Оглавление

Введение

Поиск аномальных подпоследовательностей (существенно непохожих на все остальные подпоследовательности) временного ряда является одной из наиболее востребованных задач интеллектуального анализа временных рядов и в настоящее время применяется в широком спектре практических приложений: моделирование климата, финансовые прогнозы, медицинские исследования и др.

В работе [1] Кеогом (Keogh) и др. предложен термин диссонанс, уточняющий понятие аномальной подпоследовательности временного ряда. Диссонанс определяется как подпоследовательность ряда, которая имеет максимальное расстояние (минимальную схожесть) до своего ближайшего соседа. Ближайшим соседом подпоследовательности является та подпоследовательность ряда, которая не пересекается с данной и имеет минимальное расстояние до нее (максимальную схожесть) в смысле выбранной меры схожести.

Позже, работе [2], Кеог и др. предложили алгоритм поиска диссонансов для случая, когда временной ряд не помещается в оперативную память компьютера. В ней авторы показали, что алгоритм может быть распараллелен с помощью парадигмы MapReduce. В свою очередь авторы не затронули возможность распараллеливания алгоритма внутри вычислительного узла.

Данный отчет представляет собой описание алгоритма поиска диссонансов в сверхбольшом временном ряде для вычислительных кластеров с многоядерными ускорителями.

Далее отчет построен следующим образом. В разделе 1 даются основные определения, понятия и формальная постановка задачи, приведены все использующиеся в документе обозначения. В разделе 2 приведено описание параллельного алгоритма поиска диссонансов во временном ряде PDADD, описаны методы и структуры данных, применяемые при реализации алгоритма. В приложении А указаны все обозначения, которые будут использоваться в данной работе. В приложении Б представлены все структуры данных, использующиеся при реализации данного алгоритма.

1. Постановка задачи

1.1. Формальные определения и нотация

Временной ряд (time series) T представляет собой последовательность хронологически упорядоченных вещественных значений: $T = (t_1,\ t_2 \dots ,\ t_m), t_i \in \mathbb{R}$.

Подпоследовательность (subsequence) $T_{i,n}$ временного ряда T представляет собой непрерывное подмножество *T* из *n* элементов начиная с позиции *i*: $T = (t_i,\ t_{i+1} \dots ,\ t_{i+n-1}),\ 1 \le n \le m,\ 1 \le i \le N,\ N = m - n + 1$. Подпоследовательности $T_{i,n}$ и $T_{j,n}$ *называются непересекающимися (non-self match)*, если $|i - j| \le n$. Подпоследовательность, которая является непересекающейся к данной подпоследовательности C, обозначается как MC.

Подпоследовательность D ряда S является *диссонансом (discord),* если

$$\forall C,\ M_C \in S,\ M_D \in S \qquad \min\big(Dist(D, M_D)\big) \ge \min(Dist(C, M_C) \qquad (1)$$

Отметим, что функцией $Dist$ может являться любая метрика, вычисляющая расстояние между двумя подпоследовательностями.

В приложении A указаны все обозначения, которые будут использоваться в данной работе.

2. Последовательный алгоритм DDAD

Алгоритм DADD [2] состоит из двух фаз: фаза поиска и уточнения. Во время протекания первой фазы алгоритма производится нормализация ряда и поиск кандидатов в диссонансы. На этапе уточнения происходит проверка всех кандидатов и удаление тех, что не являются диссонансами.

Z-нормализация временного ряда T представляет собой преобразование каждого элемента временного ряда согласно формуле 2.

$$\hat{t}_i = \frac{t_i - \mu}{\sigma}, 1 \le i \le m;\ \mu = \frac{1}{m}\sum_{i=1}^{m} t_i\ ;\ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} {t_i}^2 - \mu^2 \qquad (2)$$

После нормализации среднее арифметическое элементов временного ряда приблизительно равно 0, а среднеквадратичное отклонение близко к 1.

В статье [2] для определения значимых диссонансов вводится параметр r – *диапазон определения диссонанса (range discords)*. Этот параметр определяет минимальное значение расстояния до ближайшего соседа подпоследовательности C такое, чтобы она считалась диссонансом.

