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 2018

ОГЛАВЛЕНИЕ

1. Проектирование 3
1.1. Термины и определения 3
1.2. Наивный алгоритм на основе матрицы расстояний 4
1.3. Параллельный HOTSAX алгоритм 8
1.4. Выводы по разделу 12
Литература 13

1. Проектирование

Данный раздел содержит описание особенностей проектирования разработанных алгоритмов, и материал раздела организован следующим образом. В подразделе 1.1 дают основные термины и определения, в подразделе 1.2 приведено описание наивного алгоритма поиска диссонансов временного ряда на основе матрицы расстояний между подпоследовательностями. В подразделе 1.3 приведено описание параллельного HOTXSAX алгоритма поиска диссонансов временного ряда. Подраздел 1.4 содержит выводы по разделу.

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:

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

1.2. Наивный алгоритм на основе матрицы расстояний

Один из подходов решения проблемы поиска диссонансов временного ряда основан на построении матрицы расстояний между всевозможными его подпоследовательностями. Алгоритм заключается в том, что нужно построить матрицу расстояний до ближайших соседей для всех подпоследовательностей временного ряда, найти расстояние до ближайшего соседа для всех подпоследовательностей временного ряда, а затем найти максимальное из найденных минимальных расстояний.

Данный алгоритм эффективно поддается параллелизации и векторизации, т.к. нахождение расстояния между подпоследовательностями является векторной операцией, а сам процесс вычисления расстояния между двумя подпоследовательностями хорошо распараллеливается, так как вычисление каждого из расстояний не зависит от вычисления расстояний между другими подпоследовательностями. Еще одним преимуществом данного алгоритма является то, что необходим всего один интуитивно понятный параметр n - длина подпоследовательности-диссонанса. К недостаткам алгоритма можно отнести высокую вычислительную сложность ($O(n^3)$) и большие требования к размеру используемой памяти (размер необходимой для работы памяти растет квадратично с ростом временного ряда, в котором нужно найти диссонансы).

Входными параметрами алгоритма являются:

1. T – временной ряд ($T = \{t_i\}$, $1 \leq i \leq m$)
2. m – длина ряда.
3. n – длина подпоследовательности
4. C – множество подпоследовательностей.

В памяти хранятся следующие структуры данных:

1. временной ряд T в виде вектора размера $m \times 1$ (см. рисунок 1)
2. матрица подпоследовательностей размера $m - n + 1 \times n$ (см. рисунок 1)
3. матрица расстояний между подпоследовательностями временного ряда размера $m - n + 1 \times m - n + 1$ (см. рисунок 2)

В результате работы алгоритма получаем:

1. bsf_loc – позиция начала диссонанса временного ряда
2. bsf_dist – расстояние до ближайшего соседа подпоследовательности-диссонанса.

Алгоритм состоит из следующих шагов:

1. Подготовка: составить вектор C из всех подпоследовательностей временного ряда.
2. Вычислить матрицу расстояний D каждой подпоследовательности временного ряда с каждой. Расстояния между подпоследовательностями C_i и C_j вычисляются по формуле:

$$D_{i,j} = \begin{cases} \sum_{k=1}^{n}(C_i^k - C_j^k)^2, & \text{если } i \neq j \text{ и } (j \leq i \urcorner n \text{ или } j \geq i+n), \\ & 1 \leq i, j \leq m-n+1; \\ \infty, & \text{иначе} \end{cases} \quad (2)$$

3. В каждой i-й строке матрицы D находим минимальный элемент D_{i,j_min}, формируем вектор V_min из элементов D_{i,j_min} (см. рисунок 1).
4. Находим максимальный элемент в векторе V_min (позиция этого элемента будет соответствовать bsf_pos, а значение – bsf_dist (см. рисунок 2).

