Параллельный алгоритм восстановления пропущенных значений потокового временного ряда в режиме реального времени*

М.Л. Цымблер[1], А.Н. Полуянов[2]

[1]Южно-Уральский государственный университет (Челябинск),
[2]Институт математики им. С.Л. Соболева СО РАН (Омск)

В настоящее время во многих предметных областях обработка потоковых временных рядов в режиме реального времени связана с необходимостью синтеза значения ряда, которое было пропущено ввиду технического сбоя или человеческого фактора. В данной статье предлагается параллельный алгоритм восстановления пропущенных значений потокового временного ряда в режиме реального времени для многоядерного процессора. Алгоритм использует набор опорных временных рядов (reference time series), которые имеют семантическую связь с исходным рядом и упорядочены по убыванию корреляции с ним. Алгоритм применяет следующую эвристику: если в опорных рядах имеют место повторяющиеся (схожие) подпоследовательности, то в ряде, содержащем пропущенное значение, повторяющиеся подпоследовательности возникают в тех же временны́х интервалах. Образцами поиска для каждого опорного ряда полагаются подпоследовательности заданной длины, оканчивающиеся в момент пропуска значения в исходном ряде. Схожесть подпоследовательностей с образцом определяется на основе меры DTW (Dynamic Time Warping), имеющей квадратичную вычислительную сложность относительно длины подпоследовательности. Применяется техника нижних границ (lower bounding), позволяющая отбрасывать подпоследовательности, заведомо непохожие на образец поиска, без вычисления DTW. Нижние границы схожести имеют меньшую чем квадратичную сложность и вычисляются параллельно с помощью технологии программирования OpenMP. Восстановленное значение вычисляется как среднее арифметическое последних элементов найденных интервалов. В вычислительных экспериментах предложенный алгоритм демонстрирует приемлемое качество и высокое быстродействие восстановления, что позволяет применять алгоритм в режиме реального времени.

Ключевые слова: временной ряд, восстановление пропущенных значений, параллельный алгоритм, многоядерный процессор, DTW (Dynamic Time Warping), отбрасывание по нижним границам (lower bounding).

1. Введение

Обработка данных временных рядов в режиме реального времени требуется в широком спектре прикладных и научных задач: приложения Интернета вещей, персональная медицина, моделирование климата, финансовое прогнозирование и др. В задачах подобного рода зачастую неприемлемы пропущенные значения во временном ряде, возникающие вследствие технических сбоев или человеческого фактора: пропуски подлежат замене на правдоподобные значения, синтезированные на лету. В соответствии с этим является актуальной задача разработки алгоритмов восстановления пропущенных значений потокового временного ряда, эффективных в смысле точности и быстродействия восстановления.

В данной работе предлагается параллельный алгоритм восстановления пропущенных значений потокового временного ряда в режиме реального времени для многоядерного процессора. Алгоритм основан на применении меры схожести временных рядов DTW (Dynamic

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 20-07-00140) и Министерства науки и высшего образования РФ (государственные задания FENU-2020-0022, FWNF-2022-0016).

Time Warping, динамическая трансформация времени) [2]. На сегодня DTW признается научным сообществом одной из лучших мер схожести временных рядов для многих предметных областей [4]. Однако мера DTW имеет квадратичную вычислительную сложность относительно длины ряда и определяется рекуррентными формулами, что требует нетривиального подхода к распараллеливанию ее вычисления.

Статья организована следующим образом. Раздел 2 содержит обзор работ, наиболее близких к выполненному исследованию. В разделе 3 приводится формальная постановка задачи. В разделе 4 дано описание предлагаемого параллельного алгоритма. В разделе 5 представлены результаты вычислительных экспериментов по исследованию эффективности предложенного алгоритма. Заключение содержит резюме работы и обозначает направление будущих исследований.

2. Обзор работ

Создание эффективных методов и алгоритмов восстановления пропущенных данных является одной из наиболее актуальных проблем обработки временных рядов [8]. В данном разделе кратко рассмотрены только те работы, которые наиболее близки проведенному исследованию и затрагивают следующие аспекты: применение для восстановления ряда принципа ближайших соседей, меры схожести DTW, опорных рядов (коррелирующих с исходным рядом), а также использование параллельных вычислений.

В работе [1] Батиста (Batista) и др. предложили метод восстановления значений временного ряда kNNI (k-Nearest Neighbor Imputation) на основе ближайших соседей. Пусть имеется объект с множеством атрибутов, один из которых (обозначим его как A) имеет пропущенные значения. Метод предписывает сначала найти k «соседей» — объектов, имеющих схожие значения в атрибутах, отличных от A (без конкретизации меры схожести). Затем отсутствующее значение синтезируется на основе значений в атрибуте A у соседей. В работе [16] Троянская (Troyanskaya) и др. предложили использование взвешенных ближайших соседей, когда вес соседа пропорционален схожести с образцом поиска в смысле евклидовой метрики.

В работе [6] Хсю (Hsu) и др. предложили восстановление отсутствующих значений, основанный на использовании принципа k ближайших соседей и меры DTW. Алгоритм используется для восстановления отсутствующих значений во временном ряду, который представляет собой значения уровня экспрессии генов, полученные с помощью ДНК-микрочипов в серии экспериментов.

