申请上海交通大学博士学位论文

面向超大规模时间序列的异常检测

专　　业： 电子科学与技术

博 士 生： 黄田

导　　师： 祝永新　副教授

上海交通大学电子信息与电气工程学院

2015 年 7 月

Ph.D. Dissertation Submitted to Shanghai Jiao Tong University

LARGE SCALE TIME SERIES DISCORD DISCOVERY

Author: Tian Huang

Advisor: Associate Prof. Yongxin Zhu

Specialty: Electrical Engineering

School of Electronics and Electric Engineering

Shanghai Jiao Tong University

July 31, 2015

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在_____年解密后适用本授权书。

本学位论文属于

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：黄田　指导教师签名：

日期：2016年1月15日　　日期2016年1月15日

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：黄田

日期：2016年1月15日

面向超大规模时间序列的异常检测

摘 要

时间序列是对某个物理量进行等时间间隔观测所得的数值序列，能够反映受监测事物的状态、状况。时间序列的异常检测方法能够检测出时序序列中的异常，同时能够评价异常的程度，帮助用户了解受观测事物的状态与情况。时间序列的种类和规模的爆炸性增长对时间序列异常检测提出新的要求。现有时间序列异常检测方法存在检测效果差下，检测性能低下的问题。

本文提出一套面向超大时间序列的异常检测系统，该系统结合本文提出的多种检测方法与方案，改善现有时间序列异常检测存在的问题：1）超大规模时间序列中同一异常重复出现的概率增加，现有异常定义无法发现这类异常，本文提出基于 J 距离的时间序列异常定义（J-distance Discord，JDD），使用 J 距离作为衡量该子序列异常程度的标准，实验表明 JDD 与传统定义相比，能够捕捉到更有意义异常；2）多维时间序列可用于描述复杂系统的多种状态和情况，传统异常检测方法无法发现异常发生的原因，本文提出多维时间序列异常的检测方案（Multi-dimensional Discord Discovery，MDD），我们首先提出异常溯源方法（Dimension Reasoning LOF，DR-LOF），通过不同维度对异常的贡献程度分析异常发生的原因，同时起到异常溯源和数据降维的作用。MDD 结合 DR-LOF 与 JDD 形成完整的检测方案。在云计算环境的多种异常的检测案例表明 MDD 更快更有效地检测异常以及发现异常发生的原因；3）为减少磁盘操作对计算时间的影响，缓解单一计算节点存储空间对异常检测规模的限制，我们提出并行化的时间序列异常检测方法（Parallel Discord Discovery，PDD），PDD 将时间序列分段存储在多个计算节点，并行求解子序列在所有分段内的最近邻距离，并通过高效地节点间通讯保证检测结果的正确性。PDD 通过分布式的异常估算方法（Distributed Discord Estimation，DDE）和剪枝技术降低计算复杂度，且通过批量处理数据、多发射的方法提高资源利用率。我们使用 Apache Spark 实现 PDD，在随机时间序列数据集上的实验证明，相对于单线程的

HOTSAX 检测方法，10 个计算节点使 PDD 获得 8 倍的加速比，与基于磁盘的异常检测方法相比，PDD 的计算资源利用率提高了将近一倍；4）通过分治法加速时间序列异常检测会使检测结果不正确。我们提出了近似的并行化时间序列异常检测方法（Approximated Parallel Discord Discovery，APDD），把时间序列分段并行化检测异常，提出自适应停止条件（Adaptive Stop Criteria，ASC）用于在兼顾检测结果正确性的前提下降低计算复杂度。理论分析表明 APDD 降低了异常检测的计算复杂度，实验证明 APDD 在单机环境下比经典的 HOTSAX 检测方法快 2 至 14 倍， APDD 检测结果的前 6 位与原始定义的检测结果保持一致，且检测结果的正确性对 APDD 的并行度不敏感。

总之，超大规模时间序列给时间序列异常检测带来检测效果与检测性能两方面的新挑战，本文针对现有时间序列异常检测方法的不足，提出了面向超大规模时间序列的异常检测系统，系统组合多种检测方法与方案，从检测效果与计算复杂度两个方面进行改进，实验结果表明，组合了 JDD，MDD，PDD，APDD 的异常检测系统有效改善由数据规模引起的时间序列异常检测中的检测效果差与计算性能差的问题。

关键词：时间序列异常，异常溯源，异常近似，并行化异常检测

LARGE SCALE TIME SERIES DISCORD DISCOVERY

ABSTRACT

A time series is a sequence of data points that consists of successive measurements made over a fixed time interval. Time series reflects the state and status of the objects being measured. The goal of time series anomaly detection is to find interesting or unusual patterns, which may provide important information for application purpose. The increasing number of categories and scales impose challenges to the time series detection method. Existing methods suffer from the issues of poor effectiveness, low performance.

This dissertation proposes an time series discord discovery system that coordinates multiple methods and scheme to mitigate these issues：1) In large scale time series, a discord may appear several times. Original definition of discord hardly discover multiple occurrences of the same discord. We propose J-distance Discord (JDD), which takes the distance of a subsequence and its J^{th} nearest neighbor as the criteria of its discordness. Experiments on one synthetic and four real-world datasets show that JDD captures more de-fecto anomalies than original definition does. 2) Multi-dimensional time series describes multiple aspects of an object. Original definition of discord find the moment of discord but cannot find the reason of being anomalous. We propose Multi-dimensional Discord Discovery (MDD) Scheme. MDD combines JDD and our newly proposed Dimension Reasoning LOF method (DR-LOF), which finds the dimension that has the most contribution of being anomalous and reduces the dimensionality of the dataset. Experiments on detecting various intrusions in cloud computing environment show that MDD detects discord more quickly and effectively provides clues of the discord to users. 3) To reduce the time on disk I/O operation and to mitigate the limitation of a single computing node to the scalability of discord discovery, we propose Parallel Discord Discovery (PDD) and implement PDD with Apache Spark. PDD stores segmentations of a time

series in distributed computing platform and find nearest neighbor distance of every subsequence in parallel. PDD reduce computational complexity though our newly proposed Distributed Discord Estimation method (DDE) and early abandon technique. PDD improve the utilization of computational resources by batch processing and multi-issuing. Experiments on randomly generated time series datasets show that given 10 computing node, PDD is 8 times faster than classic discovery method HOTSAX. Compared to disk based discovery method, PDD almost double the utilization of computational resources. 4) Discord discovery cannot be parallelized by divide and conquer technique because this will results in incorrect results. We propose Approximated Parallel Discord Discovery (APDD). APDD discover discord respectively from different segmentation of a time series and use a verification phase to improve the correctness of the results. We also propose Adaptive Stop Criteria (ASC) to simultaneously improve the correctness of the results and reduce the computational complexity. Experiments on four real-world datasets show that given a single computing node APDD is 2-14 times faster than classic discovery method HOTSAX. APDD produce top six discord exactly identical to the results of the original definition. Furthermore, the correctness of APDD is not sensitive to the parallelism.

In conclusion, large scale time series brings challenges to time series discord discovery from two aspects: effectiveness and computational complexity. This dissertation propose a time series discord discovery system that conbines multiple methods and scheme to mitigate the issues of existing discovery methods. Experiments show that the system that combines JDD, MDD, PDD and APDD discover discord from large scale time series more quickly and effectively.

Keywords: Time Series Discord, Anomlay Reasoning, Discord Approximation, Parallel Discord Discovery

目 录

第一章 绪论

本章主要介绍本文研究的背景、动机和创新点。首先陈述时间序列异常检测研究的背景，相关工作以及面临的主要挑战；其次介绍本文的核心创新点；最后列出本文的组织结构

1.1 研究背景

本小节主要介绍时间序列异常的基本概念和背景知识；分析现有时间序列异常检测方法的不足，引出本文的研究对象和问题“超大规模时间序列的异常检测方法”。

1.1.1 时间序列

时间序列是对某个物理量进行等时间间隔的观测所得的一串数值。现实生活中有许多数据都是以时间序列的形式存在的，例如：股票交易的记录、年降雨量记录、气温观测的记录、心电和脑电监测的记录等[1][2]。时间序列涉及的领域非常广泛，包括天文[3]、地理[4]、生物和化学等自然科学领域，也包括语音识别[5]、地震预测[6]和资源勘探等工程技术领域以及金融分析[7]、市场预测和人口统计等社会经济领域。

时间序列的一个固有特征是相邻的观测值之间存在着相互的依赖性，这种相互的依赖性往往蕴含着被观测事物或现象在特定环境或特定时刻的大量信息，研究与分析这种相互的依赖性具有极大的实用价值。时间序列分析正是研究这种观测值之间相互依赖性的一种分析技术，一般通过曲线拟合和参数估计对时间序列建立数学模型进行分析[8]。时间序列分析常应用于国民经济宏观控制、区域发展综合规划、市场潜量预测、地震前兆预报、气象预报、水文预报、农作物病虫灾害预报、环境污染控制、天文学和海洋学等方面。

