-мерное пространство и использовать существующие методы обнаружения выбросов. Проблема с этой идеей заключается в неинтуитивном факте, что диссонансы не обязательно живут в разреженных областях n-мерного пространства (наоборот, повторяющиеся шаблоны не обязательно живут в плотных частях n-мерного пространства). Другими словами, простые (и гладкие) фигуры оказываются в плотных окрестностях, потому что мы пересчитываем их смещенные версии. Эта проблема не позволяет нам использовать существующие алгоритмы, основанные на плотности, для поиска диссонансов временных рядов.

2. **Диссонансы не комбинируются.** Несколько общих парадигм для решения проблем основаны на способности разлагать проблему на более мелкие подзадачи, которые могут быть решены и допустимо рекомбинированы. В зависимости от точных определений такие методы по-разному называются динамическим программированием, «разделяй и властвуй», «снизу вверх» и др. [7]. Как показано в [1], такие идеи не помогут эффективно найти диссонансы временного ряда.

Перечисленные особенности показывают, что существующие алгоритмы и подходы мало полезны для поиска диссонансов. Это мотивировало исследователей на создание оригинальных алгоритмов, предназначенных именно для поиска диссонансов, обзор которых приводится в следующих подразделах.

1.3. Последовательные алгоритмы поиска диссонансов временного ряда

Алгоритм полного перебора

Наиболее простым с точки зрения реализации является алгоритм полного перебора (в статье Кеоха данный алгоритм назван «BRUTE-FORCE алгоритм»). Реализация данного алгоритма представлена на рисунке 1.

```
1  [dist, loc] BruteForce(T,n)
       Data: best_so_far_dist = 0
       Data: best_so_far_loc = NaN
2      for p = 1 to |T| − n + 1 do
           Data: nearest_neighbor_dist = infinity
3          for q = 1 to |T| − n + 1 do
4              //non-self match?
5              if (|p − q| ≥ n then
6                  if Dist (t_p, ..., t_{p+n−1}, t_q, ..., t_{q+n−1}) < nearest_neighbor_dist then
7                      nearest_neighbor_dist = Dist (t_p, ..., t_{p+n−1}, t_q, ..., t_{q+n−1})
8                  end
9              end
10         end
11         if nearest_neighbor_dist > best_so_far_dist then
12             best_so_far_dist = nearest_neighbor_dist best_so_far_loc = p
13         end
14     end
       Result: [ best_so_far_dist, best_so_far_loc ]
15 end
```

Algorithm 1:Наивный алгоритм поиска диссонансов

Алгоритм заключается в последовательном переборе всех возможных подпоследовательностей временного ряда заданной длины n и нахождения для каждой из них расстояния до ближайшей к ней несамопересекающейся подпоследовательности временного ряда длины n. Та из подпоследовательностей, для которой это расстояние максимально и есть диссонанс временного ряда.

Данный алгоритм требует всего один входной параметр – необходимую длину подпоследовательности, прост в реализации и позволяет получить точный результат. Однако он имеет фатальный недостаток для задач обработки больших данных – временная сложность алгоритма $O(m^2)$, что является неприемлемым при работе с достаточно большими наборами данных.

Время работы алгоритма можно улучшить на основе следующих двух наблюдений [1]:

1. Во внутреннем цикле нет необходимости находить ближайшего соседу для текущей подпоследовательности. Как только найдена подпоследовательность, расстояние до которой от исследуемой подпоследовательности меньше, чем $best\_so\_far\_dist$, можно считать, что исследуемая подпоследовательность не является диссонансом, выполнение внутреннего цикла можно прервать и перейти к следующей подпоследовательности во внешнем цикле.

2. Скорость работы зависит от того, в каком порядке внешний цикл предлагает кандидатов на роль диссонанса для проверки во внутреннем цикле, и от того, в каком порядке внутренний цикл проходится по остальным подпоследовательностям, в попытках найти подпоследовательность, которая расположена к текущей ближе, чем $best\_so\_far\_dist$ и досрочно завершить цикл.

HOTSAX алгоритм

HOTSAX-алгоритм был впервые предложен E. Keogh и кратко описывается в [1]. HOTSAX использует две эвристики для аугментации входных данных для получения отсортированных наборов подпоследовательностей, по которым проходятся внешний и внутренний циклы из переборного алгоритма (рис.1). Эвристика для внешнего цикла применяется один раз, а эвристика для внутреннего цикла учитывает текущую обрабатываемую во внешнем цикле подпоследовательность и генерирует новый порядок обхождения остальных подпоследовательностей на каждой итерации внешнего цикла. Реализация этого алгоритма показана на рисунке 2.