В работе [13] Фан (Phan) и др. применили DTW для восстановления пропущенных значений в многомерном временном ряде. Восстановление выполняется отдельно для каждого ряда-координаты следующим образом. Подпоследовательность ряда, которая начинается непосредственно после промежутка пропущенных значений имеет ту же длину, что и указанный промежуток, объявляется образцом поиска. Далее в части ряда, следующей после образца, выполняется поиск подпоследовательности, которая по длине равна образцу и является самой похожей на образец в смысле меры DTW. Подпоследовательность, располагающаяся непосредственно перед найденной и имеющая ту же длину, используется для поэлементного заполнения пропущенного промежутка.

Те же авторы в работе [14] представили алгоритм DTWBI (DTW-Based Imputation) для восстановления пропущенных значений в одномерном временном ряде. Восстанавливаемый временной ряд подвергается преобразованию DDTW (Derivative DTW) [7]. Далее подпоследовательность ряда, находящаяся непосредственно перед промежутком из пропущенных значений, и имеющая ту же длину, что и указанный промежуток, объявляется образцом поиска. Далее в части ряда, предшествующей образцу, выполняется поиск подпоследовательности, самой похожей на образец в смысле меры DTW. Подпоследовательность, расположенная после найденной и имеющая ту же длину, используется для поэлементного

заполнения пропущенного промежутка.

В работе [17] Велленцон (Wellenzohn) и др. предложили алгоритм TKCM (Top-k Case Matching), использующий для восстановления пропущенных значений ряда набор опорных временных рядов (reference time series), которые имеют семантическую связь с исходным рядом. Примерами исходного и опорных временных рядов могут служить показания температурных датчиков, установленных в географически близких локациях [17]. Алгоритм применяет эвристику, согласно которой похожие ситуации в опорных рядах возникают в те же временные промежутки, что и в ряде, подлежащем восстановлению. В каждом из опорных рядов TKCM объявляет образцом поиска подпоследовательность, завершающуюся в момент возникновения пропуска в исходном ряде. Далее в каждом из опорных рядов алгоритм выполняет поиск k подпоследовательностей, которые являются ближайшими соседями выбранных образцов и не могут перекрывать друг друга. При поиске используется схема динамического программирования на основе евклидовой метрики, которая минимизирует целевую функцию суммарного отличия подпоследовательностей в опорных рядах от соответствующих образцов. Восстанавливаемое значение получается как среднее арифметическое точек исходного ряда, которые соответствуют завершающим точкам временных интервалов найденных ближайших соседей. Алгоритм обеспечивает хорошую точность восстановления, когда отсутствуют блоки значений. Однако это достигается за счет линейной временной сложности по всем ее параметрам (длина образца, количество опорных временных рядов, число ближайших соседей).

В обзоре и экспериментальном сравнении двенадцати современных алгоритмов восстановления пропущенных значений временных рядов [8] Хаяти (Khayati) и др. отмечают, что в настоящее время, по-видимому, отсутствуют успешные попытки использовать аппаратное обеспечение для ускорения алгоритмов восстановления отсутствующих значений в больших временных рядах.

В данном исследовании предлагается параллельный алгоритм IDK восстановления пропущенных значений временного ряда для многоядерного процессора. Подобно алгоритму TKCM, IDK использует поиск ближайших соседей в опорных временных рядах. Однако при поиске вместо схемы динамического программирования и евклидовой метрики IDK использует меру схожести DTW, которая имеет квадратичную вычислительную сложность, но в общем случае лучше отражает схожесть формы образца поиска и подпоследовательности ряда [4]. При этом применяется каскад нижних границ схожести [15]: подпоследовательности, заведомо непохожие на запрос, отбрасываются без вычисления DTW, что существенно сокращает объем вычислений [4]. В реализации используется идея предварительного параллельного вычисления нижних границ, предложенная нами ранее в работе [11]. Указанные разработки обуславливают возможность применения алгоритма IDK в режиме реального времени.

3. Формальные определения и обозначения

Потоковый временной ряд (streaming time series) представляет собой набор вещественных значений, поступающих последовательно одно за другим в режиме реального времени:

$$T = (\ldots, t_{n-2}, t_{n-1}, t_n), t_i \in \mathbb{R}. \tag{1}$$

Мы полагаем, что n-элементное окно ряда T может быть размещено в оперативной памяти и n имеет порядок десятков-сотен тысяч элементов.

Значение элемента ряда t_n в текущий момент времени пропущено, что обозначается как t_n=`NULL`. Решаемая нами задача состоит в том, чтобы вместо пропущенного значения в режиме реального времени вычислить синтетическое значение $\tilde{t}_n$, которое как можно меньше отличалось бы от t_n.

Предлагаемый в данной работе алгоритм синтеза значения $\tilde{t}_n$ основан на поиске в ряде T подпоследовательности C, которая является самой похожей на заданный образец поиска Q в смысле меры схожести DTW [2]. Ниже приводятся соответствующие формальные определения с использованием нотации из работы [15].

Подпоследовательностью (subsequence) временного ряда T, имеющей длину m, называют непрерывное подмножество T из m элементов, начиная с позиции i:

$$T_{i,\,m} = (t_i, \ldots, t_{i+m-1}), 1 \leqslant m \ll n, 1 \leqslant i \leqslant m - n + 1. \tag{2}$$

Множество всех подпоследовательностей ряда T, имеющих длину m, обозначается как S_T^m.

