图形处理器上的高性能时间序列异常发现

米哈伊尔·***L. Zymbler***[1]，亚娜·***A. Kraeva***[1]

© 作者 2023 年。

arXiv:2304.01660v1 [cs.DC] 4 Apr 2023

目前，时间序列中的子序列异常发现仍是最热门的研究问题之一。子序列异常是指一系列时间点整体上呈现异常，尽管其中每个点不一定都是离群点。在众多子序列异常发现方法中，discord 概念被认为是最优秀的方法之一。直观而言，时间序列 discord 是指给定长度的一个子序列，它与其非重叠最近邻的距离最大。近期提出的 MERLIN 算法能够发现指定范围内所有可能长度的时间序列异常片段，从而无需设置任何参数即可发现时间序列中的异常片段。然而，MERLIN 是串行算法，并行化有望提升异常子序列发现的性能。本文提出一种面向图形处理器的新型并行化方案，称为 PALMAD，即“基于 MERLIN 的任意长度并行异常发现”。与其串行前身算法不同，PALMAD 采用我们推导出的递推公式以避免冗余计算，并使用先进的数据结构来高效实现并行处理。对真实世界和合成时间序列进行的实验评估表明，我们的算法优于并行同类方法。我们还运用 PALMAD 在真实世界时间序列中发现异常，并采用所提出的异常子序列热力图技术展示结果。

Keywords: time series, anomaly detection, discord, MERLIN, DRAG, parallel algorithm, GPU, CUDA.

引言

过去几十年来，时间序列数据广泛存在于人类活动的各个领域：工业、医疗保健、科学、社会生活等。目前，在时间序列中发现异常（或称离群值）仍是最热门的研究问题之一。在时间序列中，点异常和子序列异常都可以作为检测目标 [4]。前者指在某个特定时刻，与时间序列中的其他值或其相邻点相比出现偏差的数据点。后者指一段连续的时间点，其整体行为异常，尽管其中每个观测值单独来看未必是点异常。子序列异常检测更具挑战性，因为除了其他因素外，还需要考虑子序列的长度 [4]。

在用于时间序列子序列异常检测的各种分析方法和基于神经网络的方法中， [4, 10], discord 概念 [31]被认为是最佳方法之一 [7, 8]。时间序列 discord 直观上定义为：与其非重叠最近邻之间的距离最大的子序列。对于终端用户而言，异常子序列颇具吸引力，因为它们只需要指定一个参数，即子序列长度。然而，异常子序列的应用受到对这一单一用户选择的敏感性的限制。一个直接的解决方案是发现所有可能长度的异常子序列，然后根据某种度量选择最佳异常子序列，但这在计算上似乎难以承受。

[1]俄罗斯联邦车里雅宾斯克南乌拉尔国立大学

尽管如此，最近提出的 MERLIN 算法 [36] 能够在指定范围内高效且精确地发现每个可能长度的异常子序列，在准确性和性能方面领先于竞争方法。因此，MERLIN 使终端用户无需设置上述唯一参数即可发现时间序列中的异常子序列。MERLIN 反复调用 DRAG 算法 [51] ，该算法能够发现给定长度且与其最近邻的距离至少为 r 的异常子序列，并自适应地选择参数 r。然而，多次调用上述子算法会导致部分重复计算；如果将这些计算同时执行，则可以提升 MERLIN 的性能。此外，MERLIN 算法是串行的，通过并行化来提高异常子序列发现的性能颇具吸引力。

在本研究中，我们延续此前利用并行架构加速各类时间序列挖掘任务和数据库内时间序列分析的研究 [29, 58–63, 65]，探讨如何将 MERLIN 算法并行化，以便在 GPU 上发现长度任意的异常片段。本文的贡献可概括如下。我们全面回顾了与基于异例的方法、时间序列异常发现以及这些方法在不同硬件平台上的并行化相关的研究。随后，我们基于 MERLIN [36], 提出并行算法 PALMAD（基于 MERLIN 的任意长度并行异常发现），用于在图形处理器上发现长度任意的异常片段。PALMAD 使用我们推导出的递推公式来避免冗余计算，并采用先进的数据结构来高效实现并行处理。此外，通过对真实世界和合成时间序列进行广泛的实验评估，我们证明该算法优于现有同类方法。最后，我们将 PALMAD 应用于真实世界时间序列的异常发现，并使用所提出的异常子序列热力图技术展示结果。

本文其余部分安排如下：第 1 节讨论相关工作。第 2 节介绍符号表示和形式化定义，并简要说明原始串行算法。第 3 节介绍所提出的并行算法，用于在图形处理器上发现时间序列异常子序列。第 4 节给出算法的实验评估结果并进行讨论。第 5 节介绍一个在真实世界时间序列中进行异常片段发现的案例研究。最后，在结论部分，我们总结所得结果，并提出未来研究方向。

1. 相关工作

时间序列异常片段这一概念由 Keogh 等人 [31], 提出， [7, 8]目前被认为是发现时间序列异常的最佳分析方法之一。直观而言，时间序列异常片段是指时间序列中与其非重叠最近邻距离最远的子序列。异常子序列对最终用户颇具吸引力，因为只需指定一个参数，即子序列长度。下文将介绍基于异常片段的异常发现研究，其中包括探讨上述方法并行化的研究。

在 [31], Keogh 等人提出了 HOTSAX（启发式排序的时间序列符号聚合近似）算法，用于发现可完全存放在随机存取存储器中的时间序列中的异常子序列。HOTSAX 使用 SAX 技术 [32] 和欧几里得距离对时间序列进行编码。HOTSAX 遍历所有子序列对，计算它们之间的距离，并从各子序列与其最近邻之间的距离中找出最大值。该算法

使用前缀树 [15] 为子序列建立索引。遍历过程中，无需计算距离即可舍弃不具潜力的子序列。若某子序列与邻居之间的距离小于当前所有最近邻距离的最大值，则该子序列不具潜力。HOTSAX 利用一种特定的启发式方法，从而能够舍弃更多不具潜力的候选项。HOTSAX 的改进版本包括 *i*SAX [41] 和 HOT-*i*SAX [6] （可索引 SAX）、WAT [5, 16] （以哈尔小波替代 SAX，并采用增强树）、HashDD [45] （以哈希表替代前缀树）、HDD-MBR [9]（采用 R树）、BitClusterDiscord [30] （对子序列的位表示进行聚类），以及 HST（HOTSAX时间） [3] （通过预热过程，以及利用时间上相近的子序列之间的相似性，缩小异常子序列搜索空间）。

在 [40], Senin 等人 提出了基于 HOTSAX 的稀有规则异常（RRA）算法，用于发现变长异常子序列。RRA 处理的是经过 SAX 离散化并采用语法归纳过程的时间序列。由于语法规则中很少使用的符号不会重复出现，因此可能较为异常；相应地，异常子序列对应于长度各异的低频语法规则。考虑到子序列长度各不相同，算法通过分段聚合近似（PAA） [23] 缩短最长子序列，使子序列长度一致，从而计算它们之间的距离。尽管 RRA 算法相比 HOTSAX 更进一步，实现了无需参数的异常子序列发现，但与其前身一样，它仅适用于存储在随机存取存储器中的时间序列。

在 [51], Yankov、Keogh 等人提出了 DRAG（异常子序列范围感知收集）算法，用于发现存储在磁盘而非随机存取存储器中的时间序列里的异常子序列。DRAG 引入了范围异常子序列概念：这类异常子序列与其非重叠最近邻之间的距离至少为 r，而 r 是用户定义阈值。DRAG 算法分两个阶段执行，即候选项选择（收集潜在的范围异常子序列）和异常子序列精炼（剔除假阳性）；每个阶段都需要对磁盘上的时间序列进行一次线性扫描。在 [43],Son 通过采用哈希桶数据结构略微改进了 DRAG，从而加快候选选择阶段。DRAG 的作者提出了以下方法来选择参数 r。通过均匀采样，可以获得原始时间序列中能装入随机存取存储器的最长片段。接着，HOTSAX 算法在上述片段中发现一个异常子序列。最后，假定 r 阈值等于所发现的异常子序列与其最近邻之间的距离。

然而，在 DRAG 算法中，上述启发式方法并未给出正式的方式来选择参数 r，以确保异常子序列发现 [34] 的效率。理想情况下， r 的设置应略小于最终发现的异常片段与其最近邻之间的距离 [36]。此时，DRAG 的时间和空间复杂度为 $O(mn)$，其中 n 是时间序列的长度， m 是异常片段的长度。如果将 r 的值设得远小于上述距离，那么算法仍能找到异常片段，但时间和空间复杂度会更高，为 $O(n^2)$。最后，如果 r 大于上述距离，则找不到任何异常片段。此外，DRAG（与其前身 HOTSAX 一样）无法发现所有 异常子序列，具体来说：该算法只能发现指定长度的异常子序列，而无法发现指定范围内所有可能长度的异常子序列。对于后一种情况，在指定的异常子序列长度范围内循环运行 DRAG 算法的暴力方法并不可行，因为在此类循环的每次迭代中，我们都必须从头选择参数 r。

Keogh等人 最近提出的 MERLIN 算法 [36] 克服了上述 DRAG 的局限性。MERLIN 会反复调用 DRAG，并自适应地选择参数 r。实验表明，MERLIN 能够高效且精确地发现所有可能长度的异常子序列，在准确性和性能方面均优于竞争方法 [36]。作者还通过实验证明，按照 [7] 中的分类法，MERLIN 能够发现点异常、上下文异常和集体异常。此外，作者指出，尽管深度学习异常检测方法（例如 [2, 13, 20, 35]）近来迅速发展，但它们是否优于基于异例的方法并不显而易见，因为前者本质上需要设置许多关键参数，而后者与领域无关，只需设置一个直观的参数；MERLIN 甚至可以免去设置该参数。不过，MERLIN 算法仍是串行的，对其进行并行化（适用于各种硬件平台）可以提升异常子序列发现的性能。此外还要指出，在 MERLIN 中，多次调用 DRAG 会导致部分计算重复执行（例如，对指定长度范围内的子序列进行归一化）；若将这些调用同时执行，也能提升 MERLIN 的性能。

针对异常片段发现并行化问题的研究包括以下内容。在 [58, 65], Zymbler 等人提出了一种 HOTSAX 并行化方案，分别通过 OpenMP [44] 或 OpenACC [39] 技术，利用英特尔众核处理器或图形处理器发现异常子序列。该算法采用矩阵数据布局组织计算，以尽可能多地使用可向量化循环。与其前身类似，该算法区分以下几组子序列：包含出现频率最低的 SAX 词的子序列与其余子序列；对于任一子序列，则区分其 SAX 词与给定子序列的 SAX 词相匹配的子序列与其余子序列。在通过两个嵌套循环遍历所有子序列时，该算法会根据运行线程数和上述集合的基数，分别以不同方式并行化外层循环和内层循环。

在 [52] （ [51]的扩展版本）中，Yankov、Keogh 等人讨论了基于 MapReduce 范式的 DRAG 并行版本 [11], ，其核心思想如下：将输入时间序列均匀分配到 P 高性能集群节点上。每个节点使用相同的参数 r 在其分区中选取候选项 $\mathcal{C}_i$，从而得到局部候选集。随后，构建全局候选集 $\mathcal{C}$ ，其形式为 $\mathcal{C} = \cup_{i=1}^{P} \mathcal{C}_i$ ，并将其发送到每个集群节点。接下来，各节点以全局候选集 $\mathcal{C}$ 为输入，在自己的分区中细化候选项，并生成局部细化候选集 $\tilde{\mathcal{C}}_i$。最后，计算得到全局异常子序列集 $\mathcal{D}$ ，其形式为 $\mathcal{D} = \cap_{i=1}^{P} \tilde{\mathcal{C}}_i$。然而，在实验评估中，作者只是在最多八台计算机上模拟了上述方案，结果获得了接近线性的加速比。

在 [60], Zymbler 等人提出了一种针对配备英特尔众核处理器的高性能集群的 DRAG 并行化方案。对于集群节点，作者定义了矩阵数据结构，并通过 OpenMP 技术实现线程级并行 [44] ，而集群节点之间的通信则通过 MPI（消息传递接口）实现 [42]。与 DRAG 并行化方案不同，作者首先在每个集群节点上根据相同的参数 r 对局部候选集 $\mathcal{C}_i$ 进行筛选，得到 $\mathcal{C}_i$ 集，然后构建全局候选集 $\mathcal{C} = \cup_{i=1}^{P} \tilde{\mathcal{C}}_i$ ；其依据是，在选择阶段，如果某个候选项至少被一个集群节点剪枝，那么它就不是真正的异常片段。在实验中 [60], 作者表明，这种技术能够大幅缩减全局候选集，并提升整体

算法性能。该算法的性能也显著优于以下面向高性能集群、基于 DRAG 的并行异常子序列发现算法：DDD（分布式异常片段发现） [50] 和 PDD（并行异常片段发现） [18]。上述竞争算法远逊于该算法，因为它们需要在集群节点之间进行大量数据交换。然而，上述并行化方案仍无法高效地发现指定长度范围内所有可能长度的异常子序列。