1.1.2 时间序列异常

异常是数据中的特定模式，这些模式与事先定义的正常模式存在不一致[9]。时间序列异常是时间序列上下文中与正常模式存在不一致的时间子序列。导致时间序列异常的原因有很多，例如：

1. 由于操作失败导致数据在读取、记录、计算时产生错误；
2. 可能是在合并来自不同数据库的数据时导致的错误，因为不同的数据库的度量单位或时间不一致
3. 数据内部结构的特异性也会引起时间序列异常

时间序列是受监测事物状态的具体表现，时间序列中的异常数据往往包含重要的信息，表明受监测事物内部或外部环境存在异常。为了预防与治理这些异常，我们需要使用时间序列异常检测方法检测时间序列中是否存在异常，并寻找有效的方法去检测这些异常所隐含的意义。

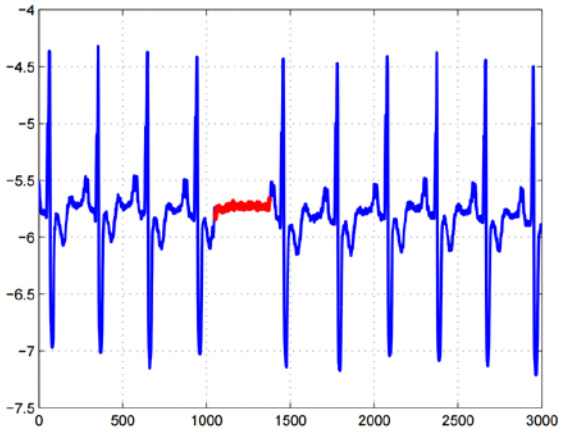

图 1-1. 时间序列异常

Fig. 1-1 Anomalous time series

虽然在异常检测领域有很多工作[9]，但是大部分异常检测方法只用来寻找与其他正常数据个体不同的单个数据个体，并没有考虑数据在时间序列方面的特性，这一类异常也被称为点异常。请看图 1-1 中给出的示例，该图给出了一个病人的心电图。心电图通常应该是近似的周期性时间序列。图中标红的部分是异常，因为相近的低幅值持续出现

的时间很长，该时间跨度与心率周期相比显得异常。值得注意的是，该低幅值在该时间序列中出现过多次，所以幅值本身并不是异常。所以，如果时间序列数据集被当作一系列幅值而忽略在时间序列方面的特性，那么图示红色部分的异常是无法被检测出来的。

时间序列异常的检测问题可以从不同的角度看待，这里我们讨论两种情况以下三种情况：

1. 检测时间序列上下文异常。在这种情况下，异常是时间序列中的单一数据实例。这些异常在时间序列上下文中是异常的，然后仅从幅值的角度来看它们并不是异常。这在统计学领域是一个被广泛关注和研究的问题[11][12][13]，图 1-2 展示了某地区总长为 2 年的月平均气温的例子。1 摄氏度的气温对于冬天（在 t_1 时间点）来说是正常的，但是同样的气温在夏天（在 t_2 时间点）出现则是异常的。该例中第二年 6 月附近的月平均气温

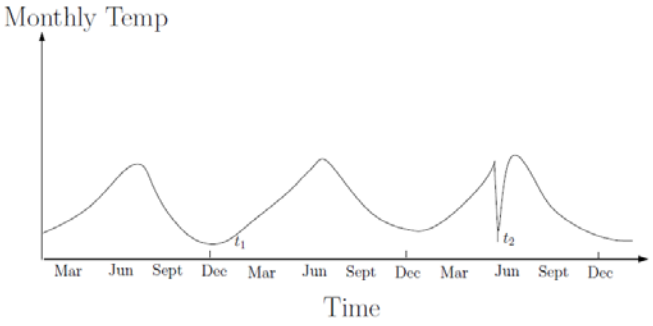

图 1-2. t2 位置的数据点在月平均气温时间序列上下文中是异常

Fig. 1-2 Contextual Anomaly at t2 in monthly temperature time series

2. 在给定的一个时间序列中检测异常的子序列。本问题要求在一段很长的时间序列中找到一小段异常的时间子序列。图 1-1 展示了一整段时间序列中包含一段异常的时间子序列（红色部分）的例子。红色部分因持续时间过长而可以被视为异常，其幅值本身在该时间序列中出现过多次，所以就幅值而言不是异常。该问题与无监督学习环境有关，在训练过程中缺少经过标记的数据，时间序列的绝大部分都被认为是正常的。如果被寻找的时间子序列的长度为 1，则该问题退化为第一类问题。这一类异常检测问题（Discord Discovery）的概念由 Keogh 等人

[15]在 2005 年提出，其定义为： “时间序列异常是时间序列中与其他子序列最不相似的子序列。[14]” 该定义被广泛的接受与研究，它也是本文将要研究的问题。

3. 在给定的时间序列集合中检测异常的序列。给定一个时间序列集合，其中包含很多相互独立的时间序列，这些序列集合中正常的时间序列的比例占到绝大部分，每个序列可以含有或不含有正常或异常的标记。第三类问题试图检测一个时间序列相对于给定的时间序列集合是否异常。图 1-3 展示了第三类问题的例子。右图中的红色时间序列相对于左图中给定的时间序列集合是异常的。虽然这类问题的处理对象也是时间序列，但是在比较过程中将每个时间序列当作单一数据实例看待，任意两个时间序列之间不存在序列关系，因此本文将不对该问题进行讨论。

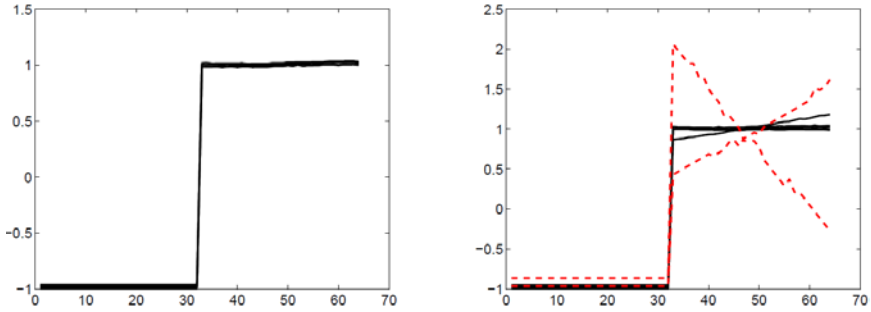

图 1-3. NASA 磁盘缺陷时间序列数据集：时间序列集合中的正常时间序列（左图）以及待检验的时间序列（右图）[16]

Fig. 1-3 Reference (left) and Test (right) time series for the NASA disk defect data [16]

以上三类问题是时间序列异常检测的常见问题，其中第二类问题，是第一类问题在时间序列长度方面的推广，而第三类问题中的时间序列相互之间不存在序列的关系。本文将针对第二类问题，即在给定的一个时间序列中检测异常的子序列，进行着重研究。

1.2 研究动机

本小节首先围绕时间序列异常检测方法工作进行调研，其次基于当前工作分析提炼

关键挑战和细化本文的研究问题。

1.2.1 相关工作

随着信息技术的发展，可监测对象的种类越来越多，采集数据的设备数量越来越大。时间序列的多样性与规模化对时间序列异常检测方法提出了新的要求。现有时间序列异常检测方法主要面临以下两个问题：

1.2.1.1 检测效果问题

现有时间序列异常检测方法在检测效果方面无法适应时间序列多样化的要求。时间序列异常检测方法从数据的角度出发，在没有应用相关专业知识的情况下，寻找时间序列中的异常。此类方法是应用弱相关的检测方法。此类方法对应用的假设较少，适用范围更广。但是此类方法由于缺乏专业知识，方法受到检测性能的限制，检出率低，误报率高，且只能检测出异常在时间序列中的位置，无法对检出异常的原因做出解释。此类方法的检测结果仍需由使用者进行人工检查与辨伪，才可能起到帮助人们了解受监测事物的异常情况的作用。

现有与时间序列异常检测有关的工作从磁盘 I/O[10]、索引方法[19]、参数自动化选择[22]，时间序列近似方法[20][21]等方面对异常检测方法进行优化，使得检测速度有所提升。但是在检测性能方面没有相关的改进工作。

能有效检测出异常以及发现异常的原因是时间序列异常检测方法的先决条件，本文致力于改进现有时间序列异常检测方法的检测效果，在不影响应用弱相关方法的应用范围适应能力的前提下，改善检测方法的检测效果。

1.2.1.2 检测性能问题

现有时间序列异常检测方法无法适应数据规模增长的需求。时间序列的规模分别在两个方面快速增长，即维度和长度。

在现实世界中，无处不在的传感器技术使得许多时间序列以多维时间序列的形式存在着。多维时间序列即以等时间间隔，从多个数据源同步采集数据所得到的序列。例如：一个气象站同时采集温度、湿度、风速、风相、光照强度等多个指标；在监护病房中需