Пусть имеются две подпоследовательности ряда T: $C = (c_1, \ldots, c_m)$ и $Q = (q_1, \ldots, q_m)$. Тогда DTW-расстояние между C и Q определяется следующим образом [2]:

$$\begin{gathered} \mathrm{DTW}(C, Q) = D(m, m), \\ D(i, j) = (c_i - q_j)^2 + \min \begin{cases} D(i-1, j) \\ D(i, j-1) \\ D(i-1, j-1) \end{cases}, \\ D(0, 0) = 0,\ D(i, 0) = D(0, j) = +\infty,\ 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant m. \end{gathered} \tag{3}$$

В формуле (3) $D \in \mathbb{R}^{m \times m}$ является *матрицей трансформации*, которая задает выравнивание между подпоследовательностями C и Q друг относительно друга. В данной матрице строится *путь трансформации*, представляющий собой набор смежных элементов матрицы, который начинается и заканчивается в диагонально противоположных крайних элементах и устанавливает соответствие между C и Q, минимизируя общее расстояние между ними.

Подпоследовательность $C \in S_T^m$ называется *самой похожей (best match subsequence)* на заданный образец поиска Q, если

$$\mathrm{DTW}(C, Q) \leqslant \mathrm{DTW}(T_{i,\,m}, Q), 1 \leqslant i \leqslant m - n + 1. \tag{4}$$

Прямолинейный поиск самой похожей подпоследовательности не подходит для режима реального времени, поскольку DTW имеет вычислительную сложность $O(m^2)$ [4]. В силу этого нами используется *техника нижних границ (lower bounding)* [4], которая позволяет отбрасывать подпоследовательности, заведомо непохожие на запрос, без вычисления DTW, и существенно сократить объем вычислений.

Нижняя граница (LB, lower bound) представляет собой строго доказанный порог DTW-расстояния между подпоследовательностью и образцом поиска, сложность вычисления которого меньше $O(m^2)$. Техника нижних границ предполагает движение скользящего окна длины m в ряде T с шагом 1. DTW-расстояние от текущей подпоследовательности до запроса обозначают как *bsf (best-so-far)*. Если нижняя граница для рассматриваемой подпоследовательности превышает порог bsf, то значение DTW для нее и запроса также превысит bsf. Поэтому данная подпоследовательность заведомо непохожа на запрос и может быть отброшена без вычисления DTW. Перед началом сканирования bsf инициализируется значением $+\infty$, далее на каждом последующем шаге выполняется попытка уменьшить bsf следующим образом:

$$bsf_{(0)} = +\infty;\ \ bsf_{(i)} = \min\Big(bsf_{(i-1)}, \begin{cases} +\infty & , \mathrm{LB}(Q, C) > bsf_{(i-1)} \\ \mathrm{DTW}(Q, C) & , otherwise \end{cases}\Big). \tag{5}$$

Для применения техники нижних границ требуется, чтобы подпоследовательность и образец поиска были подвергнуты z-нормализации [15]. Z-нормализацией ряда $T = (t_1, \ldots, t_m)$ является ряд $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_m)$, элементы которого вычисляются следующим образом:

$$\hat{t}_i = \frac{t_i - \mu}{\sigma},\ \mu = \frac{1}{m}\sum_{i=1}^{m} t_i,\ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} t_i^2 - \mu^2. \tag{6}$$

Для повышения эффективности отбрасывания заведомо непохожих подпоследовательностей нижние границы применяют каскадом [15] (одна за другой в порядке возрастания их вычислительной сложности). В нашем исследовании применяются три наиболее известные нижние границы $\mathrm{LB_{Kim}FL}$ [9], $\mathrm{LB_{Keogh}EQ}$ [5] и $\mathrm{LB_{Keogh}EC}$ [15].

Нижняя граница схожести $\mathrm{LB_{Kim}FL}$ представляет собой квадрат Евклидова расстояния между первой и последней парами точек запроса и подпоследовательности:

$$\mathrm{LB_{Kim}FL}(Q, C) = (\hat{q}_1 - \hat{c}_1)^2 + (\hat{q}_m - \hat{c}_m)^2. \tag{7}$$

Нижняя граница $\mathrm{LB_{Keogh}EC}$ показывает схожесть между *оболочкой (envelope)* E запроса и подпоследовательностью и вычисляется следующим образом:

$$\mathrm{LB_{Keogh}EC}(Q, C) = \sum_{i=1}^{m} \begin{cases} (\hat{c}_i - u_i)^2 & , \ \hat{c}_i > u_i \\ (\hat{c}_i - \ell_i)^2 & , \ \hat{c}_i < \ell_i \\ 0 & , \ otherwise \end{cases}. \tag{8}$$

В формуле (8) последовательности $U = (u_1, \ldots, u_m)$ и $L = (\ell_1, \ldots, \ell_m)$ обозначают *верхнюю* и *нижнюю границы оболочки* запроса Q соответственно, которые вычисляются следующим образом:

$$\begin{aligned} u_i &= \max(\hat{q}_{i-r}, \ldots, \hat{q}_{i+r}), \\ \ell_i &= \min(\hat{q}_{i-r}, \ldots, \hat{q}_{i+r}), \end{aligned} \tag{9}$$

где параметр r $(1 \leqslant r \leqslant m)$ представляет собой *ограничение полосы Сако—Чиба (Sako–Chiba band)* [4], которое показывает, что путь трансформации не может отклоняться более чем на r ячеек от диагонали матрицы трансформации.

Нижняя граница $\mathrm{LB_{Keogh}EQ}$ представляет собой Евклидово расстояние между запросом Q и оболочкой подпоследовательности C, то есть по сравнению с $\mathrm{LB_{Keogh}EC}$ роли запроса и подпоследовательности меняются местами:

$$\mathrm{LB_{Keogh}EQ}(Q, C) = \mathrm{LB_{Keogh}EC}(C, Q). \tag{10}$$

Каскад может быть дополнен другими нижними границами, например, $\mathrm{LB_{Yi}}$, $\mathrm{LB_{PAA}}$ [4]. Техника нижних границ позволяет отбросить до 99% вычислений DTW [15].