在综述中，我们还应提及 Keogh 提出的矩阵剖面（以下简称 MP）概念等。 [53]对于给定的时间序列，MP 可以非正式地定义为一个时间序列，其中第 i 个元素表示原始时间序列中的第 i 个子序列与其非重叠最近邻之间的距离。MP 是多种时间序列模式发现相关问题解决方案的基础构件（语义模式 [21], 片段 [22], 链 [55],等）。根据上述定义，可以将前 k 个异常片段作为 MP 计算的副产品发现，因为它们是使 MP 中前 k 个最大值达到峰值的子序列。然而，MP 计算的时间复杂度很高，即 $O(n^2)$（其中 n 是时间序列长度） [53, 56], ，因此直接将 MP 用于异常子序列发现会导致性能低下，正如下文实验所评估的那样。串行 SCRIMP 算法 [53] 不如 MERLIN [36]。面向图形处理器和高性能集群的并行 MP 算法 GPU-STAMP [53] 和 MP-HPC [38], 也不如文献 [60] 提出的、适用于配备英特尔众核处理器的高性能集群的并行异常子序列发现算法。

在 [46], Thuy 等人提出了 K-距离异常片段的概念，即与其非重叠 K 最近邻的距离之和最大的子序列。这种方法旨在解决所谓的“孪生异常”问题 [48] ： 当异常片段在时间序列中出现多次时，discord无法发现异常（罕见）子序列。该方法是对 J 距离异常片段 [19]概念的改进，后者采用子序列与其第 k 个非重叠最近邻之间的距离。作者还提出了 KBF GPU（K 距离异常片段暴力搜索）算法，用于在图形处理器上加速 K-距离异常片段发现。KBF GPU 通过两个嵌套循环遍历给定时间序列中的所有子序列，其中内层循环采用并行化方式，并针对距离之和的计算进行了调整。然而，在实验中，作者仅将该算法与串行 HOTSAX [31], 进行了比较，而后者不出所料地明显逊于 KBF GPU。

在 [54], Zhu 等人中提出了一种利用图形处理器加速异常子序列发现的并行算法。作者采用归一化欧几里得距离，并通过 [34] 中提出的技术，利用皮尔逊相关系数高效地计算该距离。为实现高性能的异常子序列发现，该算法采用了两种计算模式。第一种模式规定了以下两个步骤。首先，计算异常候选子序列与时间序列中所有不与该候选项重叠的其他子序列之间的最小距离。然后，找出使所有候选项中的最大距离达到最大值的候选项。第二种模式规定：当候选项与某个子序列之间的距离小于当前最优距离时，提前终止上述模式中的计算。在这种情况下，该候选项和该子序列显然都不是异常子序列，因此无需再计算该候选项与其他不重叠子序列之间的距离。在实验中， [54], 所提出的算法

超过了 SCAMP [57]，后者目前是计算矩阵剖面的最快并行算法。然而，所提出的计算模式将结果限制为时间序列中单个（尽管是最重要的）异常子序列，而上述算法基于范围异常片段概念，能够发现前 k 个异常子序列，其中参数 k 由该领域的专家预先指定。

综上所述，当前，基于范围异常片段概念的 MERLIN 算法 [36] 是发现时间序列异常的最有前景的方法之一。[51] 此外，MERLIN 具有分析性且不依赖特定假设，其表现至少可与深度学习方法相媲美。然而，MERLIN 的并行化可以提升异常片段发现的性能。并行化是一个热门议题，因为据我们所知，尚无研究使用图形处理器或其他并行硬件架构，加速发现所有可能长度的异常子序列。

2. 预备知识

在第 2.1 节和第 2.2 节详细介绍所提出的异常子序列发现并行算法之前，我们先根据 [36, 51] 介绍基本的符号表示和形式化定义，并分别概述本文所依据的原始串行算法 MERLIN 和 DRAG。

2.1. 符号与定义

一个时间序列是按时间顺序排列的实数序列：

$$T = \{t_i\}_{i=1}^{n}, \quad t_i \in \mathbb{R}. \tag{1}$$

时间序列的长度 n 记为 $|T|$。下文中，我们假设时间序列 T 能够装入主存。

A 子序列 $T_{i,\ m}$ 时间序列 T 中从第i个位置开始的 m 个连续元素的子集：

$$T_{i,m} = \{t_k\}_{k=i}^{i+m-1}, \quad 1 \le i \le n-m+1, \quad 3 \le m \ll n. \tag{2}$$

我们将 T 中所有长度为 m 的子序列集合记为 S_T^m。令 N 表示 S_T^m 中子序列的数量，即 $N = |S_T^m| = n - m + 1$。

任意两个长度为 m 的子序列的**距离函数** 是一个非负且对称的函数 Dist : $\mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$.

给定一个时间序列 T 及其两个子序列 $T_{i,\ m}$ 和 $T_{j,\ m}$，如果它们之间的距离 Dist($T_{i,\ m}$, $T_{j,\ m}$) 为 $|i-j| \ge m$，则称它们彼此为 **非自身匹配** 。我们将子序列 $C \in S_T^m$ 的一个非自身匹配记为 M_C。

给定时间序列 T，若子序列 $D \in S_T^m$ 与其最近非自身匹配之间的距离最大，则称其为**异常片段** 。形式上，异常片段 D 满足以下条件：

$$\forall C \in S_T^m \quad \min\big(\mathrm{Dist}(D,\, M_D)\big) > \min\big(\mathrm{Dist}(C,\, M_C)\big). \tag{3}$$

上述定义从 top-1 推广到 top-k 异常片段如下：如果 $D \in S_T^m$ 到其第 k 个最近非自身匹配的距离最大，则称 $D \in S_T^m$ 为第 k 个异常片段。

给定正实数 r，与其最近非自身匹配的距离至少为 r 的异常片段称为 范围异常子序列。也就是说，关于参数 r 的范围异常子序列 D 满足以下条件：$\min\big(\text{Dist}(D, M_D)\big) \geq r$。

MERLIN [36]和 DRAG [51] 算法处理的是时间序列的子序列，这些子序列此前已进行 z归一化，使其均值为零，标准差为一。这里，子序列 $X = \{x\}_{i=1}^{m} \in S_T^m$的 z归一化 $\hat{}$ 定义为一个子序列 $X = \{\hat{}x\}_{i=1}^{m}$，其中

$$\hat{x}_i = \frac{x_i - \mu_X}{\sigma_X}, \quad \mu_X = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_X^2 = \frac{1}{m}\sum_{i=1}^{m} x_i^2 - \mu^2. \tag{4}$$

MERLIN 和 DRAG 算法均采用欧几里得度量作为 Dist(·, ·) 函数，用于测量子序列之间的距离，其定义如下。设 X,$Y \in S_T^m$，则子序列之间的欧几里得距离计算如下：

$$\text{ED}(X,\ Y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}. \tag{5}$$

在我们的研究中，出于尽可能提高 discord发现性能的考虑，我们采用欧几里得度量的平方作为距离函数。为简化起见，我们将两个经过 z 标准化的子序列之间的欧几里得距离记为 ED_{norm}：$\text{ED}_{\text{norm}}(X, Y)$ =ED($\hat{}X$, $\hat{}Y$)。为了计算 $\text{ED}^2_{\text{norm}}$，我们进一步采用了 [34] 中提出的以下技术，该技术比公式 5 能够实现更快的计算：

$$\text{ED}^2_{\text{norm}}(X,\ Y) = 2m\left(1 - \frac{X\cdot Y - m\cdot\mu_X\cdot\mu_Y}{m\cdot\sigma_X\cdot\sigma_Y}\right), \tag{6}$$

其中，$X\cdot Y$表示向量 $X, Y \in \mathbb{R}^m$ 的标量积。

2.2. MERLIN 和 DRAG 算法

算法 1 展示了 MERLIN 的伪代码 [36]（在指定长度范围内，针对每个长度发现前 k 个异常片段，而非所有异常片段）。下文中，设有一个长度为 n 的时间序列 T，我们要找出其异常片段集合 $\mathcal{D}$，这些异常片段的长度范围为 最小长度..最大长度（其中最小长度 $\leq$最大长度 $\ll n$），使得 $\mathcal{D} = \cup_{m=minL}^{maxL} D_m$，其中 D_m 表示长度为 m 的异常片段的子集。$d \in \mathcal{D}$ 异常片段的最近邻距离记为 d.最近邻距离。

该算法规定了选择参数 r 的以下过程。异常子序列会按顺序被发现，长度从指定 discord范围的最小值开始，逐步增加至最大值。在每一步中，MERLIN 会计算已发现异常子序列到其最近邻的最后五个距离的算术平均值 μ 和标准差 σ，然后调用 DRAG 算法，并向其传入参数 $r = \mu - 2\sigma$。如果 DRAG 尚未发现异常片段，则会从 r 中减去 σ，直到 DRAG 成功停止（即发现一个异常片段）。对于前五种异常片段长度，参数 r 的设置如下。对于长度小于 $minL$ 的异常片段，参数设置为 $r = 2\sqrt{minL}$，因为它是任意两个$minL$长度子序列之间可能的最大距离；随后将 r 减半，直到使用这样的参数调用 DRAG 后获得成功。为了获得接下来的四种异常片段长度，该算法采用异常片段到其

算法 1 MERLIN (IN T, $minL$, $maxL$, $topK$; OUT $\mathcal{D}$)

$\mathcal{D} \leftarrow \varnothing$; $r \leftarrow 2\sqrt{minL}$; 最近邻距离$_{minL} \leftarrow -\infty$
当 最近邻距离$_{minL} < 0$ **和** $|D_{minL}| < topK$ **时，做以下操作。**
    $D_{minL} \leftarrow \mathrm{DRAG}(T, minL, r)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_{minL}$; 最近邻距离$_{minL} \leftarrow \min_{d \in D_{minL}} d$.最近邻距离
    $r \leftarrow 0.5 \cdot r$
对于 $i \leftarrow minL + 1$ **到** $minL + 4$ **执行**
    最近邻距离$_i \leftarrow -\infty$
    当 最近邻距离$_i < 0$ **且** $|D_i| < topK$ **时**
        $r \leftarrow 0.99 \cdot nnDist_{i-1}$
        $D_i \leftarrow \mathrm{DRAG}(T, i, r)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_i$; 最近邻距离$_i \leftarrow$ 最小$_{d \in D_i}$ d.最近邻距离
        $r \leftarrow 0.99 \cdot r$
对于 $i \leftarrow minL + 5$ **到** $maxL$ **执行**
    $\mu \leftarrow$ 均值($\{$最近邻距离$_k\}_{k=i-1}^{i-5}$); $\sigma \leftarrow$ 标准差($\{$最近邻距离$_k\}_{k=i-1}^{i-5}$); $r \leftarrow \mu - 2\sigma$
    $D_i \leftarrow \mathrm{DRAG}(T, i, r)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_i$; 最近邻距离$_i \leftarrow$ 最小值$_{d \in D_i}$ d.最近邻距离
    当 $nnDist_i < 0$ **且** $|D_i| < topK$ **时**
        $D_i \leftarrow \mathrm{DRAG}(T, i, r)$; $\mathcal{D} \leftarrow \mathcal{D} \cup D_i$; 最近邻距离$_i \leftarrow$ 最小值$_{d \in D_i}$ d.最近邻距离
        $r \leftarrow r - \sigma$
返回 $\mathcal{D}$

上一步获得的最近邻的距离，再减去一个等于 1% 的小值。持续额外减去 1%，直到使用这样的参数进行 discord发现后获得成功。关于上述过程的详细说明，请参阅原始工作 [36]。

算法 2 DRAG (IN T, m, r; OUT $\mathcal{D}$)

PHASE 1. SELECT CANDIDATES

$\mathcal{C} \leftarrow \{T_{1,m}\}$
对于所有 $s \in S_T^m \setminus T_{1,m}$ **，执行**
    候选标记 $\leftarrow$ 真
    对于所有 $c \in \mathcal{C}$ **和** $c \in M_s$ **，执行**
        如果 $\mathrm{ED}(s, c) < r$ **，则**
            $\mathcal{C} \leftarrow \mathcal{C} \setminus c$
            候选标记 $\leftarrow$ FALSE
    如果 候选标记 **那么**
        $\mathcal{C} \leftarrow \mathcal{C} \cup s$
return $\mathcal{C}$

PHASE 2. REFINE DISCORDS

$\mathcal{D} \leftarrow \varnothing$; $\forall c \in \mathcal{C}$ c.最近邻距离 $\leftarrow +\infty$
对于所有 $s \in S_T^m$ **do**
    对于所有 $c \in \mathcal{C}$ **和** $c \in M_s$ **，其中** $s \neq c$ **，执行**
        $dist \leftarrow$ 提前终止编辑距离(s, c)
        如果 $dist < r$ **则**
            $\mathcal{C} \leftarrow \mathcal{C} \setminus c$
        else
            $\mathcal{D} \leftarrow \mathcal{D} \cup c$
            c.最近邻距离 $\leftarrow$ 最小值(c.最近邻距离，距离)
返回 $\mathcal{D}$

DRAG 算法 [51] （见算法 2）分为两个阶段，即候选项选择和异常子序列精炼，分别用于收集潜在的范围异常子序列并丢弃假阳性。在第一阶段，DRAG 扫描时间序列 T，并对每个子序列 $s \in S_T^m$验证候选集 $\mathcal{C}$ 中已有的每个候选项 c 成为异常子序列的可能性。如果某个候选项 c 未通过验证，则将其从该候选集中移除。最终，新的 T要么在很可能是异常子序列时被添加到候选项集合中，要么被剪枝。如果一名候