要对同一个病人进行多个指标的监测，包括心率、呼吸、血压、体温等。随着监测指标的种类的增加，时间序列的规模也在成倍的增加。

另外，时间序列的规模也在长度方面不停地增长。以医疗领域为例，如果以 256Hz 的频率全年无间段采样一个病人的心电图，则可获得超过 100M 个采样点；NASA Ames 截止 2012 年累积有数 10T 个美国国内航班的遥感数据，美国田纳西某电力公司每四个月就可产生 10T 个采样数据。金融数据、网络通讯的性能统计信息、个人设备、生产流水线、传感器网络等数据源都在以爆发式的速度产生着时间序列数据。

时间序列规模在不停地增加，而时间序列异常检测方法在应对超大规模时间序列的能力方面较落后。到目前为止，基于内存的检测方法的相关论文使用到的最大的数据集包含 64K 个数据点，而基于硬盘的检测方法相关论文中使用到的最大数据集包含 100M 个子序列[10]。这些方法存在计算复杂度过高、存储空间受限或者硬盘 I/O 耗时的缺点。本文将以提高异常检测方法处理超大规模时间序列的能力作为研究目标。

1.2.2 关键挑战

基于 1.2.1 小节对当前时间序列异常检测工作的调研和分析，有两个主要问题需要解决，即超大规模时间序列异常检测的效果问题与检测性能问题。这些问题可以被细分为如下四个挑战：

1.2.2.1 双胞胎怪胎问题

随着时间序列数据规模的增长，同一个形状的异常在时间序列中出现多次的概率增加，这种现象被称为“双胞胎怪胎”问题(“Twin Freak” Problem)。双胞胎怪胎问题是现有的时间序列异常定义不能捕捉的。在此类的问题中，不寻常的时间子序列由于双胞胎的存在，不能成为“与其他子序列最不相似的子序列”，此类子序列虽然很少见，但是往往无法被识别为异常，有时甚至不会出现在最异常的 k 个子序列的列表中。

1.2.2.2 异常溯源问题

随着传感器种类的增加，时间序列的维度和规模也随之增加。与单维时间序列相比，多维时间序列能对受监测事物进行更细致具体的描述，不同维度可用于描述受观测事物

不同方面的状态与情况，为复杂系统的异常溯源提供了可能。现有检测方法只能发现异常发生的时间，但并不能异常溯源，即发现事物的哪方面存在异常。这不利于从超大规模时间序列中找到更多有意义的信息。

1.2.2.3 并行化方法的效率问题

超大规模时间序列异常检测方法的并行化存在诸多困难，如数据访问效率问题，当数据规模超过内存限制时，访问存储在磁盘上的数据会给检测方法带来性能上的负面影响；如计算资源利用率问题，多个计算节点之间计算结点需要同步，计算节点因同步而闲置，从而降低计算资源的利用率；如通讯效率问题，计算节点之间的通讯使计算资源长时间处理等待状态，因此通讯效率问题也会影响计算资源利用率。

1.2.2.4 并行化方法的正确性问题

时间序列异常检测的计算复杂度随着数据规模的增加而快速增加，串行方法已经不能满足时间序列异常检测在数据规模方面的要求。简单的分治法在对异常检测进行加速时会引入检测结果的“正确性问题“[15]。这意味着我们不能通过常用的分治法改善时间序列异常检测在数据规模方面的可扩展性。

上述四个问题可以概括为检测效果问题、检测性能问题，本文拟针对上述问题开展系统的优化方法设计和实现。

1.3 创新点

针对当面时间序列异常检测方法的效果问题与检测性能问题，本文提出了高效的异常检测优化方法。本文主要有以下四个关键创新点：

1.3.1 异常定义的改进

当单维时间序列中存在多个相似的异常时，这类异常检测问题被称为双胞胎怪胎问题 (Twin Freak Problem) 。根据时间序列异常的定义，即时间序列异常是指一个时间序

列中与其他子序列最不相似的子序列，现有时间序列异常的定义无法捕捉这类异常。本文改进时间序列异常的定义，利用 J 近邻距离 (J^{th} nearest neighbor distance) 作为衡量时间序列异常程度的标准，即考量任意子序列与其第 J 个最邻之间的距离。J 近邻异常为一个时间序列中拥有最大该距离的子序列。本文提到的 J 近邻异常以及相应的异常检测方法改进单维时间序列异常检测方法的检测效果问题以及计算复杂度问题。

1.3.2 多维时间序列异常溯源与降维

实际生活中的许多数据都是以多维时间序列的形式存在的。多维时间序列记录着受观测对象的多个方面的状态与情况，现在现有时间序列异常检测方法只能发现异常发生的时刻，而不能找到异常的具体原因。同时，检测方法的计算时间会随着时间序列维度的增长而增长。降维是检测方法加速的重要手段。我们通过实验观察发现，多维时间序列之间可能存在关联，不同维度对时间序列异常的贡献程度可能不同，而贡献程度最大的维度是异常发生的原因。本文结合多种异常检测方法，从多维时间序列中挑选对异常的贡献最多的维度（异常溯源），然后利用单维时间序列异常检测方法检测异常。

异常溯源具有降维的作用，与异常溯源相比，现有的经典数据降维方法（如 PCA[17]、SVD、FFT）的缺点包括以下两点：1、算法复杂度高；2、产生的低维数据与高维数据之间没有对应关系，对应关系的缺失使后续时间序列分析受到限制。

与现有的数据降维方法相比，通过异常溯源进行降维有以下优点：1、算法复杂度低；2、保留了降维数据与原始数据之间的对应关系，该降维方法为使用者对受监测对象的异常进行溯源提供了一定的参考依据。

1.3.3 并行化的时间序列异常检测

随着时间序列的多样化与规模化的发展，超大规模时间序列异常检测逐步受到单台计算机性能与内存存储方面的限制，而基于磁盘的检测方法也受到磁盘 I/O 操作的性能约束。

本文从定义层面分析并证实时间序列异常检测方法的并行化可行性，并使用并行化计算平台 Apache Spark 实现对时间序列异常检测的加速。本文提出了适用于分布式计算

环境的异常估算方法和并利用剪枝技术降低异常检测的计算复杂度，且利用批处理和多发射的方法改善了计算资源的利用率。

1.3.4 近似的并行化时间序列异常检测

Keogh 等人指出：时间序列异常的第二类检测问题，即从一个很长的时间序列中找到异常的时间子序列，该类问题是不能直接通过分治法进行并行加速的，分治法会使检测结果不正确 [15]。目前还没有与时间序列异常检测方法相关的并行化研究。

本文通过异常的近似检测实现了异常检测的并行化。近似检测方法通过一个验证阶段提高检测结果的正确性。本方法降低了异常检测问题的计算复杂度，且可以通过并行化方法加速检测。本文还提出了自适应停止条件（Adaptive Stop Criteria，ASC），在提高结果正确性的同时降低计算复杂度。

1.4 文章组织结构

本文组织结构：第二章介绍本文使用的基本概念、方法，并对现有时间序列异常检测方法做全面的调研；第三章提出了面向超大规模时间的异常检测系统，介绍系统中各种方法与方案的关系与合作方式；第四、五、六、七章分别针对时间序列异常检测方法的检测效果问题与检测性能问题的挑战设计了不同的优化方案：其中第四章通过改进单维时间序列异常的定义改善异常检测方法的检测效果；第五章结合多种异常检测方法对时间序列异常进行溯源以及数据降维，从而达到改善检测效果与降低计算复杂度的目的；第六章研究并实现了超大规模的时间序列的并行化异常检测方法；第七章研究并提出了近似的并行化异常检测方法；最后，第八章对本文内容进行总结，并针对后续可能的研究进行讨论分析。

1.5 本章小结

本章介绍了本文的研究背景和动机，主要通过相关工作的调研分析发现时间序列异常检测方法的性能与速度问题，并简要介绍本文的创新点和文章的组织结构。

第二章 异常检测的基本方法

本章首先介绍了各种异常检测方法以及优点与局限；其次介绍了本文使用的基本符号、概念和方法，并对现有时间序列异常检测方法进行调研，为第三至第七章的系统、方法与方案阐述提供必需的背景介绍。

2.1 异常检测方法的分类

现有异常检测方法从数据使用的角度可以被分为两类方法：1、模型驱动；2、数据驱动。本文研究的时间序列异常检测方法属于数据驱动方法。为了了解数据驱动方法与模型驱动方法的区别，我们首先简单介绍模型驱动方法与数据驱动方法。

2.1.1 模型驱动的方法

模型驱动的异常检测方法的输入为训练数据集与测试数据集，输出为根据训练数据集建立的模型，以及测试数据中的异常。模型的使用分成训练(Trainning)和测试(Testing)两个阶段，在训练与测试这两个阶段中使用的数据集往往是不同的。模型驱动方法要求训练数据集的内容足够丰富，即能反映受监测对象所有可能的正常的工作状态，训练阶段将正常工作状态所具有的特征提取，形成模型(Model)。而模型驱动方法在测试阶段运用模型检测测试数据集中的异常，在测试阶段中不涉及训练数据集的访问，该特征是模型驱动方法与数据驱动方法的根本区别。模型驱动方法都是监督方法（Supervised method）或半监督方法（Semi-supervised method）。常见的模型驱动的异常检测方法可以归纳为以下五种。