Введём понятие диапазонного диссонанса – такой диссонанс, для которого расстояние до ближайшего соседа входит в диапазон определения диссонанса. Формула определения подпоследовательности D как диапазонного диссонанса имеет вид:

$$\min(Dist(D,\ M_D) > r \tag{3}$$

Определение кандидата в диссонансы происходит следующим образом. Для первой фазы алгоритма ведется список кандидатов C, который изначально пустой. Каждая подпоследовательность ряда P сравнивается с каждым кандидатом из этого списка. Если выполняется условие (4), то подпоследовательность P добавляется в C, иначе подпоследовательности C_i, для которых это условие ложно, удаляется из списка C.

$$\forall C_i \in \mathrm{C},\ \mathrm{P} \in \mathrm{S} \quad \min(Dist(P, C_i) > r \tag{4}$$

Определение диссонансов происходит во время выполнения фазы уточнения, на основе списка кандидатов, полученных после выполнения фазы поиска. Нахождение диссонансов заключается в сравнении каждой подпоследовательности из списка кандидатов C со всеми непересекающимися подпоследовательностями ряда M_C и удаление таких кандидатов C_i, для которых выполняется условие (5).

$$\forall C_i \in \mathrm{C},\ M_C \in S \quad \min(Dist(C_i, M_C) \leq r \tag{5}$$

Подпоследовательности, оставшиеся в списке кандидатов после выполнения фазы уточнения, являются диссонансами.

Распараллеливание алгоритма DADD между узлами вычислительного кластера основывается на статье [2], в которой Keog и др. представили параллельную версию своего алгоритма на базе парадигмы MapReduce [3] (эту версию далее будем называть MR-DADD). Распараллеливание заключается в разделении исходного ряда на непересекающиеся фрагменты. Эти фрагменты обрабатываются на узлах независимо друг от друга, а узлы обмениваются информацией только после выполнения каждой фазы алгоритма (рис. 1).

3. Параллельный алгоритм PDADD

Распараллеливание алгоритма DADD внутри узла с многоядерными ускорителями выполнено с помощью технологии OpenMP [4]. Указанная технология предполагает параллелизм уровня нитей, внедряемый программистом в исходный текст последовательной программы с помощью директив компилятора (например, для циклов с фиксированным количеством повторений — это директива #pragma omp parallel for). Распараллеливание основано на следующих основных идеях: раздельное распараллеливание циклов перебора подпоследовательностей, использование квадрата евклидова расстояния и матричное представление данных, косвенная адресация.

2.1. Реализация

Параллельный поиск диссонансов временного ряда PDADD состоит из четырех последовательно выполняемых стадий: стадия поиска, стадия сбора списка кандидатов, стадия уточнения и стадия сбора списка диссонансов (см. алгоритмы 1 – 4).

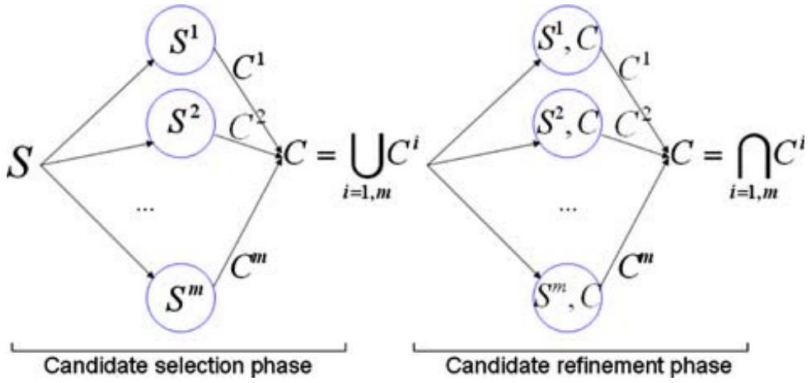

Рис. 1. Обмен информацией между узлами для алгоритма MR-DADD

Изначально исходный временной ряд подвергается z-нормализации. Да-

Алгоритм 1: Поиск кандидатов в диссонансы во временном ряде (ВХОД *S, C, r*; ВЫХОД *I, Bottom, Count*)

```
iam ← omp_get_thread_num()
#pragma omp for
for i from 1 to N do
        isCandidate ← TRUE
        for j from 1 to Bottom(iam) do
                if I(iam, j) ≠ NIL then
                        dist ← ED²(S(i, ·), S(I(iam, j), ·))
                        if dist < r then
                                isCandidate ← FALSE
                                I (iam, j) ← NIL
                                Insert (iam) ← j
                                Count (iam) ← Count (iam) – 1
                        end if
                end if
        end for
        if isCandidate then
                if  I(iam, Insert(iam)) ≠ NIL then
                        I (iam, Insert(iam))← i
                else
                        I (iam, Bottom (iam)) ← i
                        Bottom (iam) ← Bottom (iam) + 1
                end if
                Count (iam) ← Count (iam) + 1
```

```
        end if
end for
return I, Bottom, Count
```