选项 c 未通过验证，则将其从该集合中移除。最终，新的 s 要么在很可能是异常子序列时被添加到候选项集合中，要么被剪枝。在第二阶段，算法首先将所有候选项与其最近邻之间的距离设为正无穷。然后，DRAG 扫描时间序列 T，计算每个子序列 $s \in S_T^m$ 与每个候选项 c之间的距离。在计算 ED(s, c)时，EarlyAbandonED 过程会在求和 $\sum_{k=1}^{m}(s_k - c_k)^2$ 达到 $k = \ell$时停止，因此 $1 \leq \ell \leq m$ 满足 $\sum_{k=1}^{\ell}(s_k - c_k)^2 \geq c.nnDist^2$。如果该距离小于 r，则该候选项为假阳性，并将其从 $\mathcal{C}$中永久移除。如果上述距离小于 $c.nnDist$ 的当前值（且仍大于 r，否则该候选项早已被移除），则更新当前的最近邻距离。上述过程的正确性已在原始工作 [51]中得到证明。

3. 使用图形处理器发现任意长度的异常子序列

目前，图形处理器（图形处理器） [25] 是最流行的众核硬件平台之一。图形处理器非常适合单指令多数据（单指令多数据）计算，其由对称的流式多处理器组成，而每个流式多处理器又由对称的统一计算设备架构核心构成。统一计算设备架构应用程序接口（应用程序接口）支持分配多个线程，使其对多份数据执行同一组指令。在统一计算设备架构中，所有线程构成一个网格，其中包含多个块。在一个块中，线程被划分为线程束，每个线程束是由32个线程组成的逻辑组。块中的线程并行运行，并通过共享内存相互通信。统一计算设备架构函数称为内核。在图形处理器上运行内核时，应用程序员需要指定网格中的块数以及每个块中的线程数。

下文第3.1节将介绍 PALMAD 的总体架构与数据结构。PALMAD 是一种并行算法，基于原始串行 MERLIN 算法 [36]，用于在图形处理器上发现指定长度范围内所有可能长度的异常子序列。PALMAD 使用 PD3（基于 DRAG 的并行 discord 发现） [28], ，即原始串行 DRAG 算法 [51]的并行版本。与原始串行算法类似，PD3 分两个阶段执行，每个阶段分别进行并行化。在第3.2节和第3.3节中，我们将分别讨论候选项选择阶段和异常子序列细化阶段的并行化。

3.1. 总体架构与数据结构

3.1.1. 避免冗余计算

基本上，我们的并行算法遵循其串行前身算法的计算方案（见算法 1）。PALMAD 通过反复调用 PD3（我们为图形处理器设计的串行 DRAG 算法的并行版本；见算法 2 以及算法 1 中的第 3、9、13 和 15 行）来完成计算。与原始算法不同，当子序列长度比上一步增加 1 时，PALMAD 可避免在迭代调用 DRAG 时进行冗余计算。实际上，要计算任意两个候选子序列之间的距离，我们需要部分重复与长度比上方候选项的自匹配子序列少 1 的子序列相关的类似计算。更形式化地说，对于任意$i,\ \ j$(1 $< i \leq n - m$和 $3 \leq m \ll n$)，在通过公式 6 计算 $\mathrm{ED}^2_{\mathrm{norm}}(T_{i,\ m},\ T_{j,\ m})$ 时，从

头开始，我们通过公式 4 部分重复计算均值 $\mu_{T_{i,\ m-1}}$ 和 $\mu_{T_{i,\ m-1}}$，以及标准差 $\sigma_{T_{i,\ m-1}}$ 和 $\sigma_{T_{i,\ m-1}}$。

为避免上述开销，我们采用向量 $\bar{\mu}$、$\bar{\sigma} \in \mathbb{R}^{n-minL+1}$，它们分别表示所有给定长度的时间序列子序列的均值和标准差。在这些向量中，先处理 $n-m+1$ 个元素，其中 m 是给定的子序列长度（$minL \leq m \leq maxL$），其余元素则不予处理。

对于长度为 $minL$ 的子序列，这些向量会在首次调用 PD3 之前，根据公式 4 计算一次（见算法 1 第 3 行）；对于 m 的其余取值，则会在之后每次调用 PD3 之前，根据以下递推公式更新向量 $\bar{\mu}$ 和 $\bar{\sigma}$（见算法 1 第 9、13 和 15 行）：

$$\mu_{T_{i,\ m+1}} = \frac{1}{m+1}\left(m\mu_{T_{i,\ m}} + t_{i+m}\right), \tag{7}$$

$$\sigma^2_{T_{i,\ m+1}} = \frac{m}{m+1}\left(\sigma^2_{T_{i,\ m}} + \frac{1}{m+1}\left(\mu_{T_{i,\ m}} - t_{i+m}\right)^2\right). \tag{8}$$

为避免文章正文因过多细节而显得冗长，关于公式7和公式8的引理及其证明见附录。

向量 $\bar{\mu}$ 和 $\bar{\sigma}$ 的初始计算与更新通过 CUDA 内核实现。我们构建一个包含 N 个线程的网格，其中每个块中的线程数是算法的参数，该参数设置为 GPU 线程束大小的整数倍。每个线程根据公式 7 和 8 计算向量 $\bar{\mu}$ 和$\bar{\sigma}$ 的元素。

3.1.2. 数据结构与分段

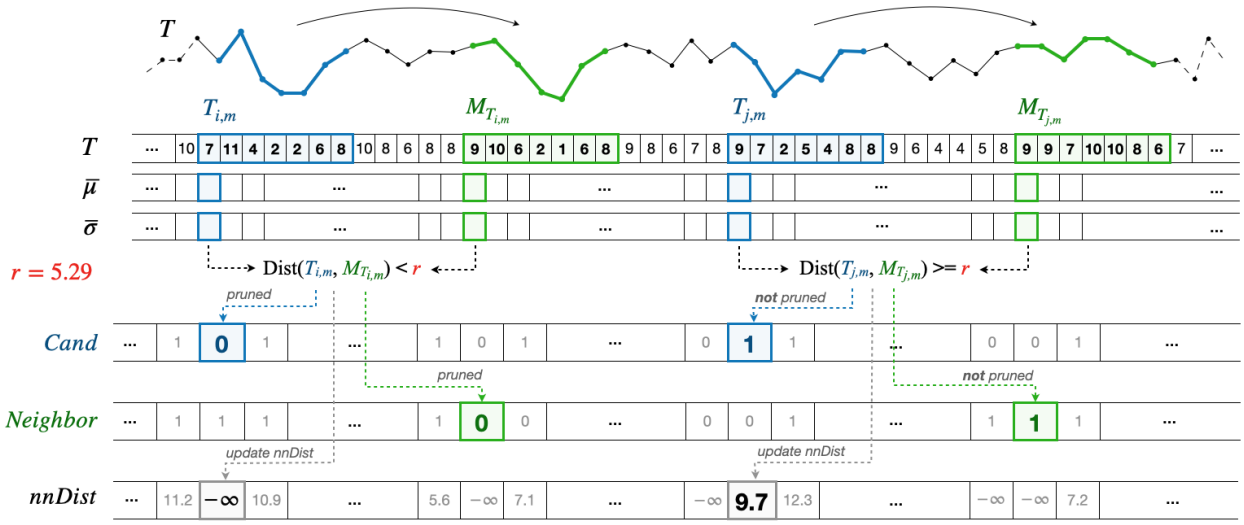


图 1. 数据结构

在图 1 中，我们展示了计算中涉及的基本数据结构。对于输入时间序列的一个子序列，实值向量最近邻距离 $\in \mathbb{R}^N$包含其到最近邻的距离：最近邻距离(i) $=\min\left(\text{编辑距离}^2_{\text{norm}}(T_{i,\ m},\ M_{T_{i,\ m}})\right)$。两个布尔值向量候选项，邻居 $\in \mathbb{B}^N$分别是子序列及其最近邻的位图：如果子序列候选项(i) $=$ 为 TRUE（或

邻居(i) = 为 TRUE，则子序列 $T_{i,\ m}$（或其最近邻）是异常片段；否则为 FALSE。这些位图初始化时均设为 TRUE。此外，我们对上述位图逐元素合取，以便在处理过程中排除更多候选项；其依据是一个显而易见的事实：不是异常片段的子序列，其最近邻也不可能是异常片段。

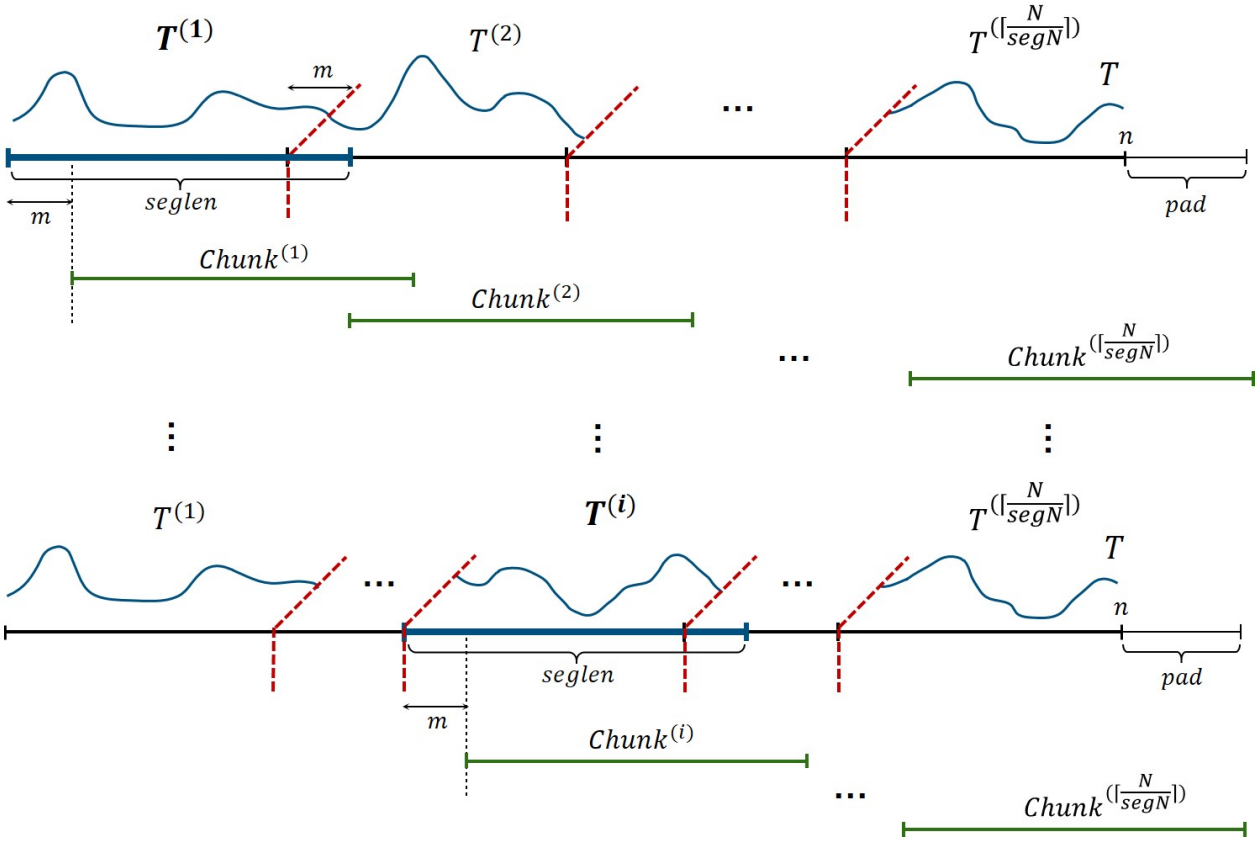


图 2. 数据分段

为了实现候选项选择和异常子序列精炼阶段，我们利用数据并行的概念，并按照图 2 所示对数据进行分段。时间序列被划分为长度相等的分段，每个分段由一个 GPU 线程块单独处理。在执行该阶段时，线程块以数据块为单位扫描子序列。数据块中的元素数量等于片段长度，第一个数据块从该分段中的第 m 个元素开始。这种技术避免了因候选项与数据块中的子序列重叠而进行冗余检查。

片段长度是算法的一个参数，应设置为 GPU 线程束大小的倍数。为了平衡块中线程的负载，我们要求时间序列中 m 长度子序列的数量，是分段中指定长度子序列数量的倍数。如果不满足这一条件，我们会在时间序列右侧填充值为正无穷的虚拟元素。设分段长度和分段中 m 长度子序列的数量分别为 $seglen$ 和 $segN$，则 $segN = seglen - m + 1$。设最右侧分段中的虚拟元素数量为

pad，则其定义如下：

$$pad = \begin{cases} m-1, & N \bmod segN = 0 \\ \lceil \dfrac{N}{segN} \rceil \cdot segN + 2(m-1) - n, & \text{otherwise} \end{cases}. \tag{9}$$

3.2. 候选选择的并行化

算法 3 PD3SELECT (IN T, m, r; OUT $\mathcal{C}$)