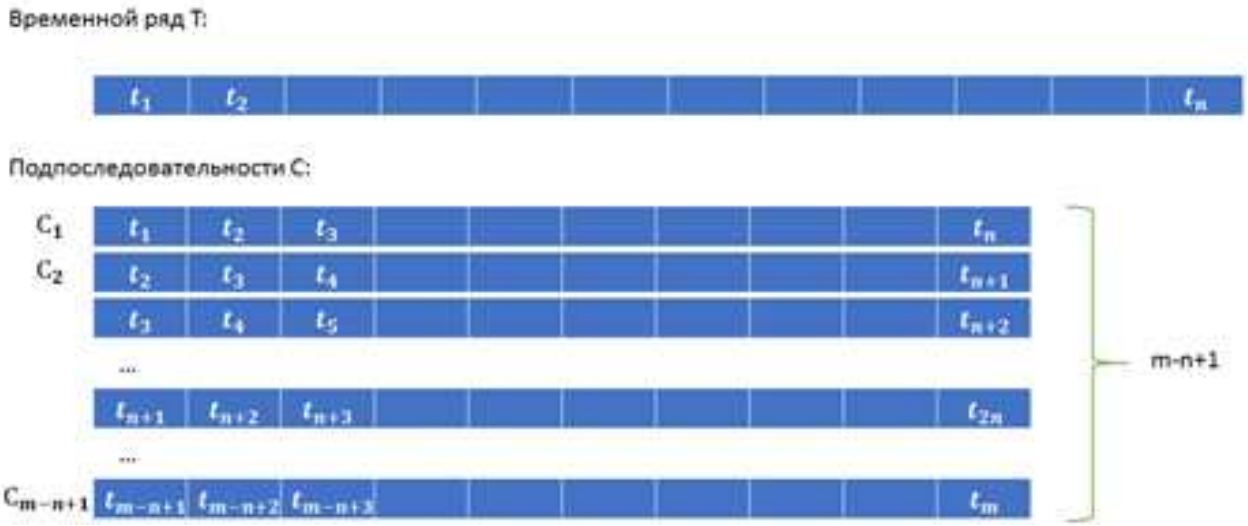

Рисунок 1 — Хранение данных в памяти

Модульная структура показана на рисунке 3. Программа состоит из модуля DistanceMatrix, в котором происходит подсчет матрицы расстояний и нахождение минимального элемента ряда матрицы расстояний, модуля Utils, который содержит вспомогательные методы для рассчета евклидова расстояния и модуля DiscordsRun, который содержит в себе основную логику параллельного наивного алгоритма поиска диссонансов. Файловая структура показана на рисунке 4. Программа состоит из следующих файлов:

∞ – фиктивное расстояние (между подпоследовательностью и самой собой и между подпоследовательностью и *self – match* подпоследовательностями)