- 在机器学习与认识科学领域，人工神经网络（Artifical Neural Network）是一类模仿生物脑神经网络的统计学习模型神经网络[79]。人工神经网络一般以分类器的形式进行异常检测[80][81][82]。使用神经网络进行异常检测需要分两步操作，第一步需要使用大量的正常数据训练神经网络，第二步是使用神经网络检测测

试数据。如果测试通过，则表明该测试数据为正常数据，否则为异常数据。

- 贝叶斯网络（Bayesian Network）是数据挖掘和机器学习中基本的分类算法，其理论基础都是贝叶斯定理[83]。例如，一个贝叶斯网络可以表达疾病和病症之间的概率关系。给定一定的病症，该网络可用于计算患者患有某一种疾病的概率。简单的说，贝叶斯网络是作为分类器的形式进行异常检测的[84][85][86]。在训练过程中，贝叶斯网络通过学习正常数据与异常，了解特定现象与事物本质之间的关联。在测试过程中，如果测试数据被分类为正常分类的概率很高则可认为是正常数据，否则会被视为异常数据。
- 支持向量机（Support Vector Machine）是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解[87]。SVM 通过分类区分正常数据与异常数据[88][89][90]。SVM 在训练过程中建立正常数据与异常数据之间的分界面，并在测试过程中通过该分界面界定数据的正常与否。
- 基于规则的检测方法通过学习规则捕捉系统的正常行为，如果一个测试数据没有被所学习到的规则覆盖到，那就被基于规则的检测方法视为异常。一个基本的基于规则的检测方法一般分为两个步骤，第一步通过规则学习方法，如RIPPER[91]或决策树[92]，学习训练数据中的规则。基于规则的检测方法必须要求通常学习包含正常与异常数据的训练数据集达到学习异常检测的目的[93][94][95]。每条规则的置信度与训练数据集中符合该规则的数据的比例有关。在测试过程中，基于规则的检测方法为每一个测试数据寻找最为合适的规则，规则的置信度的倒数即为异常的程度。
- 基于统计学的异常检测方法的基本原理就是：异常就是随机模式产生概率非常低的被怀疑是部分或者完全不相关的观察[96]。统计学方法通过将一个统计学模型与给定的数据匹配，并支持统计推论测试方法决定一个从未见过的数据实例是否属于该模型生成的数据。如果一个数据实例不太可能由该模型生成，那么被定义为异常。无论是参数化或是非参数的方法都被运用于生成统计学模型。

参数化方法[97][98]假设已知用于生成数据的统计模型的相关知识，而非参数化方法[99][100]不对生成数据的统计学模型作任何假设。

以上这些方法都属于模型驱动的异常检测方法，训练过程是检测方法从训练数据集中学习受监测事物的本质，训练过程使模型驱动方法拥有较快的检测过程，但同时也为检测方法带来诸多缺点，模型的质量是检测方法有效性的关键。

- 首先，模型驱动方法对受观测事物的本质作出假设，例如某一变量符合高斯分布，或可以通过其他变量进行线性表示，在实际应用过程中这些假设可能会使模型无法准确地描述受监测事物的本质，因此，不合适的模型可能会影响检测方法的有效性。
- 其次，模型驱动方法要求训练数据具有代表性，能够反映受监测事物的各种工作状态。通常训练的收集与标记是不便或昂贵的。有些还要求训练数据中包含所有需要检测的足够数量的异常，这使得模型驱动的异常检测无法检测大型或复杂系统的异常。
- 另外，受监测事物的变化会使事先训练好的模型无法反映事物的现有本质，例如受观测事物的新生本质会被检测方法误判为异常，而较为少见的正常工作状态会被模型判定为异常。为了更好的反映事物的本质，必需重复训练模型以保持检测方法的有效性。

2.1.2 数据驱动的方法

数据驱动方法的输入为测试数据集，输出为测试数据集中最不寻常的数据实例以及它们的异常程度。数据驱动方法仅有测试检测阶段，而没有从数据集中提取特征信息形成模型的阶段。常见的数据驱动的异常检测方法可以被归纳为以下几类:

- 最近邻分析法（Nearest Neighbor）被应用于诸多异常检测方法之中，该方法假设正常数据实例比较常见，会组成密集的簇或团，而异常的数据实例通常远离它们的最近邻[101][102][103]。最近邻距离分析法是数据驱动方法的典型例子，它不需要事先从数据集中建立模型，而是直接通过数据实例之间的关系判断数

据的正常与否。最近邻分析法需要评估两个数据实例相似度或距离的测量方法。通常，基于最近邻分析法的异常检测方法可以被分为，一类使用数据实例之间的绝对距离来计算数据实例的异常程度[104]，一类是使用数据实例的相对密度计算数据的异常程度[52][105]。

- 类聚(Clustering) [101]用于将相似的数据实例归为一类。类聚与异常检测的有着根本的区别，但是请多研究将类聚应用于异常检测方法，并取得了显著的效果。类聚即可以作为非监督学习方法，也可以作为半监督学习方法进行异常检测。类聚从异常定义的角度可以被分为三类，一类假设正常数据通过类聚被识别为不同的分类，异常数据不会被划分到任意一个分类中[106][107]； 第二类假设正常数据聚集成簇，正常数据离簇的中心距离较小，而异常数据离簇中心的距离较大[108][109]；第三类假设正常数据属于规模较大较密集的簇，而异常数据属于规模较小的较为稀疏的簇[110][111]。
- 信息理论技术使用不同的衡量方法，如柯尔莫戈洛夫复杂度[112]、熵[113]和相对熵[114]等，分析数据中信息的含量，这类检测方法假设数据中的异常增加了数据的不规律性，从而增加了数据中信息的含量。他们的优点是不对受检测事物作任何假设，而从数据的角度出发检测异常[115] [117]。这类方法检测异常时，需要从数据集中找到数据量最小信息最大的数据，因此该类检测问题也可以被视为一类帕雷托优化问题，没有唯一的解决方案。
- 频谱分析法尝试用数据集合中较少的属性近似还原出原始数据，找出近似表示与原始数据之间的差异，将这些差异作为异常。频谱分析法对异常作出如下假设：正常数据处于低维空间，而含有异常的数据集合处于高维空间。所以这一类方法的主要工作是决定可以用于容纳正常数据的子空间以及容纳异常所需要的高维空间[116][118][119][120]。这类方法即可以工作在无监督模式下，也可以工作在半监督模式下。

相对于模型驱动的异常检测方法，数据驱动的异常检测方法具有以下优点：

- 数据驱动的方法对数据集不作任何假设，它适合在新生的应用领域与前沿科学

领域在数据集中找出最不寻常的数据实例，而这些实例往往能够引起人类用户的兴趣；

- 数据驱动的方法没有训练过程与模型，因此当受监测事物的本质发生改变时，由于观测数据本身能够反映事物的本质，数据驱动的方法能够通过数据直接感受到这点变化，因此数据驱动的方法更适合于检测复杂多变的系统中的异常。

随着传感器技术的不断发展,时间序列的种类正在增长,生成速度也在不断地加快。新生应用领域的时间序列的应用相关知识尚不完备，而时间序列也随着事物的发展而变化 着。模型驱动的异常检测方法无法适用超大规模时间序列异常检测的要求。本文以数据驱动的异常检测方法为研究重点，着重研究时间序列异常检测的检测性能与效果。

2.2 时间序列分析

时间序列异常检测方法建立在基本时间序列分析方法的基础上，在这一小节中我们将介绍时间序列的定义，时间序列的近似方法，以及时间序列的索引方法。

2.2.1 时间序列定义

时间序列是对某个物理量进行等时间间隔的观测所得的一串数值。 在现实中，许多数据都是以时间序列的形式呈现的，例如：股票交易的记录、气温观测的记录、心电监测的记录等。本文将使用如下定义与符号表达时间序列及相关概念：

定义1. 时间序列(Time series)：*时间序列是一个有序的由 m 个实数组成的组合，表达为* $T = (t_1, t_2, \dots, t_m), t_i \in R$

许多情况下我们更关心一条很长的时间序列中的局部特性，我们将时间序列的局部称为子序列：

定义2. 子序列(Subsequence)：*给定长度为 m 的时间序列 T，子序列 $C_{p,n}$ 是 T 中从位置 p 开始连续采样 n 次获得的子集合。表达为* $C_{p,n} = T[p; n]$。*通常我们能讨论时间序列的异常检测时 n 的值不会改变，所以* $C_{p,n} = T[p; n]$ *可以缩略地表示为* $C_p = T[p]$。