候选项 $\leftarrow$ `TRUE`; 邻居 $\leftarrow$ `TRUE`
对所有 $T^{(i)} \in T$ **执行** ▷并行（块）
  对所有 数据块$^{(j)} \in T^{(i)}$ **其中** $i \le j$ **执行** ▷并行（线程）
    如果 $i = j$ **则**
      QTrow $\leftarrow$ CALCDOTPRODUCTS（$T^{(i)}_{1,m}$, 数据块$^{(j)}$）
      继续
    QTrow $\leftarrow$ UPDATEDOTPRODUCTS(*QTrow* $T^{(i)}_{1,m}$, 数据块$^{(j)}$)
    QTcol $\leftarrow$ CALCDOTPRODUCTS(数据块$^{(j)}_{1,m}$, $T^{(i)}$)
    dist $\leftarrow$ CALCDIST(数据块$^{(j)}_{1,m}$ $T^{(i)}$, *QTcol* $\bar{\mu}, \bar{\sigma}$)
    如果 $dist < r$ **则**
      候选项（$i \cdot segN + tid$）$\leftarrow$ `FALSE`; 邻居（$j \cdot segN + 1$）$\leftarrow$ `FALSE`
    else
      最近邻距离（$j \cdot segN + 1$）$\leftarrow$ 最小(距离，最近邻距离（$j \cdot segN + 1$）)
    如果不是 $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN}$ 候选项（k）**则**
      中断
    对于所有数据块$^{(j)}_{k,m} \in S^m_{Chunk^{(j)}} \setminus$ 数据块$^{(j)}_{1,m}$ **do** ▷并行（线程）
      QTcol $\leftarrow$ UPDATEDOTPRODUCTS（*QTcol*, *QTrow*, 数据块$^{(j)}_{k,m}$ $T^{(i)}$）
      dist $\leftarrow$ CALCDIST（数据块$^{(j)}_{k,m}$ $T^{(i)}$, *QTcol* $\bar{\mu}, \bar{\sigma}$）
      如果 $dist < r$ **则**
        候选项($i \cdot segN + tid$) $\leftarrow$ 假; 邻居($j \cdot segN + k$) $\leftarrow$ 假
      else
        最近邻距离($j \cdot segN + 1$) $\leftarrow$ 最小值(*dist*，最近邻距离($j \cdot segN + 1$))
    如果不是 $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN}$ 候选项(k) **则**
      中断
$\mathcal{C} \leftarrow \{\{T_{i,m} \in S^m_T; \text{最近邻距离}(i)\} \mid 1 \le i \le n-m+1, \text{候选项}(i) = \text{TRUE}\}$
返回 $\mathcal{C}$

算法 3 展示了我们对候选选择阶段进行并行化的伪代码。相应的 CUDA 内核会构建一个由 $\lceil \frac{N}{segN} \rceil$ 个线程块组成的网格，每个线程块包含 $segN$ 个线程。线程块将片段子序列视为异常片段的局部候选项，并对位于分段右侧 且与候选项不重叠的子序列进行分块处理。子序列的处理方式如下：如果候选项与子序列之间的距离小于参数 r，则将候选项和子序列排除在后续处理之外，因为它们显然不是异常片段（位图中相应的标志设为 FALSE）。

如果所有局部候选项都被丢弃，该块会提前终止其所有线程。

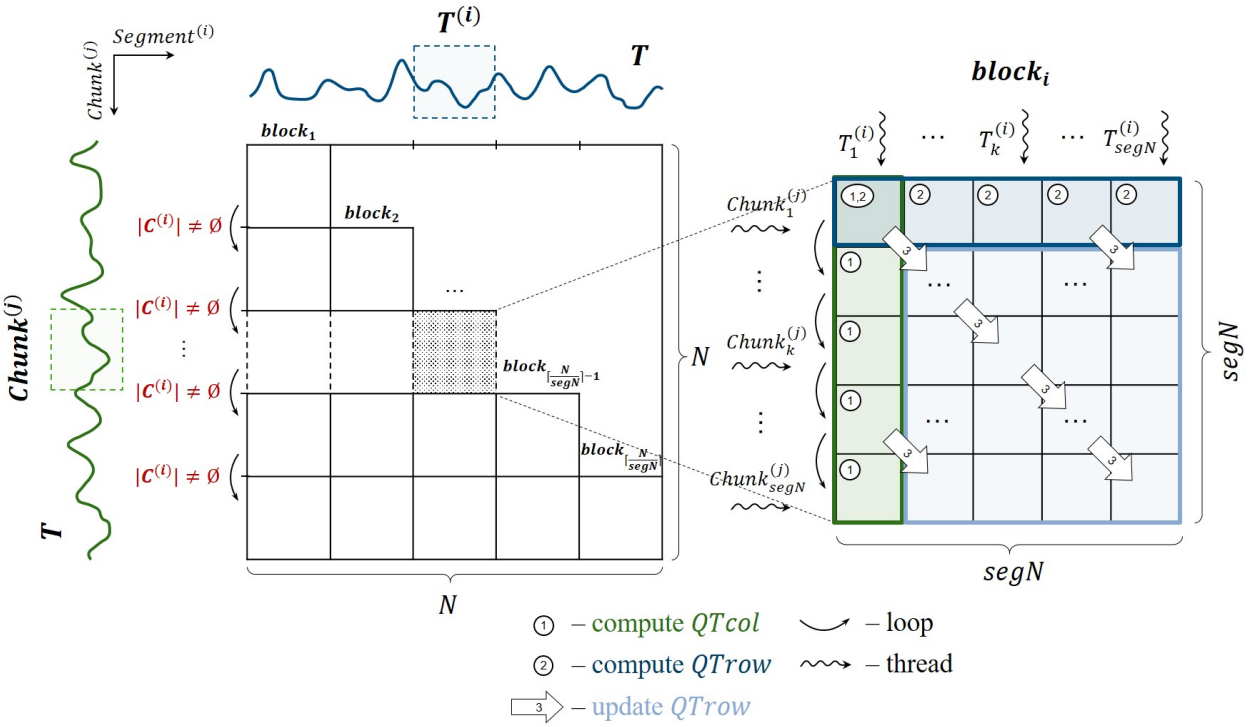


图 3. 候选项选择阶段的计算内核

图 3 详细展示了块中的线程如何工作。线程块在开始计算前会将其分段加载到共享内存中一次；在每个扫描步骤中，还会将位于该分段右侧的当前数据块加载到共享内存中。通过减少从全局内存读取时间序列元素的次数，这种技术能够提升算法性能。接下来，块中的线程计算标量积，并将结果存储在共享内存中：首先，计算该分段的第一个子序列与当前数据块中所有子序列之间的标量积；然后，计算当前数据块的第一个子序列与该分段中所有子序列之间的标量积（分别对应算法 3 第 4–7 行和第 8 行中的向量 $QTrow, QTcol \in \mathbb{R}^{segN}$ ）。

接下来，根据得到的向量 $QTcol$ 和预先计算的向量 $\bar{\mu}, \bar{\sigma}$，通过公式 6 计算数据块的第一个子序列与该分段中所有子序列之间的距离（见算法 3 第 9 行）。利用计算得到的距离，剔除位于该分段和当前数据块中缺乏希望的候选项（见算法 3 第 10–11 行）。如果该分段中的所有候选项都被剔除，则块停止运行（见算法 3 第 14–15 行）。

之后，块中的线程会对当前数据块剩余的子序列执行类似操作，但会更高效地计算标量积（见算法 3 的第 16–24 行）。我们根据先前计算得到的向量 $QTcol$ 和上一轮迭代中获得的向量 $QTcol$ ，计算当前数据块子序列与该分段的所有子序列（即向量 $QTcol$）之间的标量积（见算法 3 的第 16 行）。为计算标量

积，即 $Chunk^{(j)}$ 中第 k 个（$1 < k \leq segN$）子序列与该分段$T^{(i)}$中某个子序列之间的标量积，我们采用以下公式:

$$QTcol(tid) = \begin{cases} QTcol(tid-1) + T^{(i)}_{tid,\,m} \cdot Chunk^{(j)}_{k,\,m}(m) - \\ \quad - T^{(i)}_{tid-1,\,m}(1) \cdot Chunk^{(j)}_{k-1,\,m}(1), & 1 < tid \leq segN\,, \\ QTrow(k), & tid = 1 \end{cases} \tag{10}$$

其中 tid 表示块中的线程数。由于在公式 10 中，第一项取自上一轮迭代，因此标量积计算的复杂度为 $O(1)$，而非简单情况下的 $O(m)$。

3.3. 异常子序列精炼的并行化

算法 4 PD3REFINE (IN T, m, r; OUT $\mathcal{D}$)

对于所有 $T_{i,\,m} \in S^m_T$，**执行** ▷并行（线程）
  候选项(i) $\leftarrow$ 候选项(i) $\wedge$邻居(i)
对于所有满足 $T^{(i)} \in T$ **且** $\bigwedge_{k=i\cdot segN}^{(i+1)\cdot segN}$ 候选项(k) $=$ 为真 **的项，执行** ▷并行（块）
  对于所有 $Chunk^{(j)} \in T^{(i)}$，**其中** $i \geq j$，**执行** ▷并行（线程）
    如果 $i = j$，**则**
      QTrow $\leftarrow$ CALCDOTPRODUCTS（$T^{(i)}_{1,\,m}$，数据块$^{(j)}$）
      继续
    QTrow $\leftarrow$ UPDATEDOTPRODUCTS(*QTrow* $T^{(i)}_{1,\,m}$，数据块$^{(j)}$)
    QTcol $\leftarrow$ CALCDOTPRODUCTS（数据块$^{(j)}_{1,\,m}$，$T^{(i)}$）
    $dist \leftarrow$ CALCDIST（数据块$^{(j)}_{1,\,m}$ $T^{(i)}$, *QTcol* $\bar{\mu}, \bar{\sigma}$）
    如果 $dist < r$ **则**
      候选项$(i\cdot$*segN* $+$ *tid*$)$ $\leftarrow$ FALSE
    else
      最近邻距离$(j \cdot$ *segN* $+$ *tid*$)$ $\leftarrow$ 最小值(*dist*，最近邻距离$(j \cdot$ *segN* $+$ *tid*$)$)
    如果不是 $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN}$ 候选项(k) **则**
      中断
    对所有数据块$^{(j)}_{k,m} \in S^m_{Chunk^{(j)}} \setminus$ 数据块$^{(j)}_{1,\,m}$ **do** ▷并行（线程）
      QTcol $\leftarrow$ UPDATEDOTPRODUCTS(*QTcol* $QTrow, Chunk^{(j)}_{k}$ $_{,\,m}$, $T^{(i)}$)
      dist $\leftarrow$ CALCDIST(数据块$^{(j)}_{k,\,m}$ $T^{(i)}$, *QTcol* $\bar{\mu}, \bar{\sigma}$)
      如果 $dist < r$ **则**
        候选项$(i\cdot$*segN* $+$ *tid*$)$ $\leftarrow$ FALSE
      else
        最近邻距离$(j \cdot segN + tid)$ $\leftarrow$ 最小值(距离，最近邻距离$(j \cdot$ 分段编号 $+$ 线程 *ID*$)$)
    如果不是 $\bigvee_{k=i\cdot segN}^{(i+1)\cdot segN}$ 候选项(k) **则**
      中断
$\mathcal{D} \leftarrow \{\{T_{i,\,m} \in S^m_T;$ 最近邻距离$(i)\} \mid 1 \leq i \leq n-m+1,$ 候选项(i) $=$ TRUE$\}$
返回 $\mathcal{D}$

异常子序列精炼阶段（见算法 4）的并行化通过依次调用两个 CUDA 内核来实现。第一个内核通过对 $Cand$ 和 $Neighbor$ 位图向量进行逐元素合取，直接精炼上一阶段得到的异常子序列，并将结果写回前者。此操作可剪枝在选择阶段被丢弃的子序列的最近邻。

第二个内核执行非平凡的细化操作，其并行化方式类似于选择阶段，但仅涉及局部候选项集合非空的时间序列片段（见算法 4 第 3 行）。该算法扫描并处理不与候选项重叠、且位于分段左侧的子序列（见算法 4 第 4 行）。如果候选项与子序列之间的距离小于参数 r，则将该候选项丢弃为明显的假阳性。

4. 实验评估

为了评估所提出的算法，我们开展了实验，研究 PALMAD 在各种真实世界和合成时间序列上的表现，并与类似方法进行比较，同时考察了该算法的可扩展性。我们设计的实验易于复现，相关代码仓库 [27] 中包含算法的源代码以及本研究使用的所有数据集。下文中，第 4.1 节介绍实验所用的硬件和时间序列，第 4.2 节呈现实验结果并进行讨论。

4.1. 实验设置

表 1. 实验中使用的时间序列

时间序列	长度 (n)	异常片段长度 (最小长度 =最大长度)	领域
航天飞机	50 000	150	NASA 上某个传感器的测量值 航天器
ECG	45 000	200	成年患者的心电图
ECG-2	21 600	400	
Koski-ECG	100 000	458	
呼吸	24 125	250	通过胸廓扩张进行人体呼吸
电力需求	33 220	750	办公室年度能源消耗
RandomWalk1M	10^7	512	合成时间序列
RandomWalk2M	$2 \cdot 10^7$	512	

在我们的研究中，我们采用了表 1 所列的时间序列，这些数据也用于 HOTSAX [24], KBF GPU [46], 和 Zhu 等人的 [54] 算法的实验评估。航天飞机数据 [14] 是在实验室中对 Marotta MPV-41 系列阀门进行各种测试时测得的螺线管电流值；这些阀门在 NASA 航天器上用于控制燃料流量。ECG 和 ECG2 [17], 以及 Koski-ECG [26] 时间序列是

成年患者的心电图。呼吸时间序列 [24] 记录了一名患者醒来时的呼吸情况（通过胸廓扩张测量）。功率需求时间序列反映了荷兰研究中心的能源消耗，时间为 1997 [49]。RandomWalk1M 和 RandomWalk2M 是我们通过随机游走模型生成的时间序列 [37]。