Рисунок 2 — Нахождение диссонансов по матрице расстояний

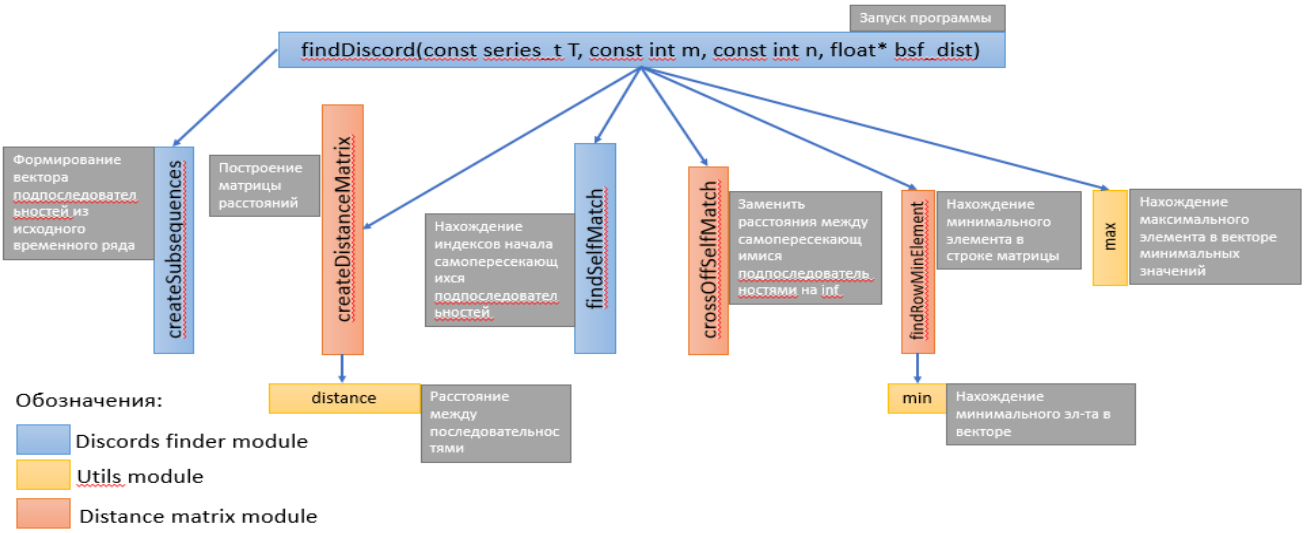

Рисунок 3 — Модульная структура

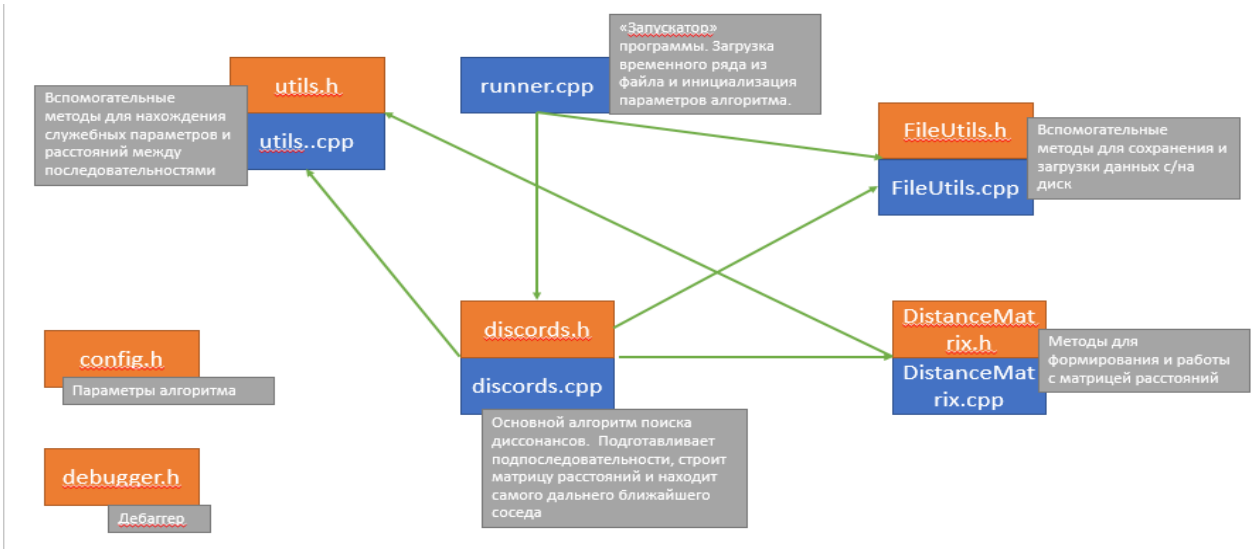

Рисунок 4 — Файловая структура

Таблица 1 — Зависимость объема используемой памяти от размера входных данных

Длина ряда	Размер элемента	Длина подпоследовательности	К-во подпоследовательностей	Эл-тов в м-це расстояний	Эл-тов в м-це подпоследовательностей	Размер памяти, Гб
1000000	4	128	999873	999746016129	127983744	3999, 50
500000	4	1024	498977	248978046529	510952448	997, 96
45000	4	1024	43977	1933976529	45032448	7, 92
45000	4	128	44873	2013586129	5743744	8, 08
43200	4	7	43194	1865721636	302358	7, 46
175000	4	128	174873	30580566129	22383744	122, 41
175000	4	1024	173977	30267996529	178152448	121, 78
90000	4	1024	88977	7916906529	91112448	32, 03
90000	4	128	89873	8077156129	11503744	32, 35
65000	4	1024	63977	4093056529	65512448	16, 63
65000	4	128	64873	4208506129	8303744	16, 87

utils.h и utils.cpp соответствуют модулю Utils, discords.h и discords.cpp соответствуют модулю DiscordsRun, DistanceMatrix.h и DistanceMatrix.cpp соответствуют модулю DistanceMatrix, config.h содержит основные настройки и константы.

Из-за перечисленных в начале подраздела особенностей алгоритма, он имеет следующие ограничения по объему обрабатываемых данных: Размер временного ряда $\sim$ 50 000 элементов типа float при объеме памяти ускорителя 8gb. Более подробно зависимость объема используемой памяти от размера входных данных показана в таблице 1. Также из таблицы видно, что объем используемой алгоритмом памяти мало зависит от размера подпоследовательностей, т.к. большую часть памяти занимает матрица расстояний, размер которой зависит от количество подпоследовательностей во временном ряде. Также следует сделать вывод, что данный алгоритм подходит только для относительно небольших временных рядов, однако для них работает очень быстро. При двукратном увеличении объема доступной памяти, размер обрабатываемого ряда может быть увеличен только в 1.4 раза.