лее из получившегося временного ряда $\widehat{T}$ формируются матрицы подпоследовательностей S. Эти матрицы распределяются между узлами вычислитель-

Алгоритм 2: Сбор списка кандидатов (ВХОД *S, I, Bottom, Count*; ВЫХОД *H, C, Candidats*)

```
H ← 0
for k from 1 to p do
        H ← H + Count(k)
i ← 0
for k from 1 to p do
        for j from 1 to Bottom(k) do
                if  I(k, j) ≠ NIL then
                        C(i, · ) ← S(I(k, j), · )
                        Candidats (i) ← I (k, j)
                        i ← i + 1
return H, C, Candidats
```

Алгоритм 3: Уточнение диссонансов (ВХОД *S, C, r*; ВЫХОД *B*)

```
iam ← omp_get_thread_num()
#pragma omp for
B ← TRUE_{p×H}
for i from 1 to N do
        for j from 1 to H do
                if B(iam, j) then
                        dist ← ED²(S(i, ·), C(j, ·))
                        B(iam, j) ← dist > r
                end if
        end for
end for
return B
```

Алгоритм 4: Сбор списка диссонансов (ВХОД *Candidats, B*; ВЫХОД D, *Discords*)

```
D ← 0
for j from 1 to H do
        isDiscord ← TRUE
        for k from 1 to p do
                isDiscord ← isDiscord and B (k, j)
        if isDiscord then
                Discords (D) ← Candidats (j)
                D ← D + 1
        end if
end for
return D, Discords
```

ного кластера. Размер этих матриц определяется таким, чтобы вся матрица подпоследовательностей могла поместиться в оперативной памяти узла.

Для поиска диссонансов в данной алгоритме определим функцию Dist как квадрат евклидова расстояния. Данный метод был выбран для использования возможности векторизации вычислений при вычислении расстояния между двумя подпоследовательностями. Пусть имеются подпоследовательности X и Y длины n временного ряда T, тогда квадрат евклидова расстояния ED^2 между X и Y вычисляется как

$$ED^2(X,Y) = \sum_{i=1}^{n}(x_i - y_i)^2 \quad (6)$$

Все четыре стадии выполняются для каждого вычислительно узла. Стадии поиска и уточнения занимают наибольшую часть времени выполнения предлагаемого алгоритма из-за наличия перебора всей матрицы подпоследовательностей.

На рисунках 2 и 3 приведены файловая и модульная структуры всей программы.