4. Параллельный алгоритм восстановления пропусков

В данном разделе представлен новый параллельный алгоритм для многоядерного процессора, получивший название *IDK (Imputation based on DTW and K Nearest Neighbors)*, который выполняет восстановление пропущенного значения потокового временного ряда в режиме реального времени. Ниже в разделе 4.1 описана общая схема восстановления, используемая алгоритмом. В разделе 4.2 представлены структуры данных алгоритма. Раздел 4.3 содержит описание принципов реализации.

4.1. Общая схема восстановления

Алгоритм применяет опорные временные ряды и следующую эвристику, примененные ранее в алгоритме TKCM [17]: если в опорных рядах имеют место повторяющиеся (похожие) подпоследовательности, то в ряде, содержащем пропущенное значение, повторяющиеся подпоследовательности возникают в тех же временны́х интервалах.

Пусть $R^1, \ldots, R^d$ $(d > 1)$ — опорные потоковые временные ряды для ряда T. Это означает, что в каждом опорном ряде отсутствуют `NULL`-значения, и его элементы получены в те же моменты времени, что и элементы ряда T с соответствующими порядковыми номерами. Кроме того, мы полагаем, что в данном списке ряды упорядочены по убыванию корреляции с T, например, по убыванию абсолютного значения коэффициента корреляции Пирсона.

Как и в случае исходного ряда, мы предполагаем, что совокупность n-элементных окон опорных рядов размещена в оперативной памяти. Алгоритм выполняет следующие шаги: поиск по образцу, скоринг и реконструкция.

На шаге *поиска по образцу* сначала для каждого опорного ряда R^i определяется образец, состоящий из m последних по времени элементов ряда (где $1 \leqslant m \ll n$ — параметр алгоритма): $Q^i = R^i_{n-m+1,\,m}$. Далее для каждого опорного ряда R^i в его окне с первого по $(n-2m)$-й элемент выполняется поиск подпоследовательности, которая имеет длину m и является самой похожей на образец Q^i в смысле меры DTW. При этом вычисляется DTW-расстояние от запроса до каждой подпоследовательности опорного ряда, за исключением подпоследовательностей, заведомо не похожих на запрос, для которых сохраняется значение $+\infty$.

На шаге *скоринга*, используя вычисленные на предыдущем шаге DTW-расстояния от запросов до подпоследовательностей опорных рядов, определяется множество $kNNset$ подпоследовательностей исходного ряда, применяемых для реконструкции. Данное множество состоит из k не пересекающихся друг с другом подпоследовательностей длины m (где $1 \leqslant k < \lceil n/m \rceil$ — параметр алгоритма), которые являются наиболее значимыми для восстановления пропущенного значения. Значимость подпоследовательности определяется нами как функция от DTW-расстояний соответствующих подпоследовательностей в опорных рядах, подсчитанных на шаге поиска по образцу.

На шаге *реконструкции* восстановленное значение вычисляется как среднее арифметическое последних элементов подпоследовательностей, найденных на шаге скоринга.

4.2. Структуры данных

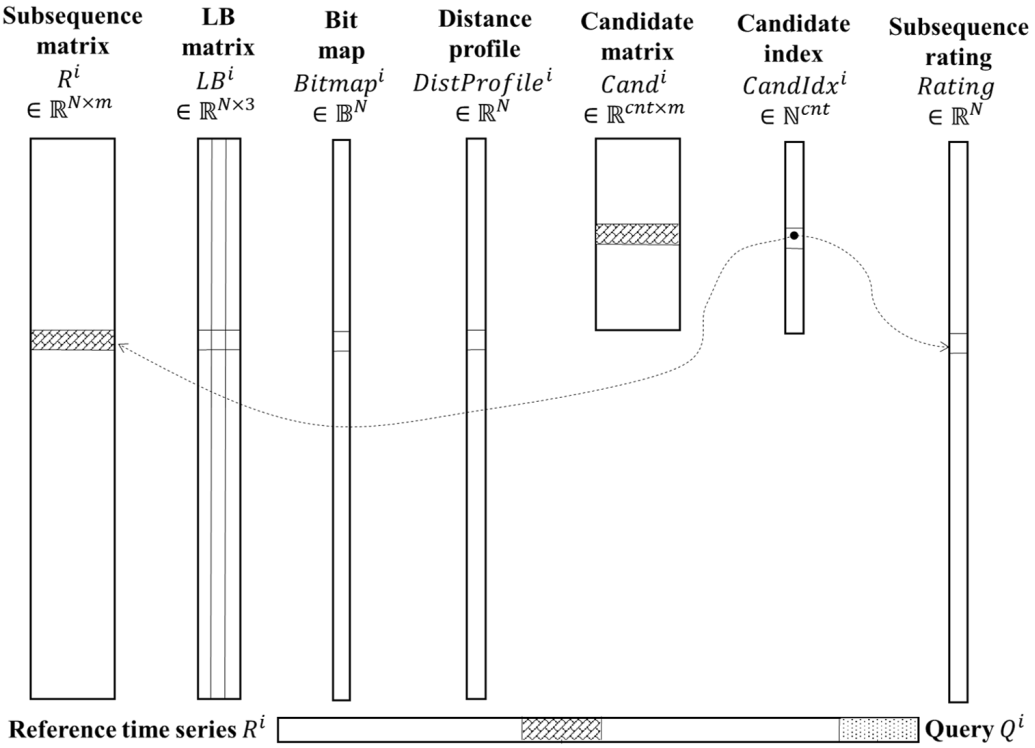

Рис. 1. Структуры данных алгоритма IDK

Структуры данных алгоритма IDK представлены на рис. 1. Каждый опорный временной ряд хранится в виде матрицы подпоследовательностей $R^i \in \mathbb{R}^{N\times m}$, где число $N = n-m$ представляет собой количество подпоследовательностей, имеющих длину m, исключая запрос. Для каждого опорного ряда определяются следующие структуры данных: матрица

нижних границ, битовая карта, профиль расстояния, матрица и индекс кандидатов. Помимо указанных структур данных, для выполнения реконструкции хранится рейтинг интервалов исходного временного ряда.