1.3. Параллельный HOTSAX алгоритм

Параллельная реализация HOTSAX алгоритма Э. Кеоха (E. Keogh) выгодно отличается от наивного параллельного алгоритма на основе матрицы расстояний между подпоследовательностями экономным использованием памяти и линейной вычислительной сложностью ($O(m) + O(m) \equiv O(m), m \gg n$), однако HOTSAX алгоритм сложнее распараллелить и векторизовать, по сравнению с наивным алгоритмом. Преимуществом данного алгоритма является также то, что необходим всего один интуитивно понятный параметр n - длина подпоследовательности-диссонанса.

Параллельная реализация HOTSAX алгоритма позволяет искать диссонансы в больших временых рядах, содержащих миллионы элементов.

Входными параметрами алгоритма являются:

1. T – временной ряд ($T = \{t_i\}$, $1 \leq i \leq m$);
2. m – длина ряда;
3. n – длина подпоследовательности;
4. C – множество подпоследовательностей;
5. w – длина слова (в SAX аппроксимации подпоследовательностей), $1 \leq w \leq n, n??????w = 0$;
6. A – мощность алфавита для SAX представления подпоследовательностей.

Заранее рассчитывается *Lookuptable* (LT) — таблица точек разделения в SAX, и в алгоритме используется уже посчитанная готовая таблица. В статье [1] Кеох (Keogh) дает рекомендации по подбору параметров w и A. Так, оптимальная мощность алфавита $3 \leq A \leq 5$, подходит для большинства реальных данных. Кеох] (Keogh) рекомендует использовать $A = 3$. Оптимальная длина слов w выбирается следующим образом: для слабо меняющихся во времени последовательностей лучше подходит малое значение w, а для имеющих сложную структуру временных рядов – большое значение w. Как показано в статье [2] ускорение последовательной версии HOTSAX не зависит критически от параметра w: значение w можно изменять в диапазоне от 60% до 150% с уменьшением скорости ускорения на 12%.

Алгоритм состоит из следующих шагов:

1. Подготовка (выбор эвристики) — подбор порядка подачи подпоследовательностей, при котором возможно быстро отбрасывать неподходящие подпоследовательности.
2. Поиск диссонансов — перебор упорядоченных подпоследовательностей временного ряда с поиском наибольшего расстояния до ближайшего соседа, с постепенным обновлением best_so_far_dist и «быстрым» выходом из итераций цикла.

Этап подготовки можно разбить на несколько подшагов:

1. Z-нормализация (Normalization to Zero Mean and Unit of Energy) подпоследовательностей C_i временного ряда T. Каждый из элементов матрицы подпоследовательностей C пересчитывается по формуле 3. 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. рисунок 5):
$$C_{z\_norm}(i,j) = \frac{C(i,j) - \mu}{\sigma}, \tag{3}$$

2. Кусочно-агрегирующая аппроксимация – PAA-представление (см. рисунок 6). PAA аппроксимирует временной ряд X длины n ($X = \{x_1, ..., x_n\}$) в вектор $\overline{X} = \{\overline{x_1}, ..., \overline{x_M}\}$, где $M \leq n$. Каждый $C_{PAA}(i,j)$ вычисляется по формуле 4:
$$C_{PAA}(i,j) = \frac{M}{n} \sum_{k=\frac{n}{M}(j-1)+1}^{\frac{n}{M}j} C(i,k) \tag{4}$$

3. Кодирование с помощью lookup table (аппроксимация с помощью символьной агрегации (см. рисунок 7)).

4. Подсчет частот, нахождение мин. значения (см. рисунок 8)).