表2. 实验所用的硬件平台

规格	GPU-SUSU	GPU-MSU
品牌和产品系列	NVIDIA Tesla	
型号	V100	P100
核心数	5 120	3 584
核心频率，GHz	1.3	1.19
内存，Gb	32	16
峰值性能（双精度），太 FLOPS	7	4

表 2 总结了我们实验所用的硬件平台，其中 GPU-SUSU 和 GPU-MSU 分别表示安装在南乌拉尔国立大学 [12] 和莫斯科国立大学 [47],高性能计算中心的图形处理器。

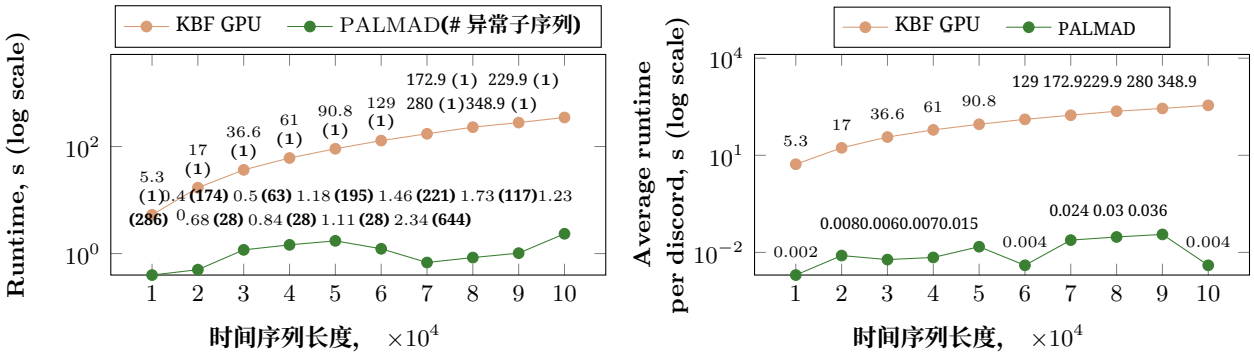
4.2. 结果与讨论

4.2.1. 与同类算法的比较

在实验中，我们将 PALMAD 与两种算法进行了比较，即 KBF GPU [46] 和 Zhu 等人提出的算法 [54]，因为我们对相关工作的全面综述（见第 1 节）未发现其他面向 GPU 的并行竞争算法。由于上述竞争算法的作者未提供其源代码，为了公平比较，实验中我们采用与 [46] 和 [54], 中所用相同的时间序列和硬件，并将我们的结果与作者在原论文中报告的结果进行比较。我们使用时间序列 Koski-ECG（见表 1），在 GPU-SUSU 系统（见表 2）上将 PALMAD 与 KBF GPU 进行比较；其余时间序列则用于在 GPU-MSU 上与 Zhu 等人提出的算法进行比较。我们省略了 MERLIN 的性能结果，因为正如预期，原始串行算法明显不如其并行后继算法；不过，在实验中我们确认 PALMAD 生成的结果与 MERLIN 完全相同。每项实验中，我们运行 PALMAD 10 次，并取平均值作为最终运行时间。

由于竞争算法只能发现 top-1 异常片段，而我们的算法能在指定长度范围内找出每种长度的所有异常子序列，为了进行公平比较，我们在实验中为 PALMAD 采用以下两种设置。首先，我们将 $minL = maxL$ 设为上述范围。其次，我们同时测量 PALMAD 的运行时间和发现的异常子序列数量，以进一步展示我们的算法发现单个异常子序列的平均耗时。

在图 4 中，我们比较了 PALMAD 和 KBF GPU 的性能。可以看出，在总体运行时间以及发现一个异常片段的平均运行时间方面，我们的算法都显著快于对手。显然，这是因为 KBF GPU 实现了



(a) 总体运行时间　　(b) 每个异常片段的平均运行时间

图4. PALMAD 与 KBF GPU 的性能比较

暴力方法，而 PALMAD 避免了冗余计算并利用了先进的数据结构。

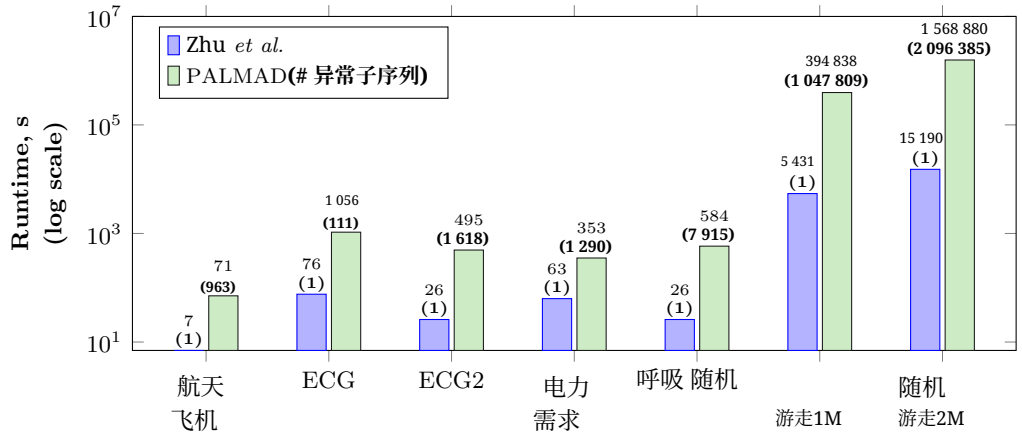
图5展示了 PALMAD 与 Zhu等人算法的性能对比实验结果。可以看出，Zhu等人的算法明显快于 PALMAD：在真实和合成时间序列上，速度分别最高可达其 20 倍和两个数量级。然而，与此同时，PALMAD 发现的异常子序列显著更多：在真实和合成时间序列上，数量分别至少高出两个和七个数量级。因此，比较发现一个异常片段的平均运行时间可以看出，从我们将待发现的异常子序列数量 $topK$ 设为实际发现数量的四分之一时起，PALMAD 就显著优于竞争算法：在真实和合成时间序列上，分别至少快两倍和三个数量级。

4.2.2. PALMAD 的可扩展性

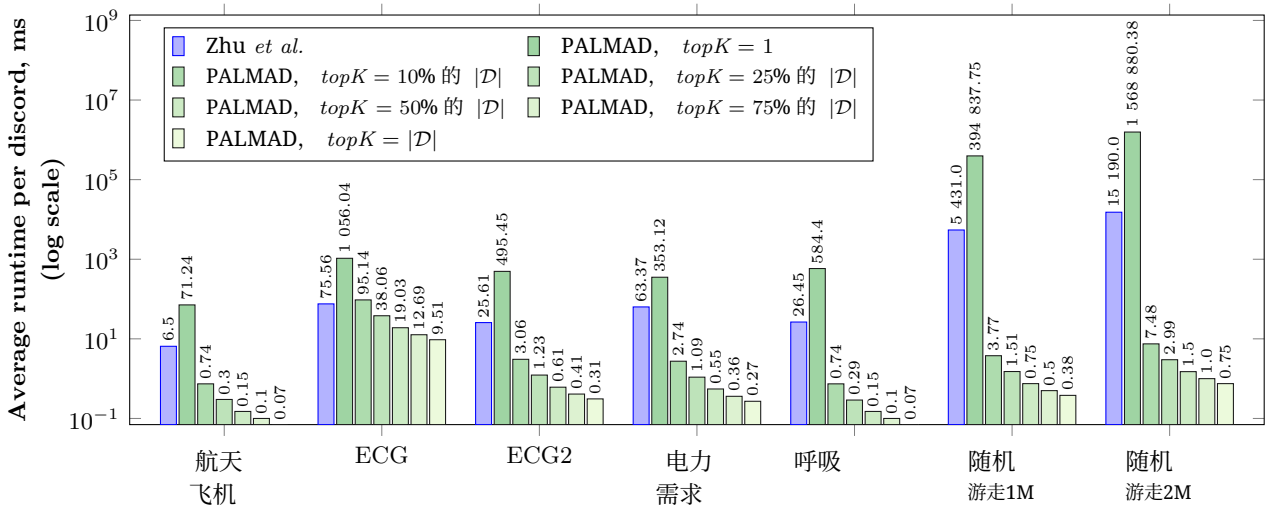
除了将我们的算法与同类算法进行比较外，我们还研究了 PALMAD 的可扩展性。首先，我们考察分段长度（参数 $seglen$，见第 3.2 节）对 PALMAD 性能的影响。其次，我们评估算法在两个直接影响计算量的输入参数下的性能，即时间序列长度和异常子序列范围长度。

在图 6 中，我们展示了片段长度对 PALMAD 性能影响的实验结果。可以看出，对于真实世界和合成时间序列，算法的运行时间都与片段长度成正比，并且片段长度的值越大，性能越高。这可以解释为，随着片段长度增加，在 GPU 共享内存中读取和写入片段的开销会降低。此外，这也是我们在上述实验中采用 $seglen = 512$ 的原因。

在图 7 和图 8 中，对于真实世界数据和合成数据，我们分别展示了算法性能随时间序列长度和 Discord 长度范围变化的情况。可以观察到，对于真实世界时间序列和合成时间序列，算法的运行时间都与上述参数成正比。



(a) 总体运行时间



（b）每个异常片段的平均运行时间

图5。 PALMAD 与 Zhu 等人的算法的性能比较

5. 案例研究

在本节中，我们将 PALMAD 应用于从智能供暖控制系统的真实世界时间序列中发现子序列异常。PolyTER系统 [64] 通过分析安装在大学校园建筑中的各种物联网传感器所采集的数据，实现对公用系统运行状况的智能监测与控制。我们选取了一个安装在演讲厅中的温度传感器所采集的时间序列，并在指定范围内发现异常。该传感器的采样频率为每小时 4 次，该时间序列对应年度测量数据（即时间序列长度 $n = 35\,040$），我们搜索的异常长度范围为 12 小时至 7 天（即分别为 $minL = 48$ 和 $maxL = 672$）。

为了可视化所得结果，我们提出了不一致热图技术，通过颜色深浅来呈现异常分数，与模式热图 [33]有些相似。

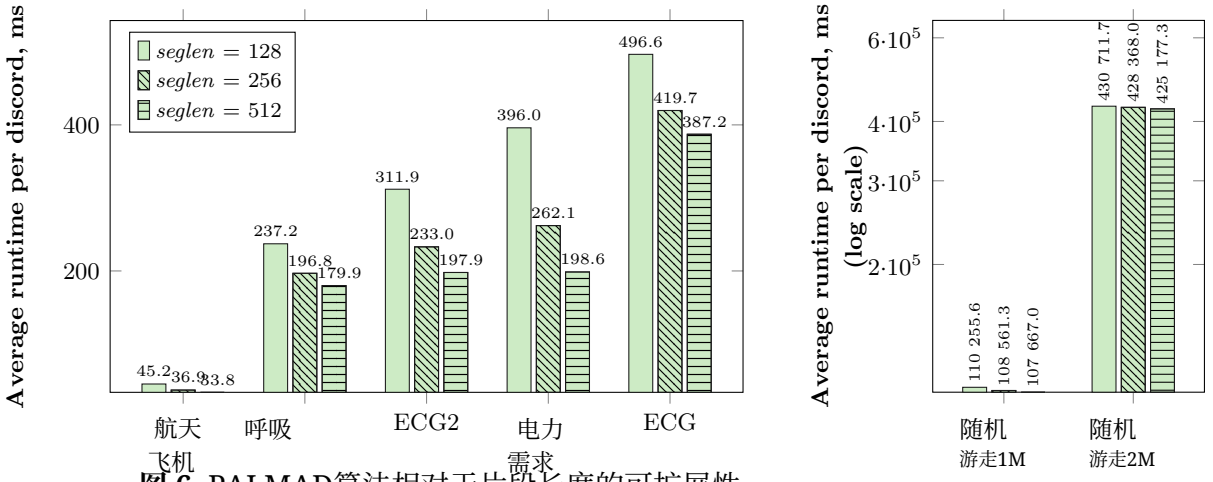
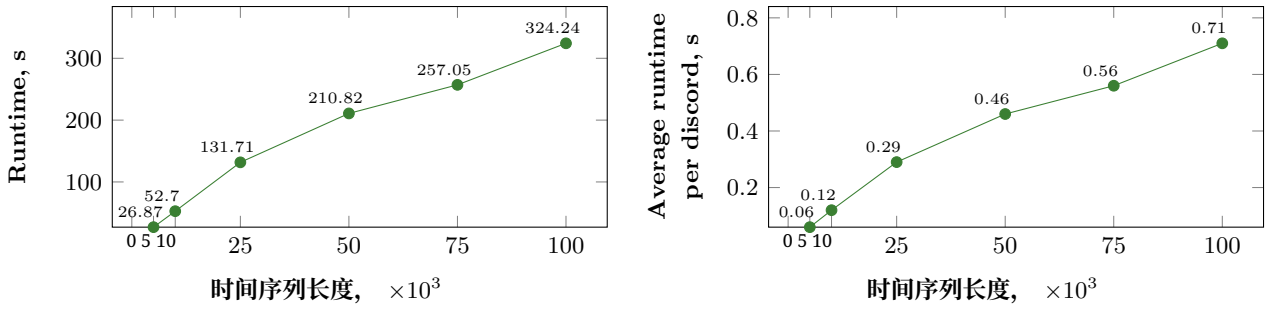
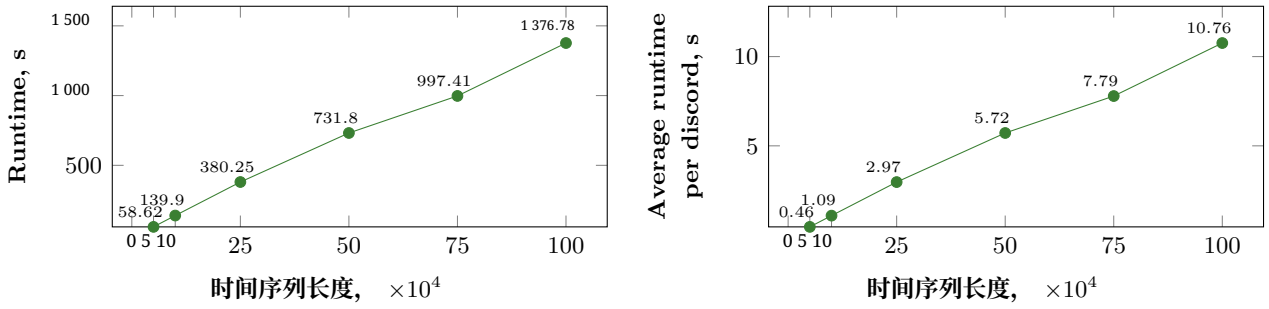