我们可以通过滑动窗获得一条时间序列中的所以可能的子序列：

定义3. 滑动窗(Sliding window)：*给长度为m的时间序列T，对T使用滑动窗可生成所有可能的长度为n的子序列。*

我们还需要比较子序列之间的相似性，我们将子序列之间的相似性定义为距离：

定义4. 时间序列之间的距离/相似度(Time series Distance/Similarity)：*距离dist的输入为两个序列p和q，输出为一个非负实数R，该输出可用于表征p和q的距离或相似度，即$dist(p,q)$。*

时间序列异常检测方法中最常用的距离定义为欧氏距离，给定两个长度都为 n 的时间序列 p 和 q，则 p 和 q 之间的欧氏距离为：

$$dist(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2-1}$$

最近邻(Nearest neighbor)：*我们将C_p的最近邻，记为*$\mathrm{nn}(C_p)$*，定义为与T中的任意子序列C_o，该子序列C_o拥有最小的与C_p之间的距离。*

通常情况下，C_p的最近邻C_o是C_{p+1}或C_{p-1}或其他与C_p存在重叠的子序列，这样的结果对于异常检测方法是无意义的，所以在比较距离和寻找最近邻时应该排除重叠的情况。这种做法在生物信息学领域[29]以及时间序列分析领域[24][25][26][27][28]得到广泛的共识，因为我们需要排除重叠对异常检测方法带来的负面效果。

定义5. 不重叠(Non-overlapping)：*给定两个时间子序列C_p和C_q。如果$|p-q| \geq n$，我们称C_p与C_q不重叠。*

相应的，我们重新对最近邻进行定义：

定义6. 不重叠最近邻(Non-overlapping nearest neighbor)：*我们将C_p的不重叠最近邻，记为不重叠*$\mathrm{nn}(C_p)$*，定义为与T中的任意子序列C_o，该子序列C_o拥有最小的与C_p之间的距离。*

此处最近邻的定义与空间数据中的最近邻定义[23]存在差别。本文中将不考虑重叠情况下的最近邻问题或时间序列之间距离比较问题，为方便起见，本文剩下的部分中将省略修饰词“不重叠”。子序列与其最近邻的距离定义为：

定义7. 不重叠最近邻距离： *C_p的不重叠最近邻距离，记为$mnDist(C_p)$，定义为C_p与$nn(C_p)$之间的距离。*

2.2.2 时间序列的近似

时间序列所包含的数据数量往往很多，直接对原始的时间序列进行异常检测，效率很低。如何降低时间序列的数据规模，同时保有关键信息，并且能够近似度量两条时间序列的相似性，是加快时间序列异常检测的速度的关键所在。因此在异常检测之前，需要对原始的时间序列使用近似方法进行重新表达，即对原始时间序列进行抽象与概括。

时间序列近似的基本思想是保留时间序列的基本形状，忽略具体细节，对时间序列进行压缩表达。时间序列近似表达的目的有二点：（1）近似表达后的长度远远小于原始时间序列的长度，从而降低了时间序列的规模；（2）时间序列异常检测的计算复杂度与时间序列的长度有关，近似表示有助于减少时间序列异常检测的计算复杂度。时间序列近似表达方法有很多，目前常用的方法可以分为以下四类：基于频域的近似表达方法（如DFT[30]和 DWT[31]）、基于分段的近似表达方法、基于符号的近似表达方法和基于奇异值（SVD[32]）的近似表达方法。由于本文只用到了基于符号的近似表达方法，而基于符号的近似表达方法从基于分段的近似表达方法派生而来，所以在此不介绍基于频域和奇异值的近似表达方法。

2.2.2.1 基于分段的近似表达方法

时间序列的分段法用若干条直线线段来近似表达原始的时间序列，是一种数据的近似表达方法，与原始时间序列之间的误差取决于线段的数量。分段表达方法是时间序列近似表达方法中研究最多最早的方法之一。分段表示常用方法有 PLR[33]、PAA[34]、APCA[35]等。由于本文未用到 PLR 和 APCA，在此将仅给出 PAA 的简介。

Lin 等人[36]提出了一个非常简单的近似表达方法，称为 PAA(Piecewise Aggregate Approximation)，PAA 利用滑动窗实现，设定一个大小固定的窗口并在时间序列上滑动，每次窗口滑动步长为窗口的宽度，PAA 利用窗口中数据的均值作为整个窗口内数据的表

达。PAA 方法假设大部分的时间序列在短期内数据变换不大，并假设滑动窗内的均值可以用于近似表达原始的时间序列。

给定一个长度为 m 的时间序列$T = \{t_1, \cdots t_m\}$，PAA 将它转换为一个 w 维向量$\bar{T} = \bar{t}_1, \ldots \bar{t}_w$。其中$\bar{T}$的第 i 个元素$\bar{t}_i$可以使用下列表达式计算：

$$\bar{t}_i = \frac{w}{m} \sum_{j=\frac{m}{w}\times(i-1)+1}^{\frac{m}{w}\times i} t_j \tag{2-2}$$

如果 w 很大并且接近时间序列 T 的长度 m，那么时间序列的近似表示将非常接近原始时间序列。如果 w 很小，那么近似表示将丢失很多细节信息。

假设有两个长度都为 m 的时间序列 S 和 T，它们的 PAA 近似表示$\bar{S}$和$\bar{T}$，PAA 表示的长度都为 w，则它们的 PAA 表示之间的欧式距离可表示为以下公式：

$$dist(\bar{S}, \bar{T}) = \frac{m}{w} \sqrt{\sum_{i=1}^{w} (\bar{s}_i - \bar{t}_i)^2} \tag{2-3}$$

PAA 近似表示的距离是原始时间序列距离的下界，即

$$dist(S, \mathrm{T}) \geq dist(\bar{S}, \bar{T}) \tag{2-4}$$

PAA 的计算以浮点加法为主，伴有少量的浮点除法，因此 PAA 相对于其他各类近似表达方法的优点是计算复杂度低。不过它也存在如下不足：（1）滑动窗的大小是应用相关的；（2）时间序列窗口内的表达采用求均值的方法，会造成极值信息的丢失。

2.2.2.2 基于符号的近似表达方法

基于符号化的近似表达方法的主要目标是将时间序列转换成字符的离散序列。这种转换需要处理时间序列时域的幅值。该方法的主要动机是利用现在的符号化异常检测算法[36][37]。另一个动机是提高计算效率[14][15][38]。不过符号化方法也会造成细节信息的丢失。

最常见的符号化方法是 SAX(Symbolic Aggregate approXimation)[36]，是基于 PAA 开发而来的近似表达方法。该符号化方法假设所有时间序列的幅值分布符合高斯分布，它

输入是一段时间序列，输出为一个字符串。完成转换需要三个步骤：（1）将最小至最大的的幅值范围划分为区间，区间的划分方法需符合等概率的高斯分部划分；（2）为每个区间分配一个字符；（3）使用 PAA 对时间序列进行近似，然后使用（1）和（2）中所述的划分方法将 PAA 的每个均值用字符代替。

SAX 方法引入断点的概念（breakpoint）用于将高斯分布划分成等概率区间，如表 2-1 所示，断点是一串经过排序的数$B = \beta_1, \ldots \beta_{\alpha-1}$，标准正态分布$N(0,1)$曲线下$\beta_i$到$\beta_{i+1}$的面积为$\frac{1}{\alpha}$。

表 2-1 将高斯分布划分成等概率区间的断点的查找表（3 到 5）[36]

Table 2-1 A lookup table that contains the breakpoints that divides a Gaussian distribution in an arbitrary number (from 3 to 5) of equiprobable regions [36]

β_1 \ a	3	4	5
β_1	-0.43	-0.67	-0.84
β_2	0.43	0	-0.25
β_3		0.67	0.25
β_4			0.84

确定断点之后，将时间序列的 PAA 表示成以字母形式出现的字符。以两个断点（或三个等概率区间）为例，如图 2-1 所示为一个长度为 128 的时间序列转换成宽度为$w = 8$的字符种类$\alpha = 3$的 SAX 表示的例子，图的左半部分显示了高斯分布的等概念划分区间，右半部分显示了转换过程。时间序列在图中显示为灰色的曲线，它首先被转换成 PAA 表示，以水平线段表示。然后 PAA 表示被转换成字符。例如所有小于较小断点的 PAA 系数都被替换成 a，所有介于两个断点之间的 PAA 系数被替换为 b，所有大于较大断点的 PAA 系统被替换成 c，以此类推。最终，该时间序列被转换成 SAX 表示 cbccbaab。