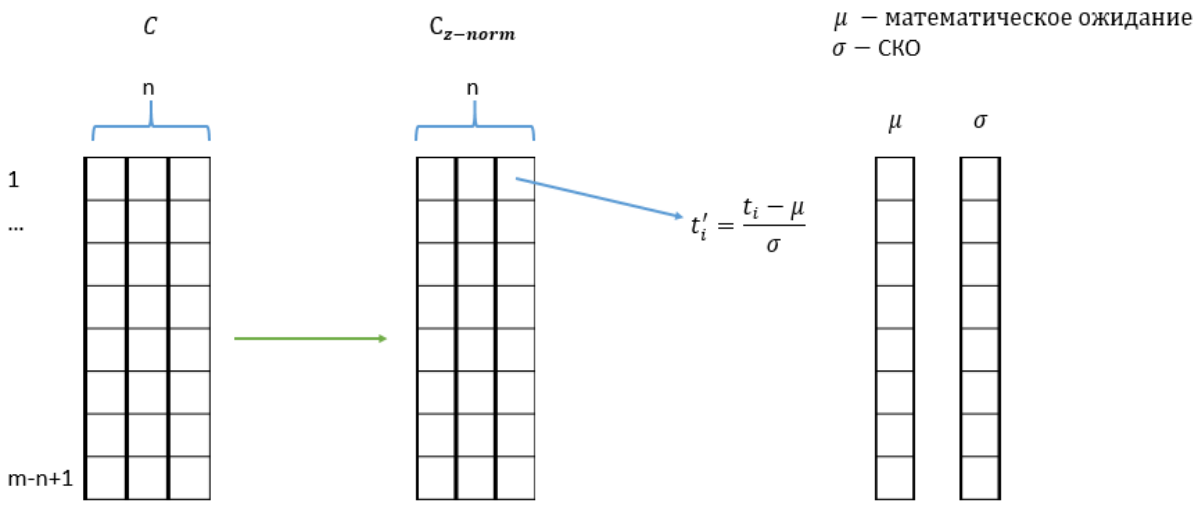

Рисунок 5 — Z-нормализация подпоследовательностей C_i временного ряда

После этапа подготовки перемещаем подпоследовательности, соответсвующие индексам из массива indexMinVal в начало матрицы подпоследовательностей. Поиск диссонансов происходит следующим образом: в цикле по всем подпоследовательностям находятся расстояния до ближайшего соседа обрабатываемой подпоследовательности, затем находится максимальное расстояние до ближайшего соседа (см. рисунок 9)).

В памяти хранятся следующие структуры данных:

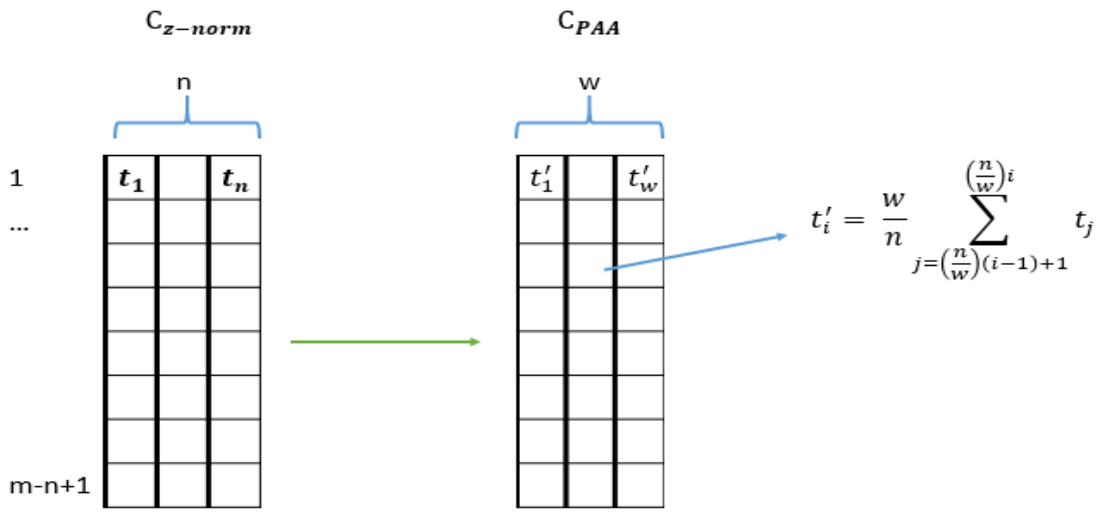

Рисунок 6 — Аппроксимация с помощью кусочной агрегации (PAA)