图 6. PALMAD算法相对于片段长度的可扩展性



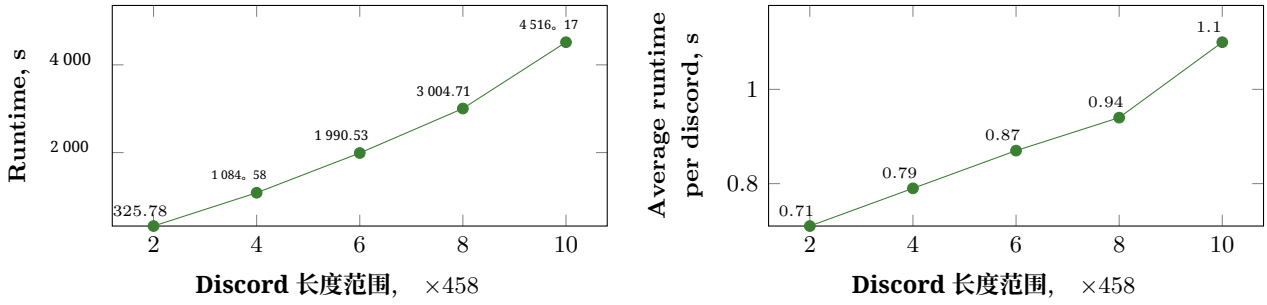
(a) 真实数据集 (Koski-ECG，discord范围为 458..916)



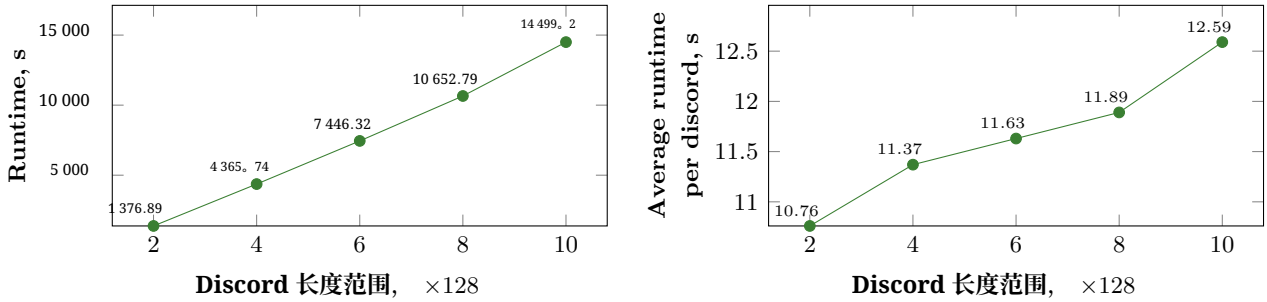
(b) 合成数据集（RandomWalk1M，discord范围为128..256）

图7。 PALMAD算法关于时间序列长度的可扩展性

严格来说，我们将单色热图绘制为一个大小为 $(maxL - minL + 1) \times (n - minL)$ 的矩阵，其中像素 (m, i) 的深浅表示异常片段 $T_{i,m} \in D_m$ 的异常分数，而像素的深浅则根据该异常片段与其最近邻之间的距离进行归一化计算



(a) 真实数据集 (Koski-ECG)



（b）合成数据集（RandomWalk1M）

图 8。 PALMAD算法相对于异常片段长度范围的可扩展性

邻居：

$$heatmap(m,\, i) = \frac{T_{i,\,m}.nnDist}{2m}, \tag{11}$$

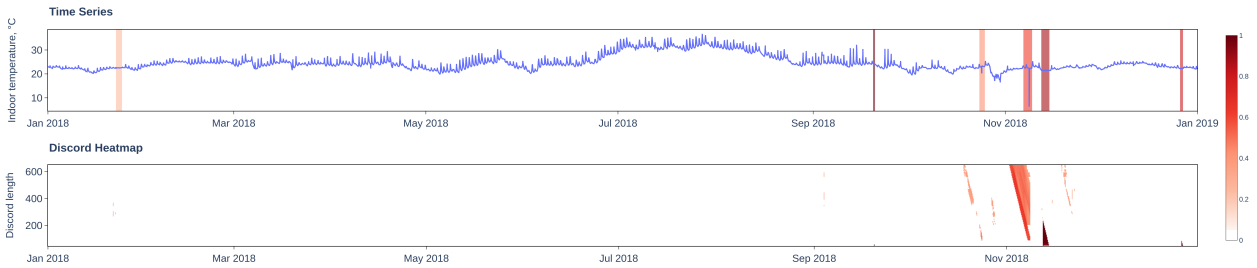
其中我们根据公式 6 使用归一化除数 $2m$ 。

尽管我们提出了一种可视化工具，用于探索和发现长度不一的异常片段，但对于从业者而言，仍有一个尚待解答的问题：如何对不同长度的异常片段进行排序，并提取出最有趣的异常片段。可以考虑多种异常片段属性，例如长度和索引、与其最近邻的距离、自匹配的数量等。然而，在本研究中，我们采用了一种直接的方法，将异常片段的兴趣度视为其与最近邻之间的归一化距离，并在具有相同索引的异常片段之间比较这些距离。因此，不同长度的异常片段中最有趣的异常片段按如下方式选出：

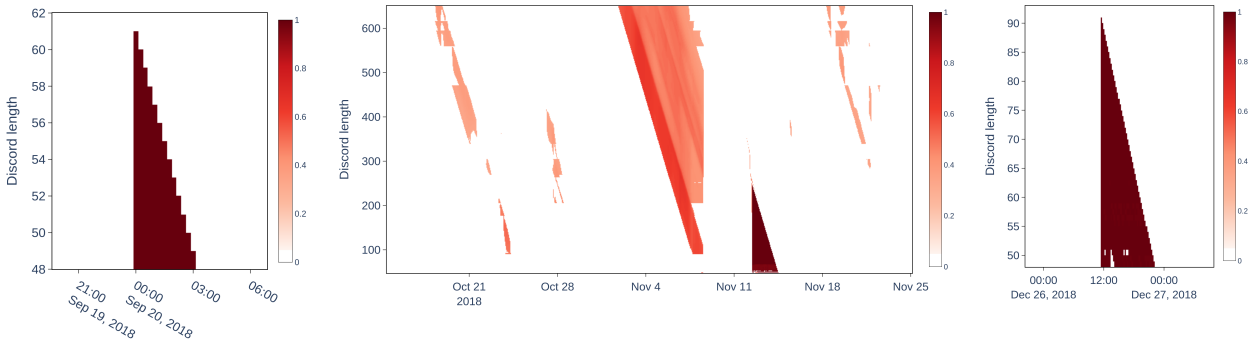
$$\arg \max_{1 \le i \le N} \max_{minL \le m \le maxL} heatmap(m,\, i). \tag{12}$$

显然，通过公式 12，我们可以选出排名前 k 的有趣异常子序列。在我们的代码仓库 [27]，中，读者可以找到表 1 中列出的所有真实世界时间序列以及其他主题领域的时间序列所对应的异常子序列热力图和排名靠前的异常子序列图。

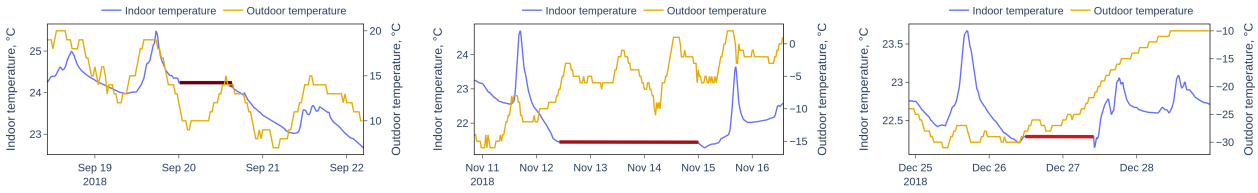
图 9 总结了案例研究的结果，展示了时间序列及其异常子序列热力图、热力图中最有趣异常片段所在区间的放大图，以及前 6 个异常片段。



(a) 时间序列及其异常子序列热力图



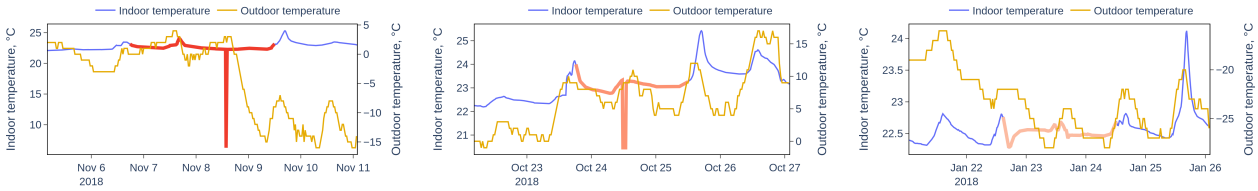
(b) 放大异常子序列热力图中最值得关关注的区间



top-1 异常片段， $m = 60$
（15 小时）

top-2 异常片段， $m = 248$
（2 天 14 小时）

排名第 3 的异常片段， $m = 90$
（22 小时 30 分钟）



排名第 4 的异常片段， $m = 266$
（2 天 18 小时 30 分钟）

排名第 5 的异常片段， $m = 168$
（1 天 18 小时）

第 6 个异常片段， $m = 183$
（1 天，21 小时 45 分钟）

（c）不同长度的异常子序列

图 9. PolyTER 案例

根据公式 12（分别见子图 9a、9b 和 9c）。此外，在前列异常片段图中，我们同时标出了室内和室外温度，后者取自开放天气档案 [1]。很可能，前三个异常片段分别反映了温度传感器的长期故障：在相当长的一段时间内，该传感器输出相同的测量值。接下来，第 4 和第 5 个异常片段显示了传感器的短期故障。最后，第 6 个异常片段可能表明

在所考虑的时期内，系统（或其操作人员）选择了效率低下的演讲厅供暖模式。

结论

本文探讨了如何在图形处理器上加速时间序列子序列异常发现这一问题。此类异常是指一段连续时间点的整体行为异常，尽管其中每个观测值单独来看未必异常。目前，在时间序列中发现子序列异常仍是最热门的研究问题之一。

在众多发现子序列异常的方法中，异常片段这一概念 [31] 被认为是最佳方法之一。直观而言，时间序列异常片段是指与其最近邻距离最远的子序列。然而，异常片段的应用受到用户选择的子序列长度的影响。通过穷举发现所有可能长度的异常片段，再根据某种度量选出最佳异常片段，显然在计算上难以承受。近期提出的 MERLIN 算法 [36] 能够发现指定长度范围内所有可能长度的时间序列异常片段，在准确性和性能方面领先于其他方法。MERLIN 算法通过反复调用 DRAG 算法 [51] 来发现给定长度的异常片段，这些异常片段与其最近邻的距离至少为 r ；此外，该算法还会自适应地选择参数 r。然而，据我们所知，尚无研究探讨如何利用任何并行硬件架构加速 MERLIN 算法。

本文基于 Keogh 等人的工作 [36, 51], ，提出了一种面向图形处理器的新型并行化方案 PALMAD（基于 MERLIN 的任意长度并行异常发现）。PALMAD 基本遵循原始串行算法，但采用我们推导出的递推公式来计算时间序列子序列的均值和标准差。由于这些数据还用于计算子序列之间的归一化欧几里得距离，最终大幅减少了计算量。此外，PALMAD 会反复调用 PD3（基于 DRAG 的并行 discord 发现） [28], ，这是我们开发的原始 DRAG 算法的并行版本。与其前身类似，PD3 分两个阶段执行。为实现候选选择阶段，我们将时间序列划分为长度相等的分段，以此利用数据并行性；每个分段由一个 GPU 线程块单独处理。线程块将该分段中的子序列视为异常片段的局部候选项，并处理位于分段右侧且与这些候选项不重叠的子序列。接着，线程块按数据块扫描子序列，每个数据块中的元素数量等于分段长度，且第一个数据块从该分段中的第 m 个元素开始，其中 m 是异常片段长度。这种技术使我们能够避免重复检查候选项与数据块中的子序列是否重叠。在 PD3 中，候选项细化阶段采用与选择阶段类似的方式进行并行化。细化阶段只涉及局部候选项集合非空的时间序列分段。算法扫描并处理与候选项不重叠且位于分段左侧的子序列。