Матрица нижних границ $LB^i \in \mathbb{R}^{N\times 3}$ хранит значения нижних границ, вычисляемых по формулам (7)–(10). Отметим, что в классическом случае при поиске наиболее похожей подпоследовательности ближайших соседей следующая нижняя граница в каскаде вычисляется только в том случае, если текущая подпоследовательность не отбрасывается. Очевидно, что такие вычисления не могут быть распараллелены. В предлагаемом нами подходе вычисляются все нижние границы, поскольку, несмотря, на избыточность таких вычислений, они выполняются однократно и параллельно ввиду отсутствия зависимостей по данным между нижними границами.

Битовая карта представляет собой вектор $BitMap^i \in \mathbb{B}^N$, в котором для каждой подпоследовательности опорного временного ряда хранится конъюнкция результатов сравнения bsf и каждой из нижних границ (7)–(10) для запроса Q^i:

$$BitMap^i(j) = \wedge_{s=1}^{3}(LB^i(j,s) < bsf). \tag{11}$$

Таким образом, если какая-либо нижняя граница превысит значение bsf, то битовая карта содержит `FALSE` и показывает, что соответствующая подпоследовательность заведомо непохожа на запрос.

Профиль расстояния представляет собой вектор $DistProfile^i \in \mathbb{R}^N$, в котором хранятся DTW-расстояния между подпоследовательностями опорного ряда R^i и запросом Q^i. Для подпоследовательностей, заведомо не похожих на запрос, в профиле расстояния хранится значение $+\infty$.

Матрица кандидатов $Cand^i \in \mathbb{R}^{cnt^i\times m}$ хранит подпоследовательности, которые не были отброшены как заведомо непохожие на образец Q^i, где количество таких подпоследовательностей cnt^i определяется после применения каскада нижних границ. Данная матрица используется для параллельного вычисления DTW-расстояний между каждой из ее строк и запросом.

Индекс кандидатов $CandIndex^i \in \mathbb{N}^{cnt^i}$ хранит номера соответствующих строк $Cand^i$ в опорном ряде R^i.

Рейтинг подпоследовательностей $Rating \in \mathbb{R}^N$ представляет собой вектор, который хранит значимость подпоследовательностей длины m в потоковом n-элементном окне для последующего поиска k наиболее значимых из них и выполнения реконструкции отсутствующего значения t_n.

4.3. Реализация

В данном разделе описаны принципы параллельной реализации шагов поиска и скоринга (шаг реконструкции является алгоритмически тривиальным). Параллелизм для многоядерного процессора реализован на основе технологии программирования OpenMP [12]. В модели OpenMP приложение рассматривается как процесс (набор инструкций, исполняемых последовательно), который может запустить (*fork*) заданное количество параллельно исполняемых нитей. Нити разделяют память процесса и в то же время для хранения собственных данных имеют приватную память. Среда исполнения приложения автоматически назначает нитям различные ядра процессора. Формирование параллельных нитей осуществляется вставкой директивы компилятора `#pragma` в исходный код. Одним из наиболее типичных случаев применения данной директивы является распараллеливание цикла **for** с помощью `#pragma omp parallel for`, когда каждая из нитей обрабатывает собственный диапазон значений счетчика цикла.

Алг. 1 SimilaritySearch(IN: R, Q; OUT: $DistProfile$)

```
 1: #pragma omp parallel for
 2: for i ∈ 1..N do
 3:     R(i,·) ←zNormalize(R(i,·))
 4:     LB(i,1) ← LB_Kim FL(Q̂, R(i,·))
 5:     LB(i,2) ← LB_Keogh EC(Q̂, R(i,·))
 6:     LB(i,3) ← LB_Keogh EQ(Q̂, R(i,·))
 7: BitMap ← TRUE; DistProfile ← +∞
 8: bsf ← DTW(Q̂, R(N,·))
 9: while TRUE do
10:     #pragma omp parallel for
11:     for i ∈ 1..N do
12:         BitMap(i) ← BitMap(i) and LB(i,1) < bsf and
            LB(i,2) < bsf and LB(i,3) < bsf
13:     cnt ← 0
14:     for i ∈ 1..N do
15:         if BitMap(i) = TRUE then
16:             cnt ← cnt + 1; Cand(cnt) ← R(i,·); CandIndex(cnt) ← i
17:     if cnt = 0 then
18:         break
19:     s ← 1
20:     while TRUE do
21:         left ← p·(s−1)+1; right ← min(cnt, p·s)
22:         #pragma omp parallel for reduction(min: dist)
23:         for i ∈ left..right do
24:             dist ← DTW(Q̂, Cand(i,·))
25:             DistProfile(CandIndex(i)) ← dist
26:             BitMap(CandIndex(i)) ← FALSE
27:         bsf ← min(bsf, dist); s ← s + 1
28:         if dist < bsf or right = cnt then
29:             break
```