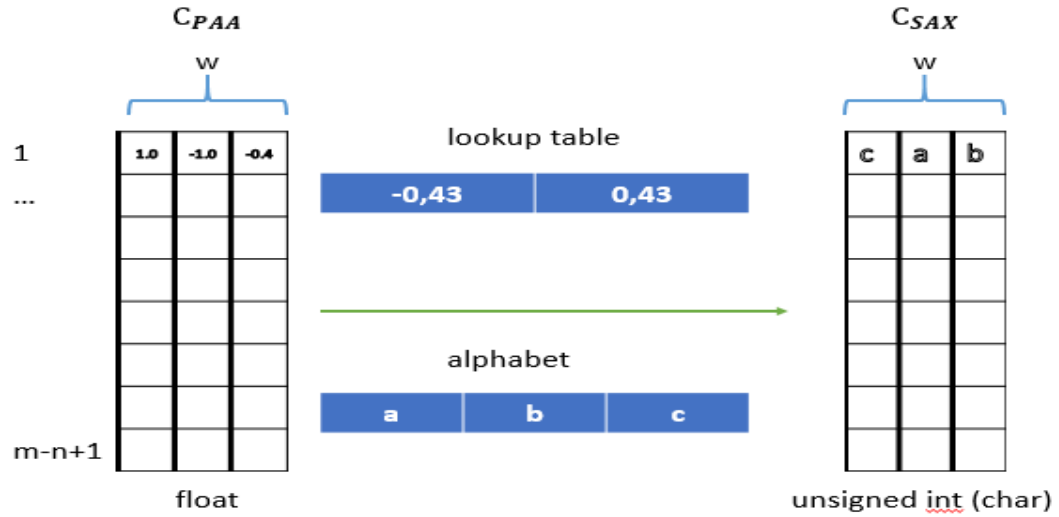

Рисунок 7 — Аппроксимация с помощью символьной агрегации)

1. временной ряд T в виде вектора размера $m \times 1$ (см. рисунок 1);
2. матрица подпоследовательностей размера $m - n + 1 \times n$ (см. рисунок 1);
3. векторы μ и σ размера $m - n + 1 \times 1$, которые используют при z-нормализации подпоследовательностей временного ряда;
4. матрица подпоследовательностей размера $m-n+1 \times w$ после PAA-апроксимации;
5. матрица символьных представлений подпоследовательностей размера $m-n+1 \times w$ после SAX-апроксимации.
6. вектор частот $frequences$ для символьных представлений подпоследовательностей размера $m - n + 1 \times 1$, вектор минимальных значений частот $minValue$ размера $m - n + 1 \times 1$ (все элементы инициализируются значением ∞, затем элементы,