Перед построением алгоритма нужно выбрать эвристики, которые будут в нем использоваться. Внешняя эвристика в HOTSAX применяется единожды и имеет вычислительную сложность $O(m)$. Внутренняя эвристика применяется m-n раз и имеет вычислительную сложность $O(1)$.

Для подбора необходимых эвристик, следует учесть следующиее наблюдения:

1. Во внешнем цикле нам не нужна идеальная сортировка для достижения максимального ускорения, достаточно чтобы среди нескольких первых исследуемых подпоследовательностей была хотя бы одна, у которой достаточно велико расстояние до ближайшего соседа. Это достаточно рано задаст большое значение для $best\_so\_far\_dist$, из-за условие в строке листинга на рисунке 2 будет true намного чаще, что позволит намного раньше завершать прохождение очередной итерации внутреннего цикла.

2. Во внутреннем цикле также не нужна идеальная сортировка для получения существенного ускорения. Достаточно потребовать, чтобы среди нескольких первых подпоследовательностей была хотя бы одна, имеющая расстояние до подпоследовательности, исследуюмой во внешнем цикле, меньшее, чем $best\_so\_far\_dist$

В статье [1] приводятся следующие виды эвристик, которые можно применить:

1. Сортировка случайным образом: вычислительная сложность алгоритма в этом случае будет в диапазоне от $O(m)$ до $O(m_2)$ и зависит от входных данных.

2. «Магия»: гипотетически наилучшая из возможных сортировок. Для внешнего цикла подпоследовательности отсортированы по убыванию минимального расстояния до ближайшего соседа, а во внутреннем - по возрастанию расстояний до ближайшего соседа. Временная сложность составляет $O(m - n - 1)$ для первого прохода по внутреннему циклу и $O(1)$ для всех последующих. Общая временная сложность составляет $O(m) + O(m) \equiv O(m), m \gg n$.

3. «Ложный»: наихудшая для производительности алгоритма сортировка. Для внешнего цикла подпоследовательности отсортированы в порядке возрастания минимальных расстояний до ближайшего соседа, для внутреннего - в порядке убывания. Временная сложность составляет $O(m_2)$, а также некоторое время дополнительно тратится на выполнение проверок на строке листинга на рисунке 2.

Получить эвристику «Магия» мы не можем, однако, с учетом приведенных выше наблюдений, мы можем найти аппроксимацию к «магии». Для этого можно применить один из алгоритмов аппроксимации временных рядов.

HOTSAX алгоритм использует для аппроксимации алгоритм *SAX (Symbolic Aggregate approXimation).*

SAX алгоритм применяется для преобразования входных временных рядов в строки. SAX преобразовывает временной ряд X длины n в строку длины w, где $w \ll n$, используя алфавит A размера $\alpha > 2$.

Алгоритм был предложен Лином (Lin) и др., и расширяет подход на основе алгоритма *PAA (Piecewise Aggregate Approximation)*, наследуя первоначальную простоту алгоритма и низкую вычислительную сложность, в то же время предоставляя удовлетворительную чувствительность и избирательность при обработке входных данных. Более того, использование символьного представления данных позволяет использовать существующее в информатике многообразие структур данных и алгоритмов манипуляции строками, таких как хэширование, регулярные выражения, сопоставление паттернов, суффиксные деревья и выводы грамматик.

Алгоритм обычно состоит из 2-х шагов:

1. Преобразование исходного временного ряда в PAA представление

2. Преобразование PAA представления в строку

PAA апроксимирует временной ряд X длины n ($X = \{x_1, ..., x_n\}$) в вектор $\overline{X} = \{\overline{x_1}, ..., \overline{x_M}\}$, где $M <= n$. Каждый ($\overline{x_i}$) вычисляется по формуле 2:

$$\overline{x_i} = \frac{M}{n} \sum_{j=\frac{n}{M}(i-1)+1}^{\frac{n}{M}i} x_j \tag{2}$$

Для того, чтобы уменьшить размерность с n до M нужно разделить исходный временной ряд на M блоков одинаковой длины, а затем вычислить среднее значение в каждом из блоков. Последовательность, составленная из этих средних значений, является PAA аппроксимацией (или преобразованием) исходного временного ряда.

```
1  [dist, loc] HeuristicSearch(T,n)
       Data: best_so_far_dist = 0
       Data: best_so_far_loc = NaN
2      for Each p in T ordered by heuristic Inner do
           Data: nearest_neighbor_dist = infinity
3          for Each q in T ordered by heuristic Outer do
4              //non-self match?
5              if (|p − q| ≥ n then
6                  if Dist (t_p, ..., t_{p+n−1}, t_q, ..., t_{q+n−1} ) < best_so_far_dist then
7                      Break
8                  end
9              end
10         end
11         if nearest_neighbor_dist > best_so_far_dist then
12             best_so_far_dist = nearest_neighbor_dist best_so_far_loc = p
13         end
14     end
       Result: [ best_so_far_dist, best_so_far_loc ]
15 end
```

Algorithm 2:Алгоритм поиска диссонансов на основе эвристик

После того, как получено PAA представление, можно продолжить преобразования, чтобы получить дискретное представление временного ряда. В статье [3], на основание более 50 наборов данных, выявлено, что нормализованные подпоследовательности имеют распределение близкое к нормальному. Поэтому после нормализации PAA представления, можно использовать *«точки разделения» (breakpoints)*, которые разделяют области одинокового размера под кривой нормального распределения.