Псевдокод шага поиска представлен в Алг. 1. Поиск реализуется с помощью двух вложенных циклов: внешний цикл по опорным временным рядам и внутренний цикл по подпоследовательностям опорного временного ряда. Поскольку, как правило, количество подпоследовательностей в опорном ряде существенно больше количества опорных рядов, а последнее, в свою очередь, существенно меньше, чем количество нитей приложения, мы выполняем распараллеливание внутреннего цикла, чтобы обеспечить значимую нагрузку нитей. Для упрощения записи алгоритма мы опускаем внешний цикл по опорным временным рядам и не пишем в структурах данных соответствующий верхний индекс. За p обозначено количество нитей, на которых исполняется параллельное приложение.

Алгоритм выполняется следующим образом. Сначала параллельно нормализуются подпоследовательности и вычисляются нижние границы (строки 1–6), инициализируются структуры данных и переменная bsf (строки 7–8), которой присваивается величина DTW-расстояния между запросом и ближайшей непересекающейся с ним подпоследовательностью.

Затем алгоритм итеративно уменьшает значение bsf и отбрасывает без вычисления DTW-расстояния подпоследовательности, заведомо не похожие на запрос (строки 9–29). При этом сначала параллельно вычисляется битовая карта (строки 10–12), а затем формируются матрица кандидатов и индекс кандидатов (строки 13–16). Если подпоследовательности-кандидаты не найдены, то алгоритм останавливается (строки 17–18). Ситуация, когда

после первого сканирования все подпоследовательности отброшены как заведомо непохожие на запрос, означает, что выбрано слишком жесткое ограничение полосы Сако—Чиба, и параметр r необходимо уменьшить.

Далее выполняется параллельное вычисление DTW-расстояния между каждой строкой матрицы кандидатов и запросом (строки 22–29). Вычисления производятся итеративно для групп по p строк, чтобы обеспечить баланс загрузки нитей. На каждой итерации мы автоматически получаем минимум DTW-расстояния и обновляем bsf и профиль расстояния, применяя параллельную свертку (клауза `reduction` в директиве `#pragma`, строка 22). В элемент битовой карты, соответствующий обработанной подпоследовательности, записывается значение `FALSE`, чтобы обеспечить однократную обработку каждого кандидата. Описанная деятельность продолжается пока не будет улучшено (уменьшено) значение bsf либо не будут просмотрены все кандидаты без улучшения значения bsf. Если значение bsf улучшено, алгоритм переходит к началу, вычисляя битовую карту и пытаясь отбросить как бесперспективные оставшиеся подпоследовательности.

По окончании работы алгоритма в элементах профиля расстояния записано значение DTW-расстояния между запросом и соответствующей подпоследовательностью, либо значение $+\infty$, которое означает, что данная подпоследовательность заведомо не похожа на запрос, причем в этом случае значение расстояния не вычислялось.

Алг. 2 SCORING(IN: $DistProfile^1, \ldots, DistProfile^d$, k; OUT: $kNNset$)

1: $Rating \leftarrow \overline{0}$; $s \leftarrow 0$; $kNNset \leftarrow \emptyset$
2: #pragma omp parallel for
3: **for** $i \in 1..N$ **do**
4: **for** $j \in 1..d$ **do**
5: $Rating(i) \leftarrow Rating(i) + \frac{1}{DistProfile^j(i)+\varepsilon} \cdot \frac{d-j+1}{d}$
6: **while** `TRUE` **do**
7: $s \leftarrow s + 1$
8: $maxRating \leftarrow \max\limits_{i \in 1..N} Rating(i)$; $maxIdx \leftarrow \arg\max\limits_{i \in 1..N} Rating(i)$
9: **if** $|maxIdx - prevIdx| > m$ **then**
10: $kNNset \leftarrow kNNset \cup \{T_{maxIdx,\,m}\}$; $prevIdx \leftarrow maxIdx$
11: **else**
12: $Rating(maxIdx) \leftarrow -\infty$
13: **if** $|kNNset| = k$ **or** $s = N$ **then**
14: **break**

Алг. 2 показывает псевдокод шага скоринга, на котором выполняется отбор подпоследовательностей исходного ряда для восстановления пропущенного значения. Сначала на основе профилей расстояния всех опорных рядов, вычисленных на предыдущем шаге, параллельно вычисляется рейтинг интервалов (строки 1–5). Вычисления выполняются с помощью двух вложенных циклов: внешний — по интервалам, внутренний — по опорным рядам; внешний цикл распараллеливается. После этого в векторе $Rating$ мы находим top-k наибольших элементов так, чтобы их индексы различались не менее чем на m (строки 6–14).

Рейтинг подпоследовательности $T_{i,\,m}$ определяется следующим образом:

$$Rating(T_{i,\,m}) = \sum_{s=1}^{d} \frac{w(s,i)}{DistProfile^s(i) + \varepsilon}, \tag{12}$$

где $DistProfile^s(i)$ — DTW-расстояние между запросом Q^s и подпоследовательностью $R^s_{i,\,m}$ (запрос и i-я подпоследовательность s-го опорного ряда соответственно), ε — машинный эпсилон (верхняя граница относительной ошибки из-за округления в арифметике с плавающей

точкой), $w : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ — весовая функция. В данной формуле весовая функция используется для учета корреляции между восстанавливаемым и опорным временными рядами (например, для предпочтения совпадений в более коррелированных опорных временных рядах) и положения, с которого начинается подпоследовательность (например, для предпочтения совпадений в наиболее недавних по времени подпоследовательностях). В нашем исследовании мы берем простую весовую функцию: $w(s, i) = \frac{d-s+1}{d}$ (больший вес имеет подпоследовательность более коррелированного временного ряда, индекс подпоследовательности не учитывается).