我们针对真实世界和合成时间序列，对 PALMAD 进行了广泛的实验评估。在实验中，我们将所开发的算法与两种算法进行了比较，即 KBF GPU [46] 和 Zhu 等人的算法 [54]，因为我们对相关工作的全面审查未发现其他面向 GPU 的并行竞争算法。这两种竞争算法都旨在发现 top-1 异常片段；前者是暴力方法的并行化版本，后者则利用计算模式减少计算量。正如预期，在实验中，我们的算法显著快于 KBF GPU。接着，Zhu等人的算法经过调整后，可用于发现指定长度的最重要异常片段，其速度显著快于 PALMAD；后者能够发现指定范围内所有可能长度的异常子序列：在真实世界和合成时间序列上，前者分别快达 20 倍和两个数量级以上。然而，PALMAD 发现的异常子序列数量多得多：在真实世界和合成时间序列上，分别至少多两个和七个数量级。因此，在发现单个异常片段的平均运行时间方面，PAL- MAD 显著快于竞争算法。最后，我们还在实验中研究了 PALMAD 的可扩展性，发现对于真实世界和合成时间序列，算法的运行时间与以下各参数成正比：片段长度、时间序列长度和异常子序列范围长度。

我们还应用 PALMAD，采用我们提出的异常子序列热力图技术，从智能供暖控制系统的真实世界时间序列中发现异常，以展示结果。

我们的后续研究可以进一步探讨以下主题：(a) 利用基于 GPU 节点的高性能集群，在无法完全装入随机存取存储器的大型时间序列中进行异常子序列发现；(b) 将 PALMAD 应用于基于深度学习的在线时间序列异常检测。

致谢

本研究得到俄罗斯科学基金会（项目编号：23-21-00465）的经费支持。研究使用了莫斯科国立罗蒙诺索夫大学高性能计算资源共享研究设施的设备，以及南乌拉尔国立大学的超级计算机资源。

参考文献

1. 全球243个国家的天气：车里雅宾斯克。https://rp5.ru/Weather_in_Chelyabinsk/（2022年），访问日期：2022-11-17 2. Ahmad, S., Lavin, A., Purdy, S., Agha, Z.：面向流式数据的无监督实时异常检测。《神经计算》262，134–147（2017）。https://doi.org/10.1016/j.neucom.2017.04.070 3. Avogadro, P., Dominoni, M.A.：一种用于时间序列复杂异常片段搜索的快速算法：HOT SAX Time。《应用智能》（2022年）。https://doi.org/10.1007/s10489-021-02897-z 4. Bl´azquez-Garc´ıa, A., Conde, A., Mori, U., Lozano, J.A.：时间序列数据中的离群值/异常检测综述。ACM Comput. Surv. 54(3)，56:1–56:33（2021）。https://doi.org/10.1145/ 3444690

5. Bu, Y., Leung, O.T., Fu, A.W., Keogh, E.J., Pei, J., Meshkin, S.：WAT：在时间序列数据库中查找排名前k的异常子序列。载于：第7届SIAM国际数据挖掘会议论文集，2007年4月26–28日，美国明尼苏达州明尼阿波利斯。第449–454页。SIAM（2007年）。https: // doi.org/10.1137/1.9781611972771.43 6. Buu, H.T.Q., Anh, D.T.：基于iSAX符号表示的时间序列异常发现。载于：第3届国际知识与系统工程会议（KSE 2011），2011年10月14–17日，越南河内。第11–18页。IEEE计算机学会（2011年）。https://doi.org/ 10.1109/KSE.2011.11

7. Chandola, V., Banerjee, A., Kumar, V.：异常检测：综述。ACM Comput. Surv. 41(3), 15:1–15:58（2009）。https://doi.org/10.1145/1541880.1541882

8. Chandola, V., Cheboli, D., Kumar, V.：检测时间序列数据库中的异常。取自明尼苏达大学数字典藏库。https://hdl.handle.net/ 11299/215791（2009），访问日期：2022年-04-12

9. Chau, P.M., Duc, B.M., Anh, D.T.：借助 R树在流式时间序列中检测异常片段。载于：2018 年高级计算与应用国际会议（ACOMP），2018 年 11 月 27-29 日，胡志明市，越南。第 96–103 页（2018）。https://doi.org/10.1109/ACOMP.2018.00023

10. Choi, K., Yi, J., Park, C., Yoon, S.：时间序列数据异常检测的深度学习：综述、分析与指南。IEEE Access 9，120043–120065（2021）。https://doi.org/10. 1109/ACCESS.2021.3107975

11. Dean, J., Ghemawat, S.：MapReduce：在大型集群上实现简化数据处理。载于：Brewer, E.A., Chen, P.（编），第六届操作系统设计与实现研讨会（OSDI 2004），美国加利福尼亚州旧金山，2004年12月6日至8日。第137–150页。USENIX协会（2004），http://www.usenix.org/events/osdi04/tech/dean.html

12. Dolganina, N., Ivanova, E., Bilenko, R., Rekachinsky, A.：南乌拉尔国立大学的高性能计算资源。载于：Sokolinsky, L., Zymbler, M.（编），第16届国际并行计算技术会议（PCT 2022），俄罗斯杜布纳，2022年3月29日至41日，修订版精选论文。计算机与信息科学通信。第1618卷，第43–55页。施普林格（2022年）。https://doi.org/10.1007/978-3-031-11623-0_4

13. Farahani, I.V., Chien, A., King, R.E., Kay, M.G., Klenz, B.：从马尔可夫链视角进行时间序列异常检测。载于：Wani, M.A., Khoshgoftaar, T.M., Wang, D., Wang, H., Seliya, N.（编），第 18 届电气电子工程师学会国际机器学习与应用会议（ICMLA 2019），美国佛罗里达州博卡拉顿，2019 年 12 月 16—19 日。第 1000–1007 页。电气电子工程师学会（2019）。https://doi.org/10.1109/ICMLA.2019.00170

14. Ferrell, B., Santuro, S.：NASA 航天飞机阀门数据。http://www.cs.fit.edu/~ pkc/nasa/data/ (2005)，访问日期：2022-03-19

15. Fredkin, E.：字典树存储。《ACM 通讯》3(9)，490–499 (1960)。https://doi.org/10.1145/367390.367400 16. Fu, A.W., Leung, O.T., Keogh, E.J., Lin, J.：基于哈尔变换查找时间序列异常片段。载于：Li, X., Za¨ıane, O.R., Li, Z. (编)，高级数据挖掘与应用，第二届国际会议 ADMA 2006，中国西安，2006年8月14日至16日，论文集。《计算机科学讲义》，第4093卷，第31–41页。施普林格 (2006)。https://doi.org/ 10.1007/11811305_317. Goldberger, A.L., Amaral, L.A.N., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.：PhysioBank、PhysioToolkit 和 PhysioNet：复杂生理信号研究新资源的组成部分。《循环》101(23)，215–220 (2000)。https://doi.org/10.1161/01.CIR.101.23.e215 18. Huang, T., Zhu, Y., Mao, Y., Li, X., Liu, M., Wu, Y., Ha, Y., Dobbie, G.：并行异常片段发现。载于：Bailey, J., Khan, L., Washio, T., Dobbie, G., Huang, J.Z., Wang, R. (编)，知识发现与数据挖掘进展——第20届亚太会议，PAKDD 2016，新西兰奥克兰，2016年4月19日至22日，论文集，第二部分。《计算机科学讲义》，第9652卷，第233–244页。施普林格 (2016)。https://doi.org/10.1007/978-3-319-31750-2_1919. Huang, T., Zhu, Y., Wu, Y., Shi, W.：J 距离异常片段：一种改进的时间序列异常片段定义与发现方法。载于：2015年IEEE数据挖掘国际会议研讨会 (ICDMW)。第303–310页 (2015)。https://doi.org/10.1109/ICDMW.2015.120 20. Hundman, K., Constantinou, V., Laporte, C., Colwell, I., S¨oderstr¨om, T.：使用长短期记忆网络和非参数动态阈值检测航天器异常。载于：Guo, Y., Farooq, F. (编)，第24届ACM SIGKDD知识发现与数据挖掘国际会议论文集，KDD 2018，英国伦敦，2018年8月19日至23日。第387–395页。ACM (2018)。https://doi.org/10.1145/3219819.3219845 21. Imani, S., Keogh, E.J.：矩阵轮廓 XIX：时间序列语义模式——一种用于发现时间序列高层结构的新原语。载于：Wang, J., Shim, K., Wu, X. (编)，2019年IEEE数据挖掘国际会议，ICDM 2019，中国北京，2019年11月8日至11日。第329–338页。电气电子工程师学会 (2019)。https://doi.org/10.1109/ICDM.2019.00043 22. Imani, S., Madrid, F., Ding, W., Crouter, S.E., Keogh, E.J.：介绍时间序列片段摘要：一种用于概括长时间序列的新原语。《数据挖掘与知识发现》34(6)，1713–1743 (2020)。https://doi.org/10.1007/s10618-020-00702-y 23. Keogh, E.J., Chakrabarti, K., Pazzani, M.J., Mehrotra, S.：大型时间序列数据库中的快速相似性搜索降维。《知识与信息系统》3(3)，263–286 (2001)。https: //doi.org/10.1007/PL00011669 24. Keogh, E.J., Lin, J., Fu, A.W.：HOT SAX：高效查找最异常的时间序列子序列。载于：第5届IEEE数据挖掘国际会议论文集 (ICDM 2005)，2005年11月27日至30日，美国得克萨斯州休斯敦。第226–233页。IEEE计算机学会 (2005)。https://doi.org/10.1109/ICDM.2005.79

25. Kirk, D.B.：NVIDIA CUDA 软件与 GPU 并行计算架构。载于：Morrisett, G., Sagiv, M.（编）《第六届国际内存管理研讨会论文集》，ISMM 2007，加拿大魁北克省蒙特利尔，2007年10月21—22日。第103–104页。ACM（2007）。https: //doi.org/10.1145/1296907.1296909 26. Koski, A.：结构性心电图特征的原始编码。《模式识别快报》17(11)，1215–1222（1996）。https://doi.org/10.1016/0167-8655(96)00079-7 27. Kraeva, Y., Zymbler, M.：PALMAD：基于 MERLIN、面向 GPU 的并行异常发现算法。https://github.com/kraevaya/PALMAD（2022），访问日期：2022-12-01 28. Kraeva, Y., Zymbler, M.：在图形处理器上进行时间序列异常发现的并行算法。《模式识别与图像分析》33(2)，已接受发表（2023）29. Kraeva, Y., Zymbler, M.L.：在 Phi KNL 集群上对超大型时间序列数据进行子序列相似性搜索的可扩展算法。载于：Manolopoulos, Y., Stupnikov, S.A.（编）《数据密集型领域的数据分析与管理——第20届国际会议 DAMDID/RCDL 2018，俄罗斯莫斯科，2018年10月9—12日，修订版精选论文》。计算机与信息科学通信丛书，第1003卷，第149–164页。施普林格（2018）。https://doi.org/10.1007/978-3-030-23584-0_9

30. Li, G., Br¨aysy, O., Jiang, L., Wu, Z., Wang, Y.：基于位表示聚类发现时间序列 discord。《知识与信息系统》54，243–254（2013）。https://doi.org/10.1016/ j.knosys.2013.09.015

31. Lin, J., Keogh, E.J., Fu, A.W., Herle, H.V.：魔术的近似：寻找异常医疗时间序列。载于：第十八届 IEEE 计算机医学系统研讨会（CBMS 2005），2005 年 6 月 23—24 日，爱尔兰都柏林。第 329–334 页。IEEE计算机学会（2005）。https://doi. org/10.1109/CBMS.2005.34 32. Lin, J., Keogh, E.J., Lonardi, S., Chiu, B.Y.：时间序列的符号表示及其对流式算法的影响。载于：Zaki, M.J., Aggarwal, C.C.（编），第八届 ACM SIGMOD 数据挖掘与知识发现研究问题研讨会论文集，DMKD 2003，2003 年 6 月 13 日，美国加利福尼亚州圣迭戈。第 2–11 页。ACM（2003）。https:// doi.org/10.1145/882082.882086 33. Madrid, F., Imani, S., Mercer, R., Zimmerman, Z., Senobari, N.S., Keogh, E.J.：矩阵剖面 XX：使用矩阵剖面查找并可视化所有长度的时间序列模式。载于：Gao, Y., M¨oller, R., Wu, X., Kotagiri, R.（编），2019 年 IEEE 国际大知识会议，ICBK 2019，中国北京，2019 年 11 月 10—11 日。第 175–182 页。电气电子工程师学会（2019），https://doi. org/10.1109/ICBK.2019.00031 34. Mueen, A., Nath, S., Liu, J.：海量时间序列数据的快速近似相关性。载于：Elmagarmid, A.K., Agrawal, D.（编），ACM SIGMOD 数据管理国际会议论文集，SIGMOD 2010，美国印第安纳州印第安纳波利斯，2010 年 6 月 6—10 日。第 171–182 页。ACM（2010）。https://doi.org/10.1145/1807167.1807188