Для нормализации исходного временного ряда используется *z-нормализация* (Normalization to Zero Mean and Unit of Energy). Z-нормализация [18] состоит в преобразовании входного вектора в выходной вектор, для которого среднее арифметическое приблизительно равно 0, а среднеквадратичное отклонение близко к 1 (см. формулу 3):

$$x'_i = \frac{x_i - \mu}{\sigma}, \tag{3}$$

где $i \in N$.

Точки разделения (breakpoints) – это отсортированный список чисел $B = \{\beta_1, ..., \beta_{\alpha-1}\}$, таких, что площадь под кривой нормального распределения $N(0,1)$ между β_i и β_{i+1} равна $\frac{1}{\alpha}$ (В качестве β_0 и β_α берут $-\infty$ и ∞ соответственно). Точки разделения для разных значений α могут быть получены из специальных таблиц соответствий (lookup tables). После того, как значения точек разделения определены, PAA представление преобразуется в строку. Затем, преобразованное представление помещается в массив слов и количество вхождений данного слова увеличивается на 1.

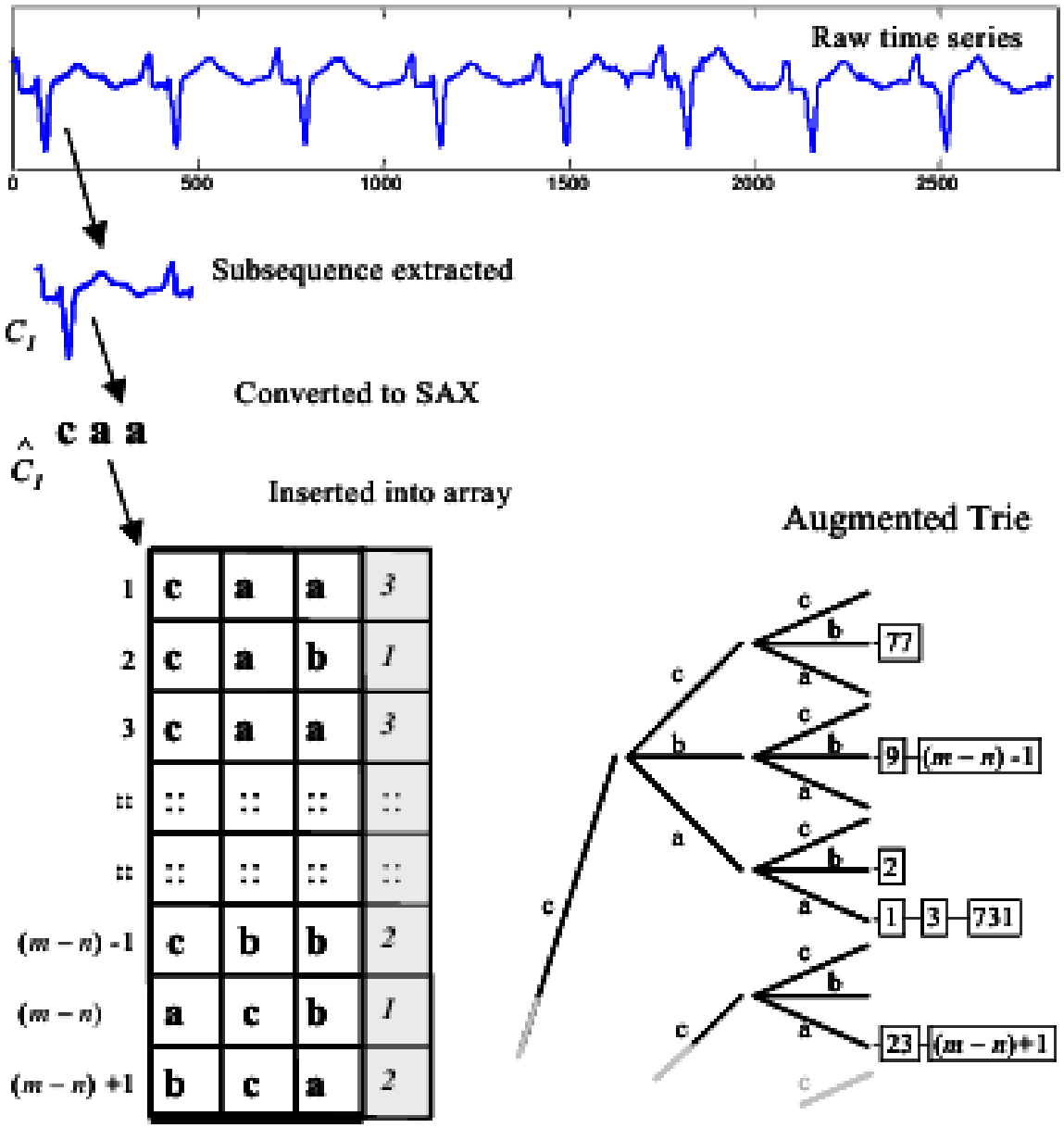


Рисунок 2 — Структуры данных, используемые в HOTSAX-алгоритме (массив слов и дерево).

После получения апроксимации в виде строк (слов) для каждой из подпоследовательностей временного ряда, массив слов сортируется по количеству вхождений каждого из слов. После сортировки массива строится агментированное дерево, листы которого содержат ссылки на строки массива слов, содержащие данное слово, а узлы - буквы слова. Процесс создания обеих структур данных (массива слов и дерева) имеет линейную временную сложность $O(n)$, что удовлетворяет требованиям, накладываемым на реализацию HOTSAX-алгоритма.

Таким образом, эвристика для внешнего цикла (Outer) заключается в следующем: необходимо найти в массиве слов элемент с наименьшим значением в колонке вхождений (так как диссонансы будут иметь апроксимацию в виде слов, которые имеют малое количество вхождений). Индексы всех элементов с наименьшим количеством вхождений запоминаются и проверяются во внешнем цикле алгоритма в первую очередь. Все остальные проверяются в случайном порядке.

Эвристика для внутреннего цикла (Inner) заключается в следующем: полученное из внешнего цикла слово находится в дереве и первыми во внутреннем цикле проверяются слова, содержащиеся в соответствующей листовой вершине. Все остальные проверяются в случайном порядке.

Итак, для HOTSAX алгоритма необходимо всего три параметра:

1. n – длина подпоследовательности.

2. w – длина слов в SAX представлении (для слабо меняющихся во времени последовательностей лучше подходит малое значение w, а для имеющих сложную структуру временных рядов – большое значение w).