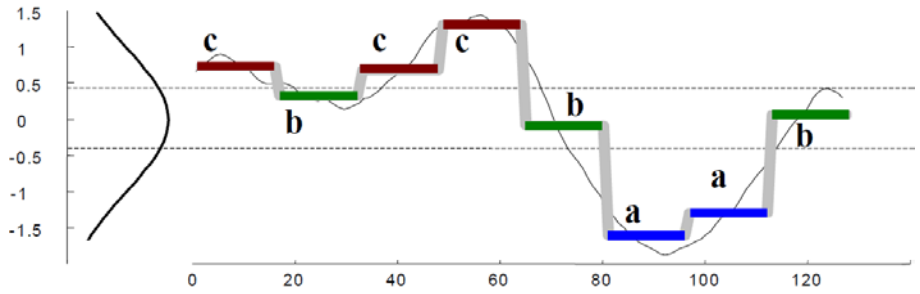

图 2-1. SAX 表示的例子，$n = 128, w = 8, \alpha = 3$ [36]

Fig. 2-1 Example of SAX representation [36]

SAX 表示的子序列之间的距离是原始时间序列之间欧氏距离的下界。设 S 和 T 为两个长度都为 m 的时间序列，它们的 SAX 表示分别记为$\tilde{S} = \tilde{s}_1, \ldots \tilde{s}_w$和$\tilde{T} = \tilde{t}_1, \ldots \tilde{t}_w$，其中 w 为 SAX 表示的长度，$\tilde{s}_i$和$\tilde{t}_i$是符号。则 SAX 距离可以通过如下公式计算获得：

$$dist(\tilde{S}, \tilde{T}) = \frac{m}{w}\sqrt{\sum_{i=1}^{w}(\tilde{s}_i - \tilde{t}_i)^2} \tag{2-5}$$

其中，$(\tilde{s}_i - \tilde{t}_i)$的值可以通过查寻断点表获得，以三个断点（四个等概率区间）为例，如表 2-2 所示，相同的字符（如 a 与 a）和相邻的字符（如 b 与 c）之间的最小欧氏距离为 0，而其他距离可从表 2-1 中查得。

表 2-2 SAX 四区间距离查找表

Table 2-2 A SAX distance lookup table for a=4

	a	b	c	d
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

2.2.3 时间序列的索引

时间序列数据库包含的数据量通常非常庞大。为了提高查找的效率需要对被查找的

时间序列建立索引。索引技术的关键问题是如何划分数据空间，以及如何根据划分方法将数据组织起来。

索引的组织很大程度上依赖于相似性度量，即时间序列之间的距离。欧氏距离是最常见的时间序列相似性度量方法。本文将主要讨论与欧氏距离有关的时间序列索引。

对时间序列进行索引一般需要经过两个步骤，即近似与索引。

（1）首先对时间序列进行近似表达。由于时间序列数据库的规模通常非常巨大，且时间序列分析经常获取数个形态相似的时间序列，所以直接对每一个时间序列进行索引在计算及存储资源方面是缺乏效率的，也是不必要的。为提高效率，需要使用时间序列近似表达方法对时间序列进行降维表达，使形态相似的时间序列可以作为一组时间序列被索引，同时降低“维度灾难”发生的可能性。

（2）随后，使用现有的索引方式对经过降维表达的时间序列进行索引。索引为时间序列分析提供最近邻搜索、范围搜索等服务。假设给定一个时间序列集合$\mathbb{T} = \{T_1, T_2, \dots\}$，一个实数 r 与一个待查询时间序列 V，最近邻搜索问题要求在$\mathbb{T}$中寻找与 V 最相似（即距离最小）的时间序列，而范围搜索要求在$\mathbb{T}$中寻找与 V 足够相似（最大距离不超过 r）的时间序列。常见的索引方式可以分为空间索引与前缀索引两类。

（a）空间索引是指依据空间对象的位置和形状或空间对象之间的某种空间关系，按一定顺序排列的一种数据结构，其中包含空间对象的概要信息。从大的方面分，空间数据索引技术可分为树结构（包括 R 树[39]、K-D 树、四叉树）和网格文件两类。时间序列索引的一种方法就是直接采用这些空间索引技术[30]。

（b）时间序列的前缀树索引是特指适用于符号化近似表达方法的索引方式。前缀树将拥有相同前缀的近似表达组织在同一节点下，以达到将相似时间序列组织在一起的目的。使用前缀树的动机是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的时间序列之间的距离的比较。不同的符号近似方法可以组织出不同的前缀树[40][41]，本文将主要介绍具有代表性的 Trie 索引[15]以及 iSAX 索引[42][43]。

2.2.3.1 Trie 索引

时间序列异常前缀树 Trie 是由 Keogh 等人[15]提出的专用于时间序列的索引方法。Trie 首先要求把时间序列转换成 SAX 近似表示，然后把拥有相同前缀的 SAX 表示组织在同一节点下，从而达到把相似时间序列组织在一起进行索引的目的。图 2-2 展示了 Trie 的一个例子。

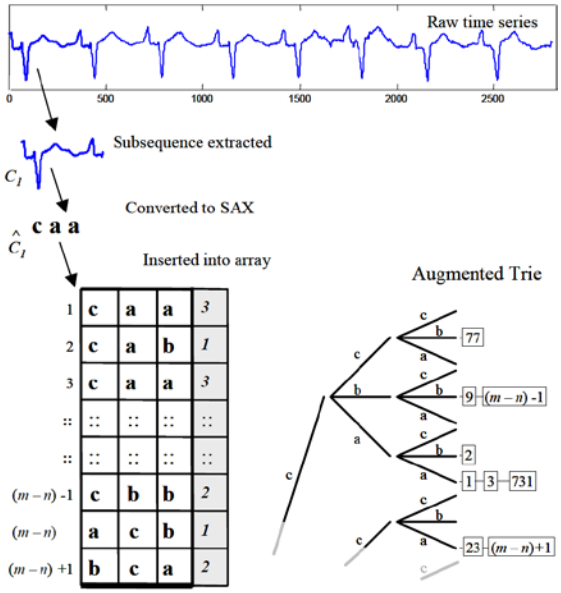

图 2-2. 时间序列前缀树的例子[15]

Fig. 2-2 A simple example of time series trie index [15]

如图 2-2 的上半部分所示为一个时间序列，通过滑动窗从时间序列中获取所有可能的子序列，并转换成 SAX 表示，其中$w = 3$，$\alpha = 3$。图的左下角为转换后所得的结果。编号为 1 的序列被转移成 caa，编号为 2 的序列被转移成 cab，以此类推。1 号与 3 号序列拥有相同的 SAX 表示，所以把它们放置在前缀树的同一叶节点，即 caa 下，如图右下角所示；编号 1、3 与编号 2 的时间序列拥有相同的 SAX 表示前缀 ca，所以把它们放置在前缀树的 2 级节点 ca 之下。以此类推。

最终可以得到如图 2-2 右下角所示的前缀树。树的根节点不包含任何字母，一级节点列举所有可能的 1 个字母，二级节点列举所有 2 个字母的排列组合，叶节点存放所有 3 个字母的排列组合。所有能被转移成某 3 个字母 SAX 表示的时间子序列的编号都会

被存放在这 3 个字母所代表的叶节点内。

Trie 并不提供精确的最近邻搜索与范围搜索功能，但是 Trie 可以统计出每一个 SAX 近似表达分别代表哪些原始时间序列数据，这为最近邻搜索问题缩小了搜索范围，同时也为时间序列异常检测提供了便利。此外，由于 Trie 的叶节点之间不存在重叠（R-tree 存在严重的叶节点相互重叠问题），因此 Trie 的存储需求更小，在遍历时 Trie 时的计算复杂度也更小。

2.2.3.2 iSAX 索引

Shieh 等人在 SAX 符号表达方法基础上提出一种新的多分辨率符号化表达方法以及相应的索引方法 [42] 。该方法它对 SAX 进行改进，将符号由字母替换为二进制数。每个二进制数的比特位宽可以随着需要达到的分辨率层次而动态地改变。iSAX 索引相对于 Trie 在性能方面也有了进一步改进，包括提供距离下界以及最近邻搜索方法的实现。

改进后的时间序列基于 SAX 表示的分割后得到的字符串使用多分辨率二进制数表达每一个字符，并可记为$SAX(T,w,\alpha)=\tilde{T}^{\alpha}=\{\tilde{t}_1,\dots\tilde{t}_w\}$。图 2-3 展示改进后的 SAX 近似表达方法的例子：