35. Munir, M., Siddiqui, S.A., Dengel, A., Ahmed, S.：DeepAnT：一种用于时间序列无监督异常检测的深度学习方法。IEEE Access 7，1991–2005（2019）。https://doi.org/10.1109/ACCESS.2018.2886457 36. Nakamura, T., Imamura, M., Mercer, R., Keogh, E.J.：MERLIN：在海量时间序列档案中无需参数即可发现任意长度的异常。载于：Plant, C., Wang, H., Cuzzocrea, A., Zaniolo, C., Wu, X.（编），第二十届 IEEE 数据挖掘国际会议，ICDM 2020，意大利索伦托，2020 年 11 月 17—20 日。第 1190–1195 页。电气电子工程师学会（2020）。https://doi.org/ 10.1109/ICDM50108.2020.00147 37. Pearson, K.：随机游走问题。《自然》72(1865)，294（1905）。https://doi. org/10.1038/072342a0 38. Pfeilschifter, G.：在高性能计算系统上使用矩阵剖面进行时间序列分析。硕士论文，慕尼黑工业大学信息学系，德国（2019），http: //mediatum.ub.tum.de/doc/1471292/1471292.pdf，访问日期：2022-03-17 39. Reyes, R., L´opez-Rodr´ıguez, I., Fumero, J.J., de Sande, F.：对 OpenACC 实现的初步评估。《超级计算杂志》65(3)，1063–1075（2013）。https://doi.org/10. 1007/s11227-012-0853-z 40. Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A.P., Chen, C., Frankenstein, S.：基于语法压缩的时间序列异常发现。载于：Alonso, G., Geerts, F., Popa, L., Barcel´o, P., Teubner, J., Ugarte, M., den Bussche, J.V., Paredaens, J.（编），第十八届数据库技术拓展国际会议论文集，EDBT 2015，比利时布鲁塞尔，2015 年 3 月 23—27 日。第 481–492 页。OpenProceedings.org（2015）。https: //doi.org/10.5441/002/edbt.2015.42 41. Shieh, J., Keogh, E.J.：iSAX：太字节级时间序列的索引与挖掘。载于：Li, Y., Liu, B., Sarawagi, S.（编），第十四届 ACM SIGKDD 知识发现与数据挖掘国际会议论文集，美国内华达州拉斯维加斯，2008 年 8 月 24—27 日。第 623–631 页。ACM（2008）。https://doi.org/10.1145/1401890.1401966 42. Snir, M.：技术视角：MPI 的未来。《ACM 通讯》61(10)，105（2018）。https://doi.org/10.1145/3264415 43. Son, N.T.：改进一种磁盘感知的异常片段发现算法，用于发现时间序列异常片段。载于：2020 年第五届绿色技术与可持续发展国际会议（GTSD），2020 年 11 月 27—28 日，越南胡志明市。第 19–23 页（2020）。https://doi.org/10.1109/GTSD50082.2020.9303111 44. de Supinski, B.R., Scogland, T.R.W., Duran, A., Klemm, M., Bellido, S.M., Olivier, S.L., Terboven, C., Mattson, T.G.：OpenMP 的持续演进。《IEEE 会议录》106(11)，2004–2019（2018）。https://doi.org/10.1109/JPROC.2018.2853600 45. Thuy, H.T.T., Anh, D.T., Chau, T.N.V.：一种用于时间序列异常片段发现的高效哈希算法。载于：2016 年第三届国家科学与技术基金会

信息与计算机科学发展会议（NICS），2016年9月14日至16日，越南岘港。第85–90页（2016）。https://doi.org/10.1109/NICS.2016.7725673

46. Thuy, T.T.H., Anh, T.D., Chau, T.N.V.: 一种新的异常片段定义以及一种利用图形处理器进行高效时间序列 discord 检测的方法。载于：ICSED 2021：2021年第三届国际软件工程与发展会议，2021年11月19日至21日，中国厦门。第63–70页（2021）。https://doi.org/10.1145/3507473.3507483

47. Voevodin, V.V., Antonov, A.S., Nikitenko, D.A., Shvets, P.A., Sobolev, S.I., Sidorov, I.Y., Stefanov, K.S., Voevodin, V.V., Zhumatiy, S.A.: Lomonosov-2超级计算机：面向用户群体的大规模深度监控与精细化分析。《超级计算机前沿与创新》6(2)，4–11（2019）。https://doi.org/10.14529/jsfi190201

48. Wei, L., Keogh, E., Xi, X.: SAX显式图像：寻找异常形状。载于：第六届数据挖掘国际会议（ICDM'06），第711–720页（2006）。https://doi.org/10. 1109/ICDM.2006.138

49. van Wijk, J.J., van Selow, E.R.: 基于聚类和日历的时间序列数据可视化。载于：1999年IEEE信息可视化研讨会（INFOVIS'99），美国加利福尼亚州旧金山，1999年10月24日至29日。第4–9页。IEEE计算机学会（1999）。https://doi.org/10. 1109/INFVIS.1999.801851

50. Wu, Y., Zhu, Y., Huang, T., Li, X., Liu, X., Liu, M.: 分布式discord发现：基于Spark的时间序列异常检测。载于：第17届电气电子工程师学会高性能计算与通信国际会议（HPCC 2015）、第7届电气电子工程师学会网络空间安全国际研讨会（CSS 2015）和第12届电气电子工程师学会嵌入式软件与系统国际会议（ICESS 2015），美国纽约州纽约，2015年8月24日至26日。第154–159页。电气电子工程师学会（2015）。https://doi.org/10.1109/HPCC-CSS-ICESS.2015.228

51. Yankov, D., Keogh, E.J., Rebbapragada, U.: 磁盘感知的异常片段发现：在太字节级数据集中查找异常时间序列。载于：第7届IEEE数据挖掘国际会议论文集（ICDM 2007），2007年10月28日至31日，美国内布拉斯加州奥马哈。第381–390页。IEEE计算机学会（2007）。https://doi.org/10.1109/ICDM.2007.61

52. Yankov, D., Keogh, E.J., Rebbapragada, U.: 磁盘感知的异常片段发现：在太字节级数据集中查找异常时间序列。知识与信息系统 17(2)，241–262（2008）。https://doi.org/10.1007/s10115-008-0131-9

53. Yeh, C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Zimmerman, Z., Silva, D.F., Mueen, A., Keogh, E.J.: 时间序列连接、模式、异常子序列与形状片段：利用矩阵剖面的统一视角。数据挖掘与知识发现 32(1)，83–123（2018）。https: //doi.org/10.1007/s10618-017-0519-9

54. Zhu, B., Jiang, Y., Gu, M., Deng, Y.：一种基于基元和异常片段的模式挖掘图形处理器加速框架。电气电子工程师学会并行分布式系统汇刊 32(8)，1987–2004（2021）。https: //doi.org/10.1109/TPDS.2021.3055765

55. Zhu, Y., Imamura, M., Nikovski, D., Keogh, E.J.：引入时间序列链：时间序列数据挖掘的新基本操作。知识与信息系统 60(2)，1135–1161（2019）。https: // doi.org/10.1007/s10115-018-1224-8

56. Zhu, Y., Yeh, C.M., Zimmerman, Z., Kamgar, K., Keogh, E.J.：矩阵剖面 XI：SCRIMP++：交互式速度下的时间序列模式发现。载于：电气电子工程师学会数据挖掘国际会议，ICDM 2018，新加坡，2018年11月17日至20日。第837–846页。IEEE计算机学会（2018）。https://doi.org/10.1109/ICDM.2018.00099

57. Zimmerman, Z., Kamgar, K., Senobari, N.S., Crites, B., Funning, G.J., Brisk, P., Keogh, E.J.：矩阵剖面 XIV：利用图形处理器扩展时间序列模式发现，每天突破十亿亿次成对比较并更进一步。载于：ACM云计算研讨会论文集，SoCC 2019，圣克鲁斯，加利福尼亚州，美国，2019年11月20–23日。第74–86页。ACM（2019）。https://doi.org/10.1145/3357223.3362721

58. Zymbler, M.：一种用于多核加速器上时间序列的并行异常子序列发现算法。数值方法与编程 20(3), 211–223（2019）。https://doi.org/0.26089/NumMet.v20r320，（俄文）

59. Zymbler, M., Goglachev, A.：使用图形处理器快速摘要长时间序列。数学 10(10), 1781（2022年）。https://doi.org/10.3390/math10101781

60. Zymbler, M., Grents, A., Kraeva, Y., Kumar, S.：一种用于海量时间序列数据中异常子序列发现的并行方法。计算机、材料与连续体 66(2), 1867–1878（2021）。https://doi.org/10.32604/cmc.2020.014232

61. Zymbler, M., Ivanova, E.：一种在关系数据库管理系统内部进行工业传感器数据分析的基于矩阵剖面的方法。数学 9(7), 2146（2021）。https://doi.org/10.3390/math9172146

62. Zymbler, M., Kraeva, Y.：在英特尔众核系统上发现时间序列模式。Lobachevskii J. Math. 40(12), 2124–2132 (2019)。https://doi.org/10.1134/ S199508021912014X

63. Zymbler, M., Kraeva, Y.：在图形处理器上进行时间序列模式发现的并行算法。《南乌拉尔国立大学学报：计算数学与软件工程》9(3)，17–34（2020）。https://doi.org/10.14529/cmse200302，（俄文）64. Zymbler, M., Kraeva, Y., Latypova, E., Kumar, S., Shnayder, D., Basalaev, A.：智能供暖控制系统中的传感器数据清洗。载于：2020年全球智能工业会议论文集，GloSIC 2020，俄罗斯车里雅宾斯克，2020年11月17日至19日。第375–381页（2020）。https://doi.org/10.1109/GloSIC50886.2020.9267813 65. Zymbler, M., Polyakov, A., Kipnis, M.：英特尔多核系统上的时间序列异常发现。载于：Sokolinsky, L., Zymbler, M.（编），第13届国际会议 PCT 2019，俄罗斯加里宁格勒，2019年4月2日至4日，修订精选论文。《计算机与信息科学通信》第1063卷，第168–182页。施普林格，Cham（2019）。https://doi.org/10. 1007/978-3-030-28163-2_12

附录

引理1。 设时间序列为 T, $|T| = n$，其两个长度为 m 的子序列为 $T_{i,\ m}$ 和 $T_{i,\ m+1}$，其中 $1 \le i \le n-m$ 和 $3 \le m \ll n$。则有:

$$\mu_{T_{i,\,m+1}} = \frac{1}{m+1}\big(m\mu_{T_{i,\,m}} + t_{i+m}\big),$$

$$\sigma^2_{T_{i,\,m+1}} = \frac{m}{m+1}\Big(\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(\mu_{T_{i,\,m}} - t_{i+m}\big)^2\Big).$$

证明。 首先，我们来证明关于均值的等式。根据定义

$$\mu_{T_{i,\,m}} = \frac{1}{m}\sum_{k=0}^{m} t_{i+k}.$$

Then

$$\sum_{k=0}^{m} t_{i+k} = m\mu_{T_{i,\,m}}.$$

接下来，考虑 $\mu_{T_{i,\ m+1}}$:

$$\mu_{T_{i,\,m+1}} = \frac{1}{m+1}\sum_{k=0}^{m} t_{i+k} = \frac{1}{m+1}(m\mu_{T_{i,\,m}} + t_{i+m}),$$

因此，第一个等式得证。接下来，我们来证明关于标准差的等式。根据定义

$$\sigma^2_{T_{i,\,m}} = \frac{1}{m}\sum_{k=0}^{m-1} t^2_{i+k} - \mu^2_{T_{i,\,m}}.$$

Then

$$\sum_{k=0}^{m-1} t^2_{i+k} = m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}).$$

接下来，考虑 $\sigma^2_{T_{i,\ m+1}}$:

$$\sigma^2_{T_{i,\,m+1}} = \frac{1}{m+1}\sum_{k=0}^{m} t^2_{i+k} - \mu^2_{T_{i,\,m+1}} = \frac{1}{m+1}\big(m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}) + t^2_{i+m}\big) - \mu^2_{T_{i,\,m}}.$$

利用上述已证明的关于均值的命题，我们得到

$$\sigma^2_{T_{i,\,m+1}} = \frac{1}{m+1}\big(m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}) + t^2_{i+m}\big) - \big(\frac{1}{m+1}(m\mu_{T_{i,\,m}} + t_{i+m})\big)^2.$$

对括号内的项进行运算并进一步合并同类项，可得

$$\sigma^2_{T_{i,\,m+1}} = \frac{1}{m+1}\big(m(\sigma^2_{T_{i,\,m}} + \mu^2_{T_{i,\,m}}) + t^2_{i+m} - \frac{1}{m+1}(m\mu_{T_{i,\,m}} + t_{i+m})^2\big) =$$

$$= \frac{1}{m+1}\Big(m\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(m(m+1)\mu^2_{T_{i,\,m}} - m^2\mu_{T_{i,\,m}} + (m+1)t^2_{i+m} - t^2_{i+m} - 2m\mu_{T_{i,\,m}}t_{i+m}\big)\Big) =$$

$$= \frac{1}{m+1}\Big(m\sigma^2_{T_{i,\,m}} + \frac{m}{m+1}\big(\mu^2_{T_{i,\,m}} - 2\mu_{T_{i,\,m}}t_{i+m} + t^2_{i+m}\big)\Big) =$$

$$= \frac{m}{m+1}\Big(\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(\mu_{T_{i,\,m}} - t_{i+m}\big)^2\Big).$$

至此，证明完毕。 □