3. α – мощность алфавита, используемого для формирования слов в SAX представлении (оптимальное значение $\alpha = 3$, как показано Кеохом в ??).

Модификации HOTSAX алгоритма

На основе HOTSAX алгоритма было создано много модификаций, позволявших обнаруживать диссонансы с большей скоростью или точностью. В [10] авторы используют локально-чувствительное хеширование для оценки сходства между подпоследовательностями, с помощью которого они могут эффективно переупорядочить поиск, чтобы обнаружить необычные подпоследовательности. Авторы [11] и [12] используют вейвлеты Хаара и аугментированные деревья для того, чтобы добиться эффективного уменьшения поискового пространства. Хотя эти подходы достигают ускорения на несколько порядков, по сравнению с наивным алгоритмом, их общим недостатком является то, что им всем в качестве входного параметра необходима длина потенциального диссонанса, и они обнаруживают диссонансы только определенной фиксированной длины. Кроме

того, они полагаются на вычисление расстояния, которое, как было выявлено Кеохом и др. [3], составляет более 99% времени выполнения этих алгоритмов.

Интересный подход к обнаружению аномалий в очень большой базе данных (набор данных с терабайтным размером) был показан Янковым (Yankov) и др. [13]. Авторы предложили алгоритм, который требует только двух проходов по базе данных. Однако для этого метода необходим диапазон, в котором может быть расположен диссонанс. Кроме того, как и алгоритму поиска диссонансов Кеоха и др., данному алгоритму требуется также и длина потенциального диссонанса временного ряда.

Некоторые решения используют методы аппроксимации, которые не требуют вычисления расстояния для необработанных временных рядов. Программа VizTree [14] – инструмент визуализации временного ряда, который позволяет одновременно обнаруживать как частые, так и редкие (аномальные) паттерны. VizTree использует trie (древовидную структуру данных, поиск в которой происходит за константное время), чтобы декодировать частоту появления для всех паттернов в их дискретизированной форме. Подобно тому, как это реализовано в VizTree, Чен (Chen) и др. [chen] также рассматривают аномалии как наиболее редкие временные ряды. Авторы используют вспомогательный счетчик для вычисления показателя аномальности каждого паттерна. Хотя определение аномалий у Чена и др. аналогично диссонансам, их техника требует больше входных параметров, таких как точность наклона e, количество паттернов аномалий k, минимальный порог и др. Кроме того, аномалии, обсуждаемые в их работе, содержат только две точки. Вей (Wei) и др. [16] предлагает другой метод, который использует растровые изображения временных рядов для измерения их сходства.