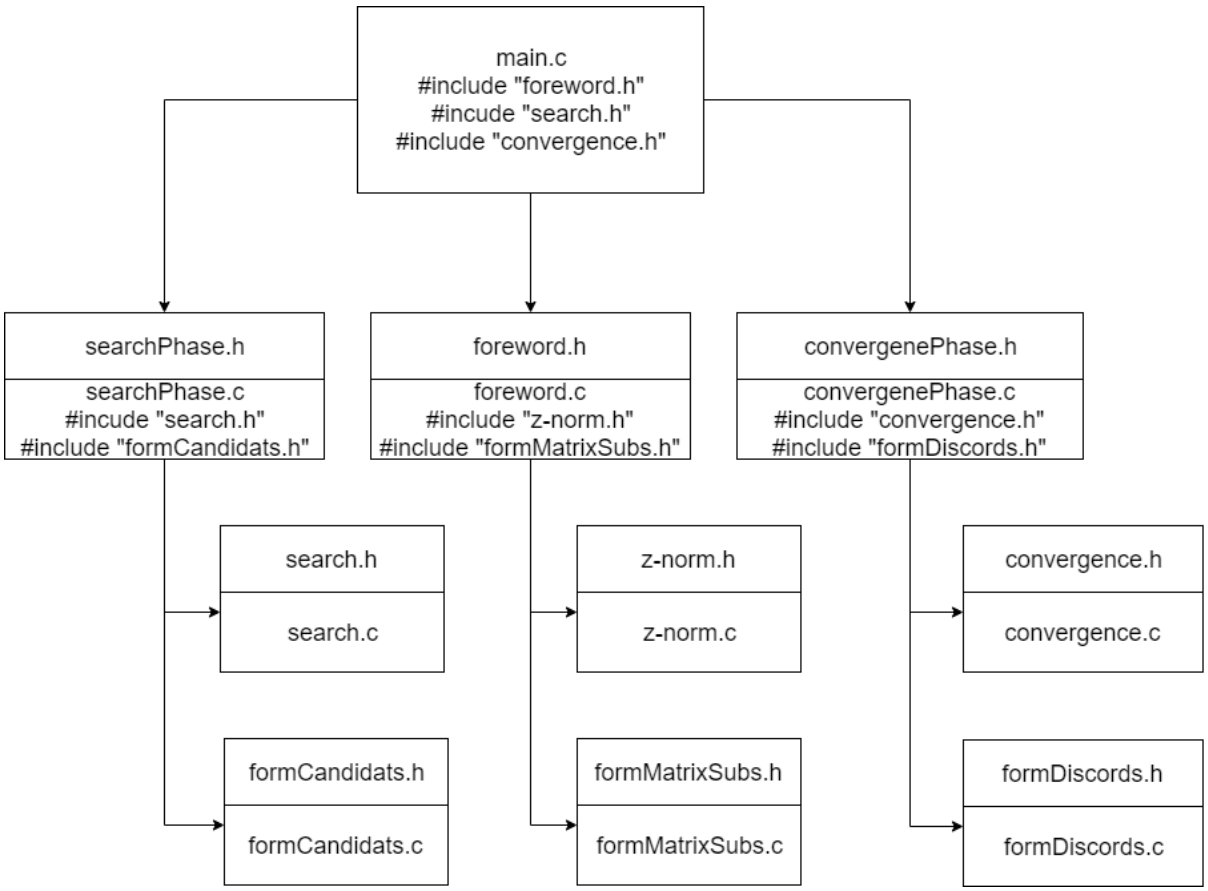

Рис. 2. Файловая структура программы

Стадии поиска и уточнения дополнительно распараллеливаются внутри вычислительного узла. Стадии сбора списка кандидатов и сбора списков диссонансов осуществляют обмен информацией о кандидатах (диссонансах), найденных на каждом вычислительном узле, между всеми узлами кластера.

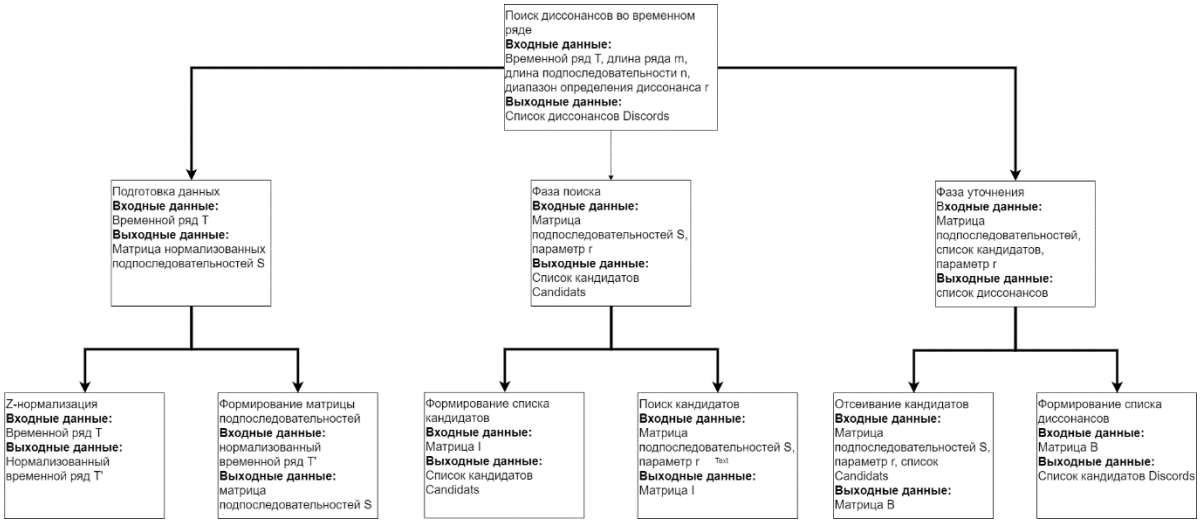

Рис. 3. Модульная структура программы

2.2. Структуры данных

Предлагаемый алгоритм PDADD использует следующие основные структуры для хранения данных внутри узла (см. приложение Г).