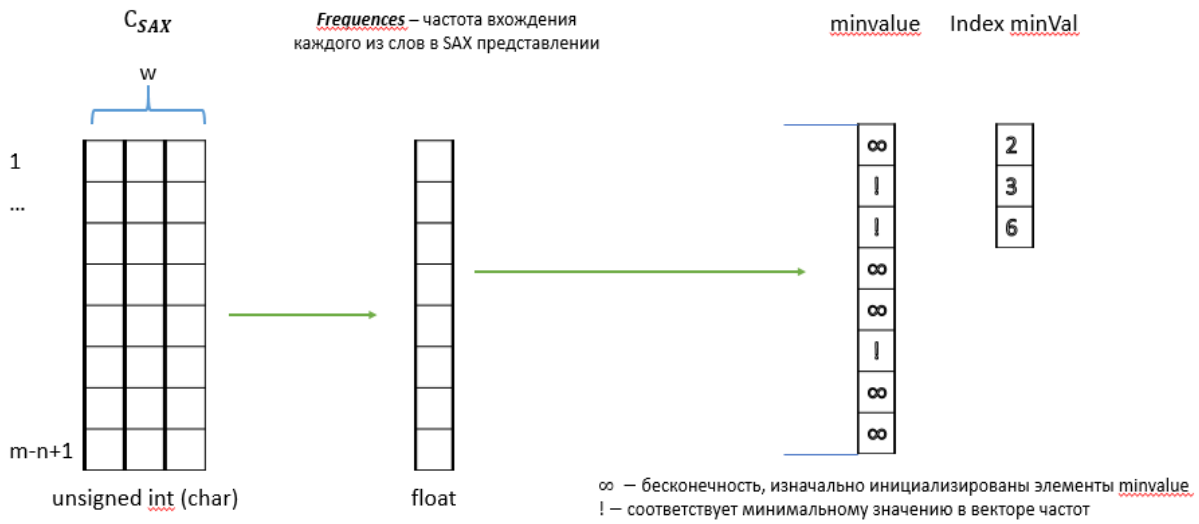

Рисунок 8 — Подсчет частот, нахождение мин. значения

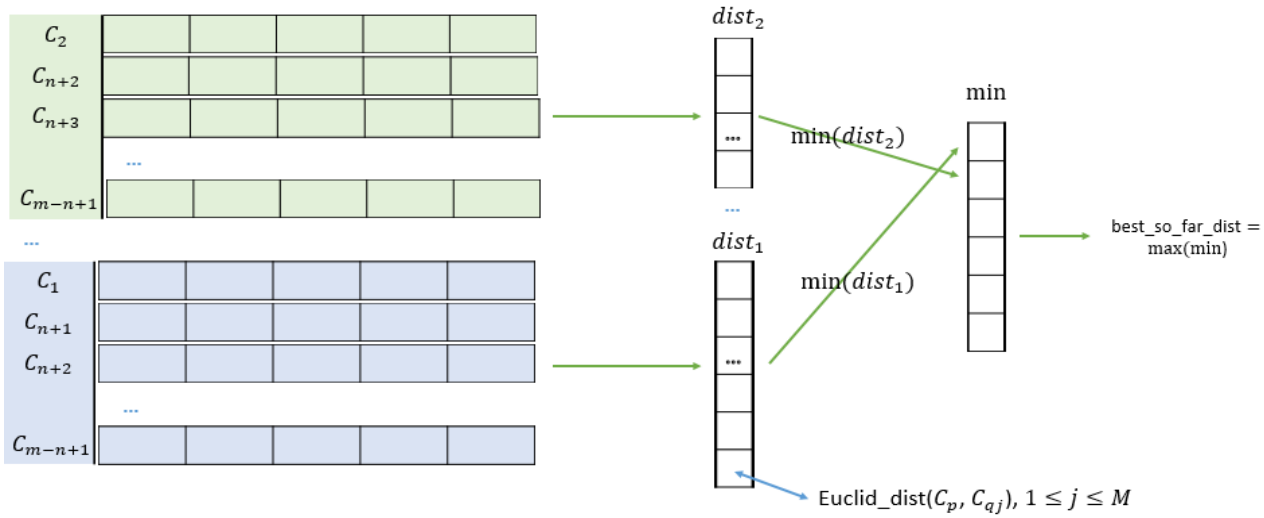

Рисунок 9 — Поиск диссонансов с учетом подобранных эвристик

соответствующие индексам sax-подпоследовательностей с наименьшей частотой вхождений заменяются на !) и вектор индексов элементов с наименьшим количеством вхождений indexMinVal.

Модульная структура показана на рисунке 10. Программа состоит из модуля SAX, в котором происходит кодирование подпоследовательностей в символьное представление, модуля PAA, в котором происходит кусочная апроксимация исходных подпоследовательностей и уменьшение их размера, модуля Utils, который содержит вспомогательные методы для рассчета евклидова расстояния и модуля DiscordsRun, который содержит в себе основную логику параллельного HOTSAX алгоритма поиска диссонансов. Файловая структура показана на рисунке 11. Программа состоит из следующих файлов: utils.h и utils.cpp соответствуют модулю Utils, discords.h и discords.cpp соответствуют модулю DiscordsRun, SAX.h и SAX.cpp соответствуют модулю SAX, PAA.h и PAA.cpp