Наконец, в некоторых предыдущих работах было рассмотрено использование алгоритмической случайности для обнаружения аномалий временных рядов. Арнинг (Arning) и др. [17] предложили алгоритм линейной временной сложности для решения проблемы обнаружения аномалий в данных, хранящихся в базе даных. Имитируя естественный механизм запоминания ранее увиденных объектов данных с абстракциями на основе регулярных выражений, улавливающими наблюдаемую избыточность, их технология показала способность обнаруживать отклонения за линейное время. Предлагаемый метод зависит от определяемого пользователем размера сущности (размера записи базы данных). Альтернативно, Кеох и др. [18] показали безпараметрический (parameter-free) метод случайного поиска для приближенного обнаружения аномалий (алгоритм WCAD). Однако, их техника требует его многократного исполнения алгоритма, что делает его вычислительно дорогостоящим; кроме того, для работы алгоритму требуется задать размер скользящего окна (то есть размер диссонанса).

1.4. Методы Machine Learning для поиска аномалий временного ряда

Поиск диссонансов временного ряда с помощью методов машинного обучения имеет некоторые особенности, которые перечислены ниже. Сначала необходимо предобработать данные, привести их к виду, готовому для анализа [16].

Параметризация: приведение данных к заранее установленному формату, готовому для анализа (т.е. к входу в модель).

Обучение модели: т.е. ее построение и оптимизация на основе нормального или ненормального поведения. Это очень вариативная часть, которая сильно зависит от задачи и имеющихся данных.

Этап обнаружения: непосредственное тестирование и применение модели. Обычно, ее выходом является некоторая величина, определяющая степень отклонения поведения от нормального. Если это значение превышает некоторый порог, то срабатывает тревога.

Обучение «с учителем»

Задача классификации подразумевает определение категории новых экземпляров данных на основе набора изначально размеченных, учебных данных. При этом может иметь место мультиклассовость и вероятностный подход. В случае задачи обнаружения аномалий можно говорить о бинарной классификации (нормальное или ненормальное поведение данных). Итак, сами методы:

1. **Classification tree**: или дерево решений, строится по предикатам на признаках, в узлах дерева – предикаты, в листьях – метки классов. Существует несколько способов обучения, самый популярный из них — C4.5. Это не самый мощный алгоритм, однако, его точность выше, чем, например, у наивного Байеса [16, 9].

2. **Fuzzy Logic**: или нечеткая логика – форма модели, выведенная из теории нечетких множеств, строящаяся на неких приблизительных рассуждениях из предметной области, выведенных из обычной логики предикатов. Решающие правила строятся на некоторых статистиках данных [17].

3. **Na?ve bayes network**: В любых данных, обычно, имеют место статистические или естественные зависимости между признаками. Вероятностные зависимости между ними выразить довольно трудно, ведь мы знаем лишь, что на какие-то переменные в данных как-то влияют другие. Для того, чтобы восстановить такие зависимости используется вероятностная модель графа, называющаяся Сетью наивного Байеса. Качество такого подхода хуже, чем, например, у решающих деревьев, однако скорость выполнения и обучения выше, поэтому на больших выборках есть смысл использовать именно наивного Байеса [9].

4. **Genetic Algorithm**: эта модель принадлежит к классу эволюционных алгоритмов, которые находят широкое применение в различных сферах, ввиду таких свойств, как устойчивость к шуму и точность результатов. Практика показала хорошее качество данного метода и на задачах обнаружения аномалий [17].

5. **Neural Networks**: набор полносвязных слоев с различными функциями активации. Это одна из самых мощных моделей машинного обучения, способная восстанавливать самые сложные зависимости в данных. В задачах классификации, нейронные сети могут строить сложнее нелинейные разделяющие гиперповерхности.

6. **Support Vector Machine**: эта модель машинного обучения также способна восстанавливать сложные зависимости в данных и хорошо показывает себя в задаче обнаружения аномалий (во многих случаях не хуже нейронок) [18].

Кластеризация

Кластеризация – это разделение данных по группам похожих объектов. Каждая группа состоит из объектов, аналогичных друг другу, а в разных группах объекты, обычно, отличаются [13]. Методы кластеризации способны обнаруживать какие-либо зависимости (в том числе вторжения и аномалии) в данных без какой-либо априорной информации. Вот некоторые методы кластеризации, применяемые для обнаружения аномалий с их кратким описанием:

1. **K-means**: этот метод проводит кластеризацию на k непересекающихся классов, основанную на расстоянии между объектами, определяемыми признаками. Параметр k заранее задается исследователем [9]. В публикации [10] этот метод использовали для разделения временных интервалов с нормальным и аномальным трафиком.