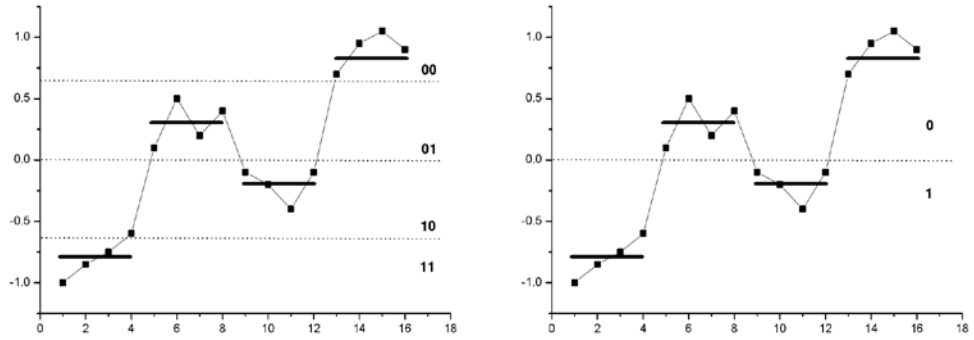

图 2-3. 一个时间序列被转换成使用二进制数表示的 SAX 表示

Fig. 2-3 A time series converted into SAX words

如图 2-3 所示，时间序列 T 可被 SAX 断点分割法（如表 2-1 所示）转换为$SAX(T,4,4)=\tilde{T}^4=\{11,01,10,00\}$，也可表示为$SAX(T,4,2)=\tilde{T}^2=\{1,0,1,0\}$。其中，$\tilde{T}^2$可以通过$\tilde{T}^4$推算获得，即二进制串 11、10 与二进制串 1 有共同的前缀，在从$\tilde{T}^4$推算至$\tilde{T}^2$的过程中，可以把 11、10 替换为 1；同样的，可以把 01、00 替换成 0。下面给出从

高分辨率 SAX 表示推算至低分辨率 SAX 表示的更多例子：

表 2-3 从高分辨率 SAX 表示推算至低分辨率 SAX 表示[42]

Table 2-3 It is possible to obtain a reduced cardinality SAX word simply by ignoring tailing bits[42]

$$SAX(T,4,16) = \tilde{T}^{16} = \{\ 1100,\ 1101,\ 0110,\ 0001\ \}$$
$$SAX(T,4,8) = \tilde{T}^{8} = \{\ 110,\ 110,\ 011,\ 000\ \}$$
$$SAX(T,4,4) = \tilde{T}^{4} = \{\ 11,\ 11,\ 01,\ 00\ \}$$
$$SAX(T,4,2) = \tilde{T}^{2} = \{\ 1,\ 1,\ 0,\ 0\ \}$$

如表 2-3 所示，可以通过舍弃高分辨率 SAX 表示的每个符号的末尾位，从而获得低分辨率 SAX 表示。为了简化 SAX 表示，将二进制数写为十进制数并在其右上角加注α的大小，以表 2-3 的第一行为例，该 SAX 表示可记为$\{12^{16}, 13^{16}, 6^{16}, 1^{16}\}$。SAX 表示之间的距离可由查表（如表 2-2 所示）得到。Shieh 等人还定义了 PAA 表示以及 SAX 表示之间的距离，该距离表达式可用于快速获取两个时间序列之间的距离下界[42]：

$$MINDIST\_PAA\_SAX\left(\tilde{S}_{SAX}, \tilde{T}_{PAA}\right) = \frac{m}{w}\sqrt{\sum_{i=1}^{w}\begin{cases}(\beta_{Li} - \tilde{t}_i)^2\ if\ \beta_{Li} > \tilde{t}_i \\ (\beta_{Ui} - \tilde{t}_i)^2\ if\ \beta_{Ui} < \tilde{t}_i \\ 0\ otherwise\end{cases}} \quad (2\text{-}6)$$

其中β_{Li}和β_{Ui}分别表示某一 SAX 符号区间的下界与上界。

由改进后的 SAX 表示可以看到，不同的时间序列最终可以被转换为互斥的 SAX 表示，而相似的时间序列可以被转换成相同或相近的 SAX 表示，最后通过特定的组织方式将这些 SAX 表示根据相似性(即距离)组织起来形成索引。

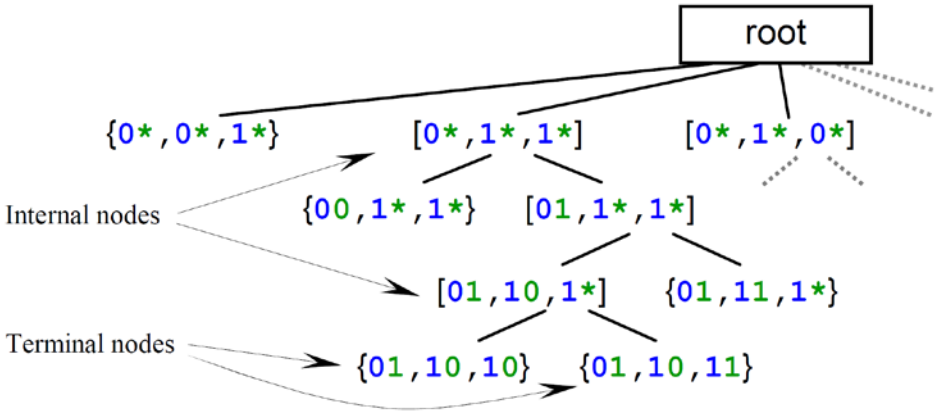

图 2-4. iSAX 索引的例子[42]

Fig. 2-4 Illustration of iSAX index [42]

如图 2-4 所示，iSAX 索引也是以树的形式存在的，建立一个 iSAX 索引需要确定字符串长度$w, w \in N^+, w > 1$，还需要确定字符基数α，$\alpha = 2^c, c \in N^+$。树中包含根节点(root)、内部节点(Internal nodes)与叶节点(Terminal nodes)。每叶节点代表一个在本索引中分辨率最高即$\alpha = 2^c$的 SAX 表示，如图中的叶节点{01,10,10}。叶节点包含着所有能被近似表达成该 SAX 表示的时间序列的指针。内部节点代表一个较低分辨率的 SAX 表示，其 SAX 表示可以通过舍弃其子节点的每个字符的末尾位获得，如图中的子节点{01,10,10}与其父节点{01,10,1 ∗}，以该 SAX 表示为前缀的所有内部节点都为该内部节点的子节点。根节点包含着整个 SAX 表示空间。

iSAX 索引的精确最近邻搜索可以通过从根节点进行广度搜索，找到与待查询时间序列最相似的时间序列，精确的范围搜索也可以通过类似的方式获得。通过 PAA 与 SAX 表示之间的下界距离(2-5)可以快速舍弃索引数据结构中与待查询时间序列相距较远的内部节点或叶节点，从而批量舍弃不必要的原始时间序列之间的距离的计算。最终 iSAX 需要少量的原始时间序列之间的距离计算来确定最近邻之间的排序。精确而快速的最近邻搜索与范围为包括异常检测在内的时间序列分析问题提供了便利。

2.3 时间序列异常

异常是数据中与正常定义不符合的模式 [9]。图 2-5 展示了二维数据集中异常的简单例子。这个数据集中有两个正常的数据区域，即N_1和N_2。大部分的数据都落在这两个区域内。距离这两个区域足够远的点，如o_1、o_2或O_3中的点都是异常。其中O_3是异常点的集合。

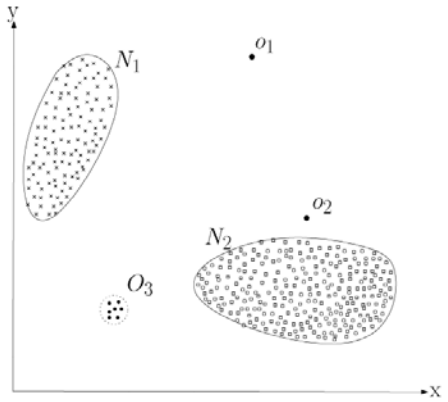

图 2-5. 二维数据集中异常的简单例子 [9]

Fig. 2-5 A simple example of anomalies in a two-dimensional data set [9]

本文要研究的时间序列异常与广义异常定义很相似，是在时间序列领域的特例。时间序列异常的本质是与其他的子序列显著不同的子序列，因而异常检测可以辨别出一条时间序列中最不寻常的子部分。时间序列异常检测对数据挖掘有很多用途，例如改进聚类、数据清洗、数据概要、异常检测等操作的质量。为了能够找出时间序列中的异常，Keogh 等人 [15] 给出了时间序列异常的形式定义。

定义8. 时间序列异常(Time series discord)： *给定时间序列T，C_d是T中起始位置为d，长度为n的子序列，如果C_d拥有最大的最近邻距离，那么C_d是T中的异常。即对于T的任意子序列C_o,$|d-o| \geq n$，都有$nnDist(C_d) > nnDist(C_o)$。*

有时我们需要关注时间序列中的多个异常，所以我们可以把第 k 个异常定义为：

定义9. 第 k 个时间序列异常(k^{th} time series discord)： *给定时间序列T，C_d是*

T 中起始位置为d，长度为n 的子序列，C_{d_i}是T 的第i 个时间序列异常，$1 \leq i < k$。如果C_d拥有第k 个最大的最近邻距离，且与第i 个时间序列异常没有重叠，即$|d_i - d| \geq n$，那么C_d是T 中的第k 个时间序列异常。

2.4 时间序列异常检测方法

根据时间序列异常的定义，我们可以设计寻找该异常的基本方法：检查每一个可能的子序列，寻找每个子序列的最近邻并计算它与最近邻之间的距离，最后找出拥有最大该距离的子序列。该方法需要遍历所有的子序列，所以需要一个循环，循环的每次迭代需要寻找当前子序列的最近邻，所以还需要一个循环。综上所述，寻找时间序列异常的基本方法需要一个双层循环。