Исходный временной ряд представляется в виде матрицы выровненных подпоследовательностей для обеспечения эффективной векторизации вычислений. Выравнивание подпоследовательностей выполняется следующим образом. Пусть w – ширина векторного регистра ускорителя. Если длина подпоследовательности n не кратна w, то каждая подпоследовательность ряда дополняется фиктивными нулевыми элементами. Обозначим количество фиктивных элементов за pad, $pad := w - (n \bmod w)$, тогда выровненная подпоследовательность $\widetilde{\mathrm{T}}_{i,n}$ определяется следующим образом:

$$\widetilde{\mathrm{T}}_{i,n} := \begin{cases} (t_i, t_{i+1}, \dots, t_{i+n-1}), 0, 0, \dots, 0 & \text{if } n \bmod w > 0 \\ (t_i, t_{i+1}, \dots, t_{i+n-1}) & \text{otherwise} \end{cases} \quad (7)$$

Матрица подпоследовательностей $\mathrm{S}_{\mathrm{T}}^{\mathrm{n}}$ временного ряда $\widetilde{\mathrm{T}}$ представляет собой упорядоченное множество всех подпоследовательностей временного ряда: $\mathrm{S} = \left(\widetilde{\mathrm{T}}_{1,n}, \widetilde{\mathrm{T}}_{2,n}, \dots, \widetilde{\mathrm{T}}_{N,n}\right), \widetilde{\mathrm{T}}_{i,n} \in \mathbb{R}^{n+pad}\ 1 \le i \le N$

Пусть доля кандидатов к количеству всех подпоследовательностей будет обозначаться как δ. Тогда максимально возможное количество кандидатов будет равно $L := \lceil \delta \cdot N \rceil$.

Также, введем обозначения p как количества нитей для узла и P как количество узлов.

Индексы кандидатов $\mathrm{I} \in \mathbb{N}^{p \times \mathrm{L}}$ представляет собой матрицу, внутри каждой строки которой содержится информация для найденных данной нитью кандидатов. Т.е. каждой строке этой матрицы соответствует одна нить, и только она изменяет данную строку.

Матрица I будет использоваться не полностью в связи с тем, что длина каждой строки этой матрицы – это допущение, которое было установлено относительно максимального числа кандидатов. Для обозначения того, что элемент матрицы не несет в себе информацию о кандидате, в нем хранится специальный маркер **NIL**. Данная логика отображена в формуле 4 (Candidats – множество кандидатов в диссонансы).

$$\forall I(iam, i) \neq \mathbf{NIL}\ \exists\ S(I(iam, i), \cdot) \in Candidats, 1 \le i \le L,\ 1 \le iam \le p \quad (8)$$

Для того, чтобы добавлять элементы в матрицу I без перебора всех ее элементов, в алгоритме присутствуют вспомогательные массивы Insert, Bottom и Count.

Insert $\in \mathbb{N}^p$ – *массив индексов второго уровня, указывающий на последний удаленный кандидат.*

Insert $\in \mathbb{N}^p$ – *массив, осуществляющий добавление нового индекса кандидата в массив I.*

Этот массив организован таким образом, что в ячейку Insert(iam) записывается индекс последнего элемента в строке I (iam, ·), куда был записан маркер **NIL**.

Bottom ∈ $\mathbb{N}^p$ – *массив максимальных занятых* (т.е. не содержащих **NIL**) *ячеек* в матрице I.

$$0 \le i \le L,\ \ I(iam, i) \ne \textbf{NIL} \qquad Bottom(iam) = \max(i) \tag{9}$$

Count ∈ $\mathbb{N}^p$ – *массив количества значимых элементов* в матрице I.

$$Count(iam) = |\{0 \le i \le L \mid I(iam, i) \ne \textbf{NIL}\}| \tag{10}$$

Использование матрицы I как представление списка кандидатов позволяет реализовать распараллеливание вычислений внутри узла для этапа поиска. Взаимосвязь матрицы I с массивами Insert и Bottom проиллюстрирована на рисунке 4.