2. **K-medoids**: алгоритм, схожий с предыдущим, но при поиске центра кластеров на каждой итерации ищет их не как среднее по группе, а как медоиду (такой элемент, различие которого с остальными минимально). Такой подход к задаче кластеризации более робастный (т.е. устойчивый к случайным воздействиям, шуму), нежели k-средних. Сравнения показывают, что на деле, в задаче обнаружения аномалий он работает лучше [11].

3. **EM Clustering**: этот алгоритм можно рассматривать как расширение метода k-средних. Тут, вместо метки кластера на основе его среднего значения, каждому элементу присваиваются степени вероятности его принадлежности к каждому кластеру. Подробнее метод описан здесь [12]. Он показывает еще большую точность, чем предыдущие два [11].

4. **Genetic Algorithm**: эта модель принадлежит к классу эволюционных алгоритмов, которые находят широкое применение в различных сферах, ввиду таких свойств, как устойчивость к шуму и точность результатов. Практика показала хорошее качество данного метода и на задачах обнаружения аномалий [17].

5. **Outlier Detection Algorithms**: обнаружение выбросов – это техника обнаружения шаблона в данных, который не соответствует ожидаемому, нормальному поведению. Выброс можно определить как точку в данных, которая сильно отличается от остальных по какой-либо метрике. Существует несколько различных подходов к решению задачи обнаружения выбросов на основе кластеризации и каждый из них лучше подходит в том или ином случае. Один из подходов использует расстояние между элементами в данных [11]. Он основан на методе k ближайших соседей и использует заранее заданную метрику. Чем дальше элемент от своих соседей, тем выше вероятность того, что это – выброс. Этот подход эффективен в обнаружении атак типа Denial of Service (DoS). Другой подход основан на плотности распределения данных. Метрические методы обнаружения выбросов зависят от общего распределения данных (заданного набора точек метрического пространства). Обычно, это распределение неравномерное и такой подход затруднителен. Основная же идея метода, основанного на плотности распределения состоит в том, чтобы сопоставить каждому элементу степень уверенности в том, что он является выбросом, это называется Local Outlier Factor (LOF). Локальный – т.к. ввиду разнородности плотности распределения данных, каждый элемент рассматривается лишь в окружении элементов, лежащих в некоторой его окрестности [14]. Применение этих методов к сетевым данным сопряжено с некоторыми сложностями, что описано в [8].

Гибридные методы

Иногда, вместо одного алгоритма лучше использовать комбинированный подход из разных моделей, чтобы компенсировать их недостатки друг другом.

1. **Каскад алгоритмов обучения с учителем**: часто, для получения хорошего качества классификации разумно использовать большое количество слабо обученных моделей, а результат рассматривать как голосование всех полученных алгоритмов (взвешенное или обычное). Чаще всего в качестве базовых алгоритмов используют разрешающие деревья, реже – SVM или что-либо еще.

2. **Комбинирование обучения с учителем и без**: существует и такое. Бывает, что предобработку данных проводят при помощи обучения без учителя (например, удаляют заранее неверные значения, дополняют пропуски или убирают слишком

похожие элементы), а затем выполняют обучение с учителем. Сочетание алгоритма k-средних и решающих деревьев дает хорошее качество на задаче обнаружения аномалий.

В целом, гибридные подходы показывают себя с хорошей стороны в плане точности (например SVM + NN [5]), но хуже в плане производительности. Разработчик сам решает, будет ли определенный прирост качества соответствовать дополнительно затраченным ресурсам.

1.5. Параллельные алгоритмы поиска диссонансов временного ряда

Параллельные алгоритмы поиска диссонансов временного ряда – наиболее перспективное направление исследования алгоритмов поиска диссонансов. На данный момент существует довольно мало работ, в которых исследуются способы распараллелить алгоритм поиска диссонансов, в их числе [8] и [13]. Для распараллеливания задачи обнаружения диссонансов ее необходимо разбить на независимые подзадачи, для возможности их решения на различных вычислительных узлах. В статье Кеоха [3] говорится, что метод «разделяй и властвуй» может давать некорректные результаты для задачи поиска диссонансов временного ряда. Чтобы избежать этого, необходимо найти возможности для разбиения задачи на подзадачи.

Алгоритм PPD

Алгоритм параллельного поиска диссонансов (Parallel Discord Discovery, PDD) впервые предложен Тианом и др. [8]. Алгоритм основан на предположении, что задачи поиска растояния до ближайшего соседа в алгоритме HOTSAX независимы друг от друга и могут выполняться параллельно. Затем, когда все расстояние найдены, можно найти максимальное из них.