В итоге реконструкция пропущенного значения t_n выполняется следующим образом:

$$\tilde{t}_n = \frac{1}{k} \sum_{T_{i,\,m} \in kNNset} t_m. \tag{13}$$

5. Вычислительные эксперименты

Для оценки эффективности алгоритма IDK нами были проведены вычислительные эксперименты, в которых исследовались следующие показатели: точность восстановления, производительность и ускорение алгоритма.

Для оценки *точности восстановления* пропущенных значений нами используется мера среднеквадратичной ошибки *RMSE (Root Mean Square Error)*, определяемая следующим образом:

$$RMSE = \sqrt{\frac{1}{h} \sum_{i=1}^{h} (x_i - \tilde{x}_i)^2}, \tag{14}$$

где x_i — фактическое значение (считающееся пропущенным), $\tilde{x}_i$ — восстановленное значение, h — количество пропущенных значений временного ряда. Точность восстановления измерялась для 100 последних точек заданного временного ряда.

В экспериментах использовался временной ряд Chlorine [3], использованный в исследованиях проекта SPIRIT [18] и алгоритма TKCM [17]. Этот синтетический набор данных был создан путем моделирования системы распределения питьевой воды; он описывает концентрацию хлора в 166 точках соединения за период времени 4310 временных точек (15 дней) с частотой дискретизации 5 минут. Распространение уровня хлора в системе вызывает фазовые сдвиги в наборе данных.

Производительность P алгоритма понимается как время, затрачиваемое на вычисление восстановленного значения. В экспериментах запуск программы осуществлялся 10 раз и в качестве итогового времени вычисления использовалось медианное значение. *Ускорение* S параллельного алгоритма, запускаемого на p нитях, вычисляется как отношение производительности алгоритма, запущенного на одной нити, к производительности алгоритма при запуске на p нитях:

$$S(p) = \frac{P_1}{P_p}. \tag{15}$$

Эксперименты были проведены на оборудовании Лаборатории суперкомпьютерного моделирования ЮУрГУ [10]: процессор Intel Xeon Gold 6254 (Cascade Lake, 18 ядер по 4 GHz).

Результаты экспериментов представлены на рис. 2. Можно видеть, что разработанный алгоритм IDK демонстрирует несколько лучшую точность восстановления по мере RMSE (около 20%) и более высокое быстродействие (примерно в 2 раза), чем алгоритм TKCM. При этом IDK показывает устойчивое сублинейное ускорение.

6. Заключение

В работе затронута проблема восстановления пропущенных значений в потоковом временном ряде в режиме реального времени, которая встречается в настоящее время в ши-

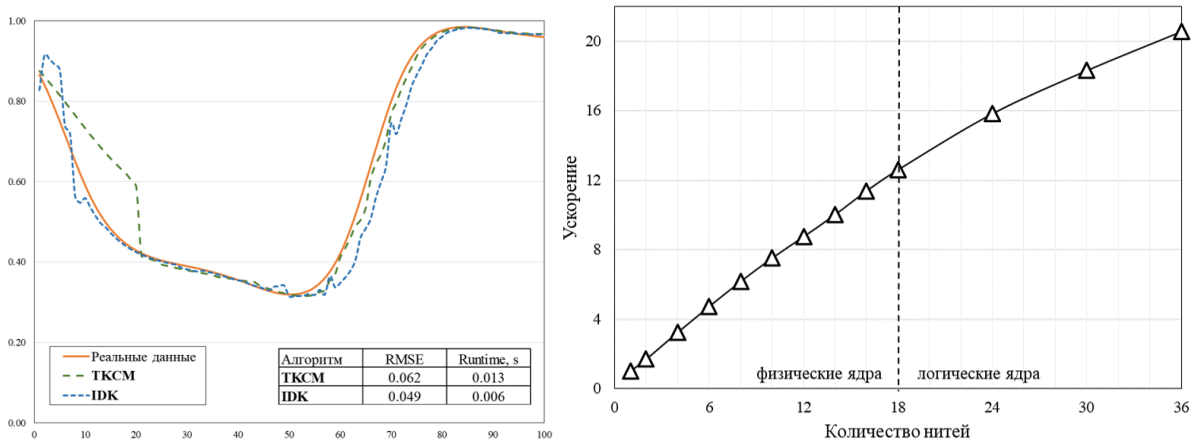

Рис. 2. Эффективность алгоритма IDK

роком спектре прикладных и научных задач (приложения Интернета вещей, персональная медицина, цифровая индустрия и др.). Пропуски, возникающие в таких приложениях ввиду технических сбоев или человеческого фактора, как правило, неприемлемы и подлежат замене на правдоподобные значения, синтезированные на лету.

В статье предложен новый параллельный алгоритм для многоядерного процессора, получивший название IDK (Imputation based on DTW and K Nearest Neighbors), который выполняет восстановление пропущенного значения потокового временного ряда в режиме реального времени. Параллелизм вычислений реализован с помощью технологии программирования OpenMP. IDK работает в предположении, что имеется набор опорных временных рядов (reference time series), которые имеют семантическую связь с исходным рядом. Опорные временные ряды упорядочиваются по убыванию корреляции с исходным рядом. Алгоритм применяет следующую эвристику: если в опорных рядах имеют место повторяющиеся (схожие) подпоследовательности, то в ряде, содержащем пропущенное значение, повторяющиеся подпоследовательности возникают в тех же временны́х интервалах.