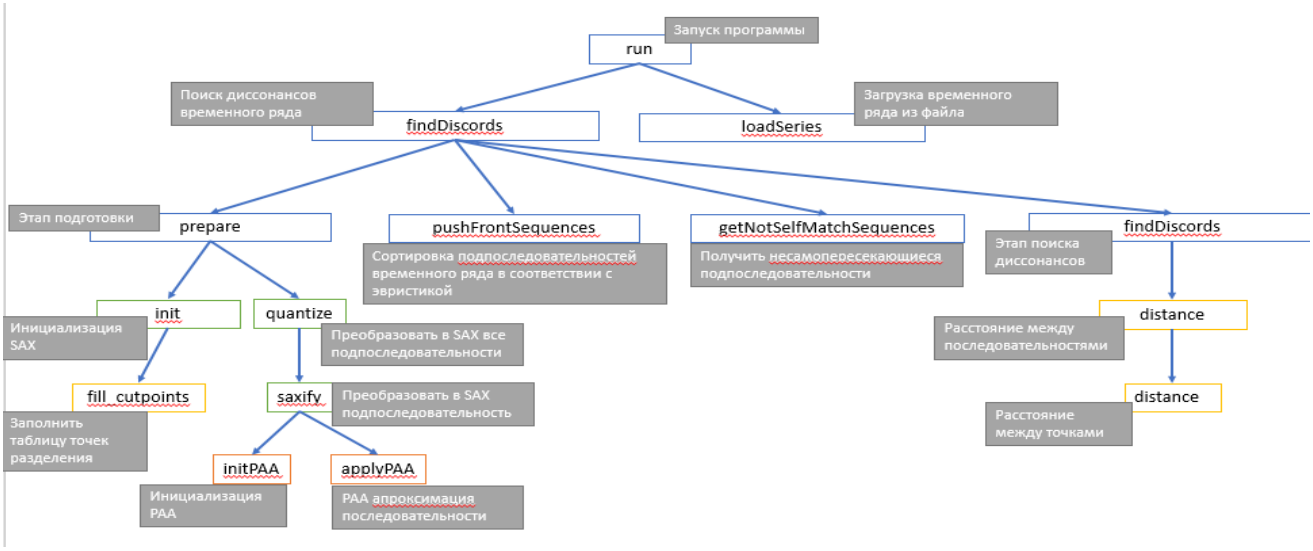

Рисунок 10 — Модульная структура

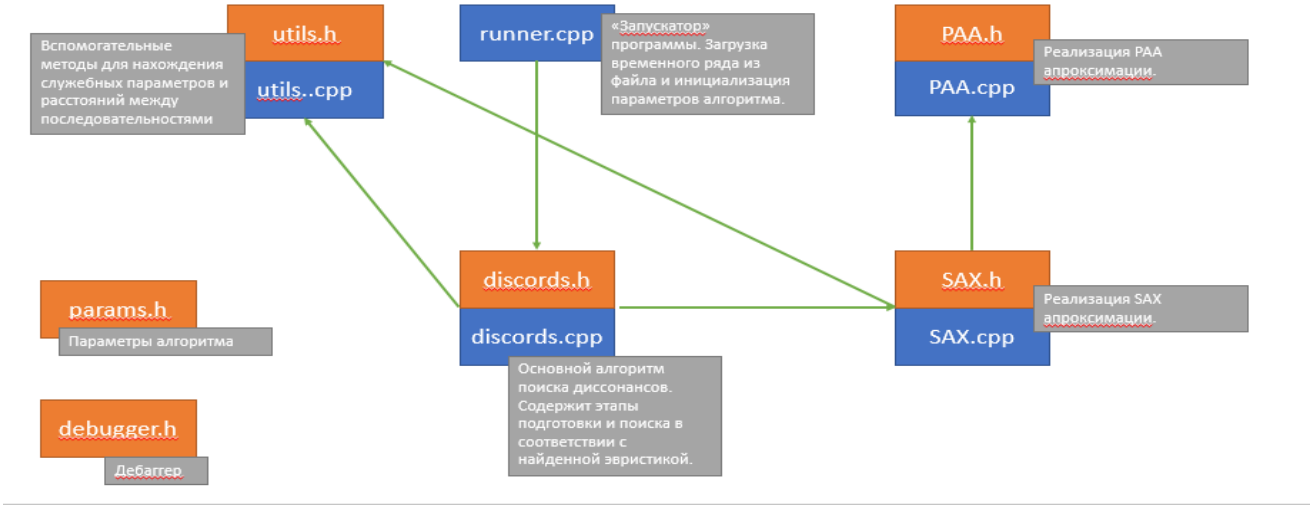

Рисунок 11 — Файловая структура

соответствуют модулю PAA, params.h содержит основные настройки и константы.

1.4. Выводы по разделу

В данном разделе были спроектированы параллельные алгоритмы поиска самой непохожей подпоследовательности временного ряда, основанные на последовательных алгоритмах предложенных Э. Кеохом (E. Keogh) для аппаратной платформы на основе сопроцессора Intel Xeon Phi, а также способы векторизации этих алгоритмов.

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.