Пусть дан временной ряд $T = \{t_1, ..., t_m\}$, где m – это его длина (все t_i - вещественные числа). Обозначим подпоследовательность ряда за $C_{p,n}$, где p – это индекс первого элемента подпоследовательности, а n – это ее длина. Для нахождения ближайшего соседа для подпоследовательности C_p, она и любые другие неперекрывающиеся подпоследовательности временного ряда передаются в один или более вычислительных узлов. Однако, расход времени на передачу двух подпоследовательностей и на вычисление расстояния между ними имеет один и тот же порядок величины. Это приводит к низкому *коэффициенту связи вычислений (CCR)* и, следовательно, к нерациональному использованию вычислительных ресурсов. Необходимо улучшить CCR. Суть улучшения состоит в том, чтобы устранить повторную передачу данных, которые уже были переданы до этого. Например, если мы задали длину скользящего окна 100 и передали подпоследовательность соответствующей длины, то следующая подпоследовательность

будет содержать 99 элементов, которые уже были переданы и всего 1 новый элемент. Если же передать 299 элементов, то мы передадим $299 - 100 + 1 = 200$ подпоследовательностей длины 100. Следовательно, CCR улучшится в $(200/299)/(1/100) = 67$ раз.

Алгоритм состоит из двух основных шагов: глобальной оценки расстояния до ближайшего соседа методом *распределенного разложения (DDE)* и линейного анализа всех для нахождения расстояния до ближайшего соседа такого, которое соответсвует диссонансу временного ряда. Второй шаг состоит из нескольких раундов. Непрерывные подпоследовательности передаются батчами (пакетами) среди вычислительных узлов, для лучшего CCR. В каждом раунде, вычислительные узлы работают независимо и в конце, обмениваются результатами. После каждого раунда, значение наибольшего найденного расстояния обновляется. Два основных шага, описанных ранее, являются самыми трудоемкими.

Перым шагом распределенной оценки диссонанса (DDE) является оценка расстояния до ближайшего соседа диссонанса. Эта оценка вместе с ранним отказом от дальнейших вычислений может значительно сократить количество вызовов функции вычисления расстояния. Чем ближе оценка к истине, тем меньше раз вычисляется функция расстояния. Для реализации такого подхода обычно используется некоторый массив индексов, однако, т.к. в нашем случае данные делятся на несколько сегментов, то создание централизованного массива с индексами для распределенных данных неэффективно. Алгоритм аппроксимирует каждую подпоследовательность символьным представлением, обозначаемым как , затем делит их на группы по сходству и вычисляет частоту каждой группы по количеству представлений , входящих в нее.

Следующим шагом алгоритм выбирает группу с наименьшим количеством членов, обозначаемую как A_d. Затем пробегается по каждому члену этой группы и находит для каждого локальные расстояния до ближайшего соседа. Минимальный среди них называют глобальным для этой подпоследовательности, максимальный среди них всех - общим глобальным. Число подпоследовательностей в рассматриваемой группе A_d влияет на качество работы алгоритма. Это число может быть довольно мало, тогда оценка будет плохой. Опытным путем Тиан и др. [8], выяснили, что всего таких групп схожести должно быть примерно 2-10.

Далее алгоритм рассчитывает все расстояния до ближайшего соседа и обновляет позицию и значение расстояния для диссонанса. Это самая трудоемкая часть всего алгоритма, поэтому непрерывные данные передаются, обычно, батчами (пакетами), также используется схема раннего отказа от дальнейших вычислений.

Все искомые расстояния до ближайшего соседа инициализируются как плюс бесконечность, затем проходим по всем вычислительным узлам, содержащим разные сегменты временного ряда. В течение этого процесса высчитывается локальное расстояние до ближайшего соседа для C_p. Если глобальное значение больше текущего расстояния до ближайшего соседа, то из C_b можно вычесть C_p и далее C_p не рассматривать. Эв-

ристический порядок доступа повышает эффективность метода раннего отказа. После вычисления расстояния между C_p и всеми другими подпоследовательностями в этом сегменте без запуска метода раннего отказа, локальное расстояние до ближайшего соседа становится глобальным для подпоследовательности C_p. Заметим, что количество промежуточных данных, которыми обмениваются вычислительные узлы, довольно мало. Это достигается благодаря тому, что значение глобального расстояния изначально инициализируется максимально возможным значением (еще в алгоритме DDE), и тогда довольно часто происходит раннее прерывание.

В ходе работы алгоритма может возникать дисбаланс нагрузки вычислительных узлов, некоторые из них могут закончить вычисление и простаивать. Состояние бездействия ухудшает использование вычислительного ресурса. Его можно уменьшить, выделив дополнительные массивы и используя общую входную очередь. В каждом раунде PDD создает общую очередь, которая содержит пакеты в несколько раз больше вычислительных узлов. Как только вычислительный узел заканчивает обработку массива, ему присваивается следующий массив из общей очереди. Круг завершается, когда обрабатываются все пакеты в общей очереди.