В вычислительных экспериментах с референсным временным рядом IDK продемонстрировал приемлемое качество и высокое быстродействие восстановления, что позволяет применять предложенный алгоритм в режиме реального времени.

В качестве направления продолжения исследований мы рассматриваем проведение вычислительных экспериментов на временных рядах из различных предметных областей и сравнение полученных результатов с другими известными алгоритмами восстановления пропущенных значений.

Литература

1. Batista G.E.A.P.A., Monard M.C. An Analysis of Four Missing Data Treatment Methods for Supervised Learning // Applied Artificial Intelligence. 2003. Vol. 17, No. 5-6. P. 519–533. DOI: 10.1080/713827181

2. Berndt D.J., Clifford J. Using Dynamic Time Warping to Find Patterns in Time Series // Proceedings of the 1994 AAAI Workshop on Knowledge Discovery in Databases, Seattle, Washington, July 1994. AAAI Press, 1994. P. 359–370.

3. Chlorine Dataset. https://www.cs.cmu.edu/afs/cs/project/spirit-1/www/ (дата обращения: 03.09.2021).

4. Ding H., Trajcevski G., Scheuermann P., Wang X., Keogh E. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures //

Proceedings of the VLDB Endowment. 2008. Vol. 1, No. 2. P. 1542–1552.

5. Fu A.W., Keogh E.J., Lau L.Y.H., Ratanamahatana C.A. Scaling and Time Warping in Time Series Querying // Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 – September 2, 2005. ACM, 2005. P. 649–660.

6. Hsu H.H., Yang A.C., Lu M.-D. KNN-DTW Based Missing Value Imputation for Microarray Time Series Data // Journal of Computers. 2011. Vol. 6, No. 3. P. 418–425. DOI: 10.4304/jcp.6.3.418-425

7. Keogh E.J., Pazzani M.J. Derivative Dynamic Time Warping // Proceedings of the 1st SIAM International Conference on Data Mining, SDM 2001, Chicago, IL, USA, April 5–7, 2001. P. 1–11. DOI: 10.1137/1.9781611972719.1

8. Khayati M., Lerner A., Tymchenko Z., Cudré-Mauroux P. Mind the Gap: An Experimental Evaluation of Imputation of Missing Values Techniques in Time Series // Proc. VLDB Endow. 2020. Vol. 13. no. 5. P. 768–782. DOI: 10.14778/3377369.3377383

9. Kim S.-W., Park S., Chu W.W. An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases // Proceedings of the 17th International Conference on Data Engineering, ICDE 2001, April 2–6, 2001, Heidelberg, Germany. P. 607–614.

10. Kostenetskiy P., Semenikhina P. SUSU Supercomputer Resources for Industry and fundamental Science // Proceedings - 2018 Global Smart Industry Conference, GloSIC 2018. Article 18308632. IEEE, 2018. DOI: 10.1109/GloSIC.2018.8570068

11. Kraeva Ya., Zymbler M. Scalable Algorithm for Subsequence Similarity Search in Very Large Time Series Data on Cluster of Phi KNL // 20th International Conference on Data Analytics and Management in Data Intensive Domains, DAMDID/RCDL 2018, October 9–12, 2018, Moscow, Russia. Revised Selected Papers. Communications in Computer and Information Science. Springer, 2019. Vol. 1003. P. 149–164. DOI: 10.1007/978-3-030-23584-0_9.

12. Mattson T. S08 - Introduction to OpenMP // Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11–17, 2006, Tampa, FL, USA. 2006. P. 209. DOI: 10.1145/1188455.1188673.

13. Phan T.-T.-H., Poisson É., Bigand A., Lefebvre A. DTW-Approach for Uncorrelated Multivariate Time Series Imputation // Proceedings of the 27th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2017, Tokyo, Japan, September 25–28, 2017. P. 1–6. DOI: 10.1109/MLSP.2017.8168165

14. Phan T.-T.-H., Poisson É., Lefebvre A., Bigand A. Dynamic Time Warping-based Imputation for Univariate Time Series Data // Pattern Recognit. Lett. 2020. Vol. 139. P. 139–147. DOI: 10.1016/j.patrec.2017.08.019

15. Rakthanmanon T., Campana B.J.L., Mueen A., Batista G.E.A.P.A., Westover M.B., Zhu Q., Zakaria J., Keogh E.J. Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping // Transactions on Knowledge Discovery in Databases. 2013. Vol. 7, No. 3. P. 10:1–10:31.

16. Troyanskaya O.G., Cantor M.N., Sherlock G., Brown P.O., Hastie T., Tibshirani R., Botstein D., Altman R.B. Missing Value Estimation Methods for DNA Microarrays // Bioinformatics. 2001. Vol. 17, No. 6. P. 520–525. DOI: 10.1093/bioinformatics/17.6.520

17. Wellenzohn K., Böhlen M.H., Dignös A., Gamper J., Mitterer H. Continuous Imputation of Missing Values in Streams of Pattern-Determining Time Series // Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21–24, 2017. P. 330–341. DOI: 10.5441/002/edbt.2017.30

18. Papadimitriou S., Sun J., Faloutsos C. Streaming Pattern Discovery in Multiple Time-Series // Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 – September 2, 2005. P. 697–708. URL: https://dl.acm.org/doi/pdf/10.5555/1083592.1083674