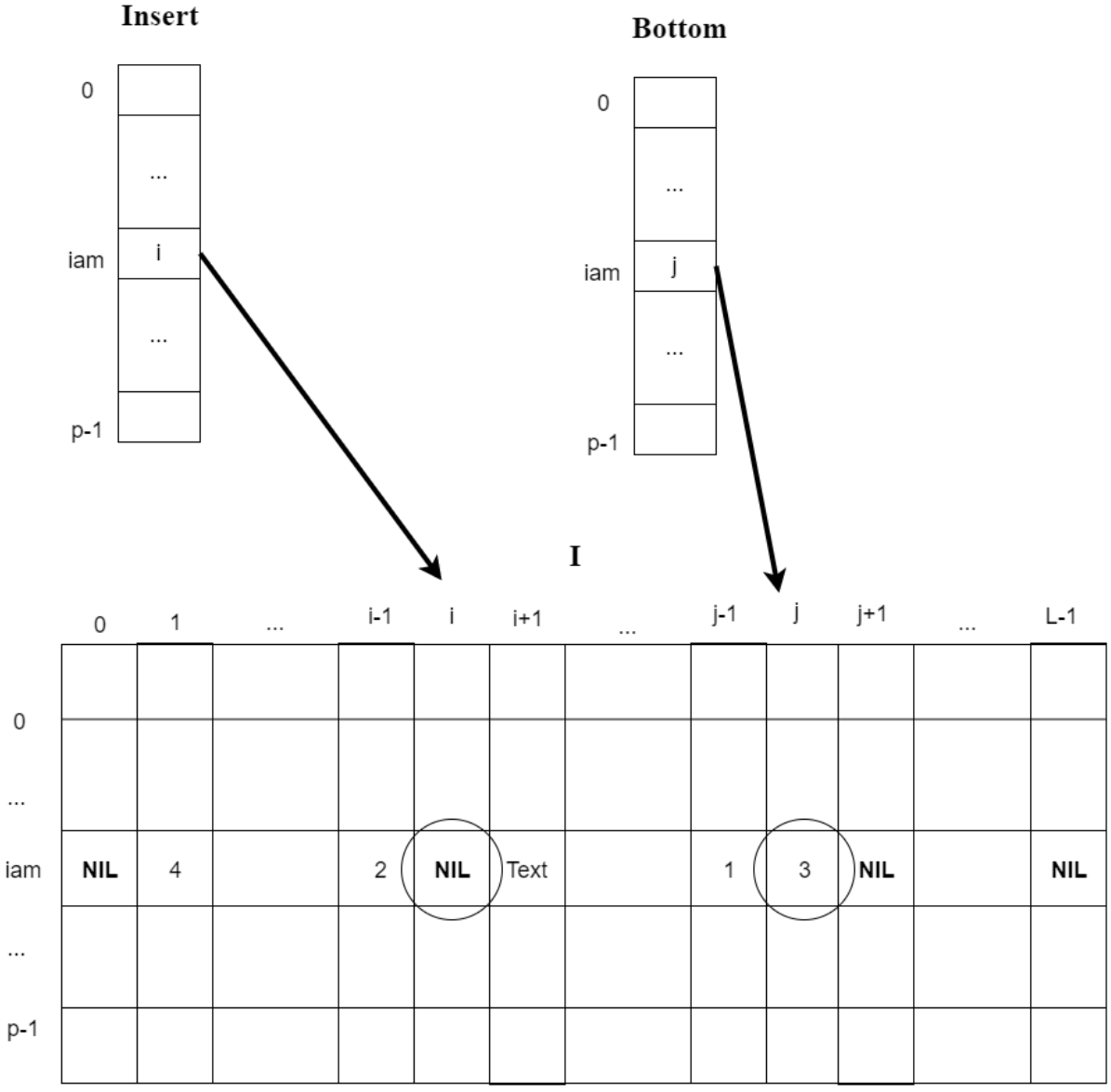

Рис. 4. Взаимосвязь матрицы I с массивами Insert и Bottom

Общий для узла список кандидатов Candidats $\in \mathbb{N}^{\mathrm{H}}$ представляет собой массив, содержащий в себе информацию о всех найденных на данном узле кандидатах.

$$\text{Candidats } = \{0 \leq i \leq L, 0 \leq k \leq p \mid I(k,i) \neq \textbf{NIL}\} \quad (11)$$

Количество найденных кандидатов обозначим как $\mathrm{H} \coloneqq |Candidats|$.