ЗАКЛЮЧЕНИЕ

В данной работе мы рассмотрели постановку задачи обнаружения диссонансов временного ряда. Был проведен анализ существующего опыта решений этой задачи. Рассмотрены классический алгоритм HOTSAX, предложенный E. Keохом и его модификации. Также рассмотрены алгоритмы поиска диссонансов методами интеллектуального анализа данных (data mining). Эти подходы можно разделить на три класса: задачи классификации, задачи кластеризации и гибридный подход. В первую входят, например, применение моделей k ближайших соседей, решающих деревьев, метода опорных векторов, нейронных сетей. В случае кластеризации, в основном, применяется метод k-means и его модификации. Метриками в методах кластеризации могут являться: евклидова (EUC), DWT. Наиболее приемлемый результат дают гибридные методы. Также был подробно рассмотрен один из способов распараллеливания алгоритма обнаружения диссонансов при помощи метода ближайшего соседа.

ЛИТЕРАТУРА

1. E. Keogh, J. Lin, S. H. Lee, and H. V. Herle, «Finding the most unusual time series subsequence: algorithms and applications», Knowledge and Information Systems, vol. 11, no. 1, pp. 1–27, 2006.

2. E. Keogh and J. Lin, «Hot SAX: Efficiently finding the most unusual time series subsequence», In Proc. of the 5th IEEE International Conf on Data Mining (ICDM 2005), Houston, Texas, Nov 27-30, 2005., pp. 226 – 233, 2005.

3. J. Lin, E. Keogh, S. Lonardi, B. Chiu «A symbolic representation of time series, with implications for streaming algorithms», In proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003.

4. H. Huang, K. Mehrotra, C. Mohan, «Detection of anomalous time series based on multiple distance measures», 28th International Conference on Computers and Their Applications (CATA-2013), Honolulu, Hawaii, USA, 2013.

5. J. Lin, E. Keogh, A. Fu, and H.V. Herle, «Approximations to magic: Finding unusual medical time series», IEEE Symposium on CBMS, pp. 329–334, 2005.

6. H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, «Querying and mining of time series data: experimental comparison of representations and distance measures», Proceedings of the VLDB Endowment, pp. 1542–1552, August 2008.

7. T.H. Coerman, C.E. Leiserson, R.L. Rivest, «Introduction to algorithms», McGraw-Hill Company, 1990.

8. H. Tian, Zh. Yongxin, M. Yishu, «Parallel Discord Discovery», ser. Lecture Notes in Artificial Intelligence, 20th Pacific-Asia Conference, PAKDD 2016 Auckland, New Zealand, April 19–22, 2016 Proceedings, Part II, vol. 9652, pp. 253-264, 2016.

9. P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, «Time series anomaly discovery with grammar-based compression», EDBT, 2015.

10. Wei, L., Keogh, E., Xi, X., «SAXually explicit images: Finding unusual shapes», In Proc. ICDM, 2006.

11. Fu, A., Leung, O., Keogh, E., Lin, J., Finding Time Series Discords based on Haar Transform, In Proc. of Intl. Conf. on Adv. Data Mining and Applications, 2006.

12. Bu, Y., Leung, O., Fu, A., Keogh, E., Pei, J., Meshkin, S., «WAT: Finding Top-K Discords in Time Series Database», In Proc. of SIAM Intl. Conf. on Data Mining, 2007.

13. Yankov, D., Keogh, E., Rebbapragada, U., «Disk aware discord discovery: finding unusual time series in terabyte sized data sets» , Knowledge and Information Systems, pp. 241-262, 2008.

14. Lin, J., Keogh, E., Lonardi, S., Lankford, J.P., Nystrom, D. M., «Visually mining and monitoring massive time series», In Proc. ACM SIGKDD Intn'l Conf. on KDD, 2004.

15. Chen, X., Zhan, Y., «Multi-scale Anomaly Detection Algorithm based on Infrequent Pattern of Time Series», J. of Computational and Applied Mathematics, 2008.

16. Wei, L., Kumar, N., Lolla, V., Keogh, E., Lonardi, S., Ratanamahatana, C., «Assumption-free Anomaly Detection in Time Series», In Proc. SSDBM, 2005.

17. Arning, A., Agrawal, R., Raghavan, P.,« A Linear Method for Deviation Detection in Large Databases», In KDD, pp. 164-169, 1996.

18. Keogh, E., Lonardi, S., Ratanamahatana, C.A., «Towards parameter-free data mining», In Proc. KDD, 2004.