Битовая карта кандидатов B $\in B^{p \times \mathrm{H}}$ представляет собой матрицу, где в каждой строке хранится информация о каждом кандидате.

Если B (k, i) = **TRUE**, то для нити с номером k подпоследовательность Candidats(i) является кандидатом, иначе она таковой не является.

Использование битовой карты B позволяет реализовать распараллеливание вычислений внутри узла для этапа уточнения.

Количество найденных диссонансов обозначим как $\mathrm{D} \coloneqq |Discords|$.

Список диссонансов Discords $\in \mathbb{N}^{\mathrm{D}}$ представляет собой массив, содержащий в себе информацию о всех найденных на данном узле диссонансах.

$$\text{Discords} = \{0 \leq i \leq L, \quad B(1,i) \wedge \ldots \wedge B(p,i) = \textbf{TRUE} \mid Candidats(i)\} \quad (12)$$

Список литературы

1. E. Keogh, J. Lin, and A. Fu, “HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence,” in Proc. 5th IEEE Int. Conf. on Data Mining, Houston, USA, November 27–30, 2005 (IEEE Press, New York, 2005), pp. 226–233.
2. D. Yankov, E. Keogh, and U. Rebbapragada, “Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets,” Knowl. Inf. Syst. **17** (2), 241–262 (2008).
3. MapReduce: Simplified Data Processing on Large Clusters – Google AI [Электронный ресурс] URL: https://ai.google/research/pubs/pub62 (дата обращения: 22.10.2019).
4. OpenMP [Электронный ресурс] URL: https://www.openmp.org/ (дата обращения: 22.10.2019).

Приложение А
Обозначения, используемые в документе

Обозначение	**Значение**	**Область значений**	**Пояснение**
NIL	-1	$\mathbb{N}$	пустой индекс
r		$\mathbb{R}$	диапазон определения диссонанса
m		$\mathbb{N}$	длина ряда
n		$\mathbb{N}$	длина подпоследовательности
p		$\mathbb{N}$	количество нитей
P		$\mathbb{N}$	количество узлов
iam		$\mathbb{N}$	номер текущей нити
δ		(0,1)	Параметр, отвечающий за то, какое количество кандидатов программа будет считать максимально возможным
N	m-n+1	$\mathbb{N}$	количество подпоследовательностей
L	$\lceil \delta \cdot N \rceil$	$\mathbb{N}$	максимально возможный размер матрицы кандидатов
H		$\mathbb{N}$	размер матрицы кандидатов
D		$\mathbb{N}$	количество диссонансов
pad		$\mathbb{N}$	дополнение размера подпоследовательности ряда до кратности размеру векторного регистра
I		$\mathbb{N}^{p \times \mathrm{L}}$	индексы кандидатов (для каждой нити)
B		$B^{p \times \mathrm{H}}$	битовая карта кандидатов (для каждой нити)
Bottom		$\mathbb{N}^{p}$	максимальный индекс в I, где элемент не равен **NIL** (для каждой нити)
Count		$\mathbb{N}^{p}$	количество всех элементов в I, не равных **NIL** (для каждой нити)
Insert		$\mathbb{N}^{p}$	индекс в I, в который можно записать данные (для каждой нити)
Candidats		$\mathbb{N}^{\mathrm{H}}$	индексы кандидатов (для узла)
Discords		$\mathbb{N}^{\mathrm{D}}$	индексы диссонансов (для узла)
S		$\mathbb{R}^{\mathrm{N} \times (\mathrm{n}+\mathrm{pad})}$	матрица подпоследовательностей временного ряда
C		$\mathbb{R}^{\mathrm{H} \times (\mathrm{n}+\mathrm{pad})}$	матрица кандидатов в диссонансы

Приложение Б
Структуры данных, используемые в алгоритме