本文将时间序列异常的基本检测方法称为 Naïve 方法，其伪代码描述如表 2-4 所示：

Naïve 方法的外层循环如表 2-4 第 2 行所示，顺序遍历所有的子序列。对于当前子序列，一个内层循环，如第 4 行所示，通过遍历所有子序列从而找到当前子序列的最近邻距离。通过不断更新当前异常子序列，如第 9 行所示，Naïve 方法最终可以找到该时间序列的异常。

表 2-4 为寻找一个时间序列异常的基本方法，寻 k 个异常的方法与此类似。在寻找第 k 个异常时，在循环中跳过前 k-1 个异常存在重叠的时间子序列，双层循环结束时就能找到第 k 个异常。因此，寻找 k 个异常需要执行 k 次表 2-4 中的双层循环。

表 2-4 时间序列异常的基本检测方法

Table 2-4 Naive discord discovery method

输入：长为 m 的时间序列 T，滑动窗的宽度为 n

输出：时间序列异常的起始位置 bsf_loc 及其最近邻距离 bsf_dist

```
1   bsf_loc=-1; bsf_dist=0;    //初始化
2   for p in 0 to m-n  //外层循环
3       nnDist=infinity;  //最近邻距离初始化为无穷
4       for q in 0 to m-n  //内层循环
5           if (|p − q| < n) continue; //忽略重叠情况
6           dist = dist(C_p, C_q)  //计算当前子序列对的距离
7           if (nnDist>dist) nnDist=dist;   //更新当前子序列的最近邻距离
8       end for
9       if (bsf_dist<nnDist)   //更新当前时间序列异常
10          bsf_dist = nnDist;
11          bsf_loc = p;
12  end for
```

Naïve 方法需要计算任意两个子序列之间的距离，假设时间序列 T 的长度为 m，则 Naïve 方法的计算复杂度为$O(m^2)$。Naïve 方法的计算复杂度过高，研究者就如何加速时间序列异常的检测进行了许多探究。

2.4.1 HOTSAX

HOTSAX 是与时间序列异常的定义一起被提出的时间时序异常检测方法，HOTSAX 方法与 Naive 方法的原理基本相同，它使用了 Trie 索引和剪枝方法大幅度减少了异常检测的计算复杂度。许多时间序列异常检测方法通过改进 HOTSAX 得来的，所以我们有必要仔细剖析 HOTSAX 方法的原理。HOTSAX 方法的伪代码如表 2-5 所示。

表 2-5 HOTSAX 方法

Table 2-5 HOTSAX method

输入：长为 m 的时间序列 T，滑动窗的宽度为 n，启发式外层循环与内层循环	
输出：时间序列异常的起始位置 bsf_loc 及其最近邻距离 bsf_dist	
1	bsf_loc=-1; bsf_dist=0; //初始化
2	for p in 0 to m-n ordered by Heuristic outer order //启发式外层循环
3	nnDist=infinity; //最近邻距离初始化为无穷
4	for q in 0 to m-n ordered by Heuristic inner order //启发式内层循环
5	if ($\|p-q\| <$ n) continue; //忽略重叠情况
6	$dist = dist(C_p, C_q)$ //计算当前子序列对的距离
7	if (nnDist>dist) nnDist=dist; //更新当前子序列的最近邻距离
8	if (nnDist<bsf_dist) break; //提前放弃正常子序列的相关计算
9	end for
10	if (bsf_dist<nnDist) //更新当前时间序列异常
11	bsf_dist = nnDist;
12	bsf_loc = p;
13	end for

剪枝方法，如第 8 行所示，旨在省去正常时间子序列最近邻搜索的相关计算。在异常检测的过程中，我们并不需要找到所有时间子序列的最近邻。若能为当前时间子序列找到一个足够近的邻居，则说明该子序列肯定不是异常，便可放弃当前子序列的最近邻搜索。该方法如第 8 行所示，称为剪枝技术（Early abandon technique），HOTSAX 方法使用了剪枝技术从而减少了很多不必要的距离计算。

Trie 索引，如第 2 行与第 4 行所示，旨在提供启发式的访问时间子序列的顺序，使得剪枝再加频繁地发生，从而降低计算复杂度。启发式排序建立在以下两个灵感的基础上：

（1）若在外层循环的第一次迭代就能访问到真正的时间序列异常，那么外层循环的

第一次迭代结束时 bsf_dist 将大于剩余子序列与其最近邻之间的距离。此后的每一次外层循环迭代只需要一次内层循环迭代便可通过剪枝方法提前结束，时间序列异常检测方法的计算复杂度将由$O(m^2)$降为$O(m)$。由观察可知，在将时间序列使用 SAX 方法近似表达时，同一 SAX 表示出现的次数越多，说明与该 SAX 表示相似的时间子序列越多，这些时间子序列越不可能是异常，反之，如果整个时间序列中有一个子序列被近似成独有的 SAX 表示，则说明该子序列是异常的可能性较高。HOTSAX 利用 Trie 计算出每种 SAX 表示在时间序列中出现的频率，并在外层循环中按升序访问 SAX 表示所对应的时间了序列，从而近似达到尽可能在外层循环初期就访问到真正时间序列异常的目的；

（2）若在内层循环的第一次迭代就能访问到当前子序列的最近邻，则将更加频繁地触发剪枝方法，从而减少计算复杂度。能被近似成相同 SAX 表示的时间子序列是较相似的子序列。HOTSAX 利用 Trie 找出与当前子序列相似的子序列，使得内层循环能更早找到当前子序列的最近邻，以达到频繁触发剪枝方法和减少计算复杂度的目的。

HOTSAX 方法相较于 Naive 方法在调用距离函数（表 2-5 的第 6 行）的次数方面降低了 3 个数量级。HOTSAX 的成功主要在于通过 Trie 索引和剪枝方法省去了很多计算。然而 HOTSAX 本身仍然存在一些弊端，后续研究尝试通过在各个方面的改进消除或缓解这些弊端。

2.4.2 索引的改进

索引能够为时间序列异常检测方法提供启发式的时间子序列访问顺序，启发式的排序使得剪枝方法被更频繁地触发，从而减少异常检测的计算时间。HOTSAX 使用了以 SAX 为近似表达方式的 Trie 索引，在此之后研究者们就寻找更好的索引做出了努力。

Fu 等人提出了 WAT（Wavelet Augmented Trie）索引 [44][45] ，该索引结合了小波变换与前缀树，利用哈尔小波变换与符号排序的方法对原始时间序列进行排序。在将时间序列转换为哈尔近似之后，罕见的哈尔表示将出现在靠近索引根节点的位置，他们使用广度优先的遍历法遍历 WAT 索引，更早访问罕见的哈尔表示，从而达到对原始时间序列进行启发式排序的目的。

Buu 等人 [19] 利用 iSAX 索引[42][43]来进行时间序列异常检测。他们针对 iSAX 索引提出了被称为 HOTiSAX 算法，该算法使用到了两个新的附加函数，用于在 iSAX 索引中寻找近似的和精确的非重叠最近邻。除此之外，他们还提出了新的启发式排序方法，通过对 iSAX 表示的排序，进一步减少了调用距离函数的次数，从而降低了异常检测的计算复杂度。

Li 等人 [20] 提出了比特近似表达方法及其相应的索引方法。 比特近似方法的步骤是：首先将时间序列近似为 PAA 表示，然后比较相邻两个 PAA 分段，根据大小关系，使用 1 和 0 表示相邻 PAA 分段的变化趋势，随后根据时间顺序将这些比特位排列成比特字符串，这就是比特近似表达方法。在获得比特近似表示之后，他们使用 K-Medoids 类聚算法[50]对比特表示进行分类以获得对时间序列的索引，该索引被他们称为比特类聚(BitCluster)。在该索引中，出现频率多的比特表示以及相近的比特表示会被归为一类，而罕见的比特表示会被单独归为一类。比特类聚因此为时间序列异常检测方法提供了支持相似性查询以及罕见程度查询的时间序列索引。

2.4.3 参数优化设置

时间序列异常的定义涉及一个参数，即滑动窗的大小 n。除此之外，HOTSAX 方法 [18] 中的 Trie 索引使用了额外的两个参数，即 SAX 表示的长度 w 和字符基数α。不适当的参数设置不但会对时间序列异常检测的性能产生负面影响，还会对检测速度产生负面影响。研究者试图找出优化的参数设置，或设计无需参数的检测方法，从而改善异常检测方法的速度。

Ameen 和 Basha 等人[46][47]运用经典的统计决策和时间序列数据挖掘工具客观地决定最合适的滑动窗大小，从而使频繁模式（frequenct pattern）与异常能被更快更有效的找到。该参数优化方法减少与最佳滑动窗长度倍数的计算时间。Fu 等人[45]提出了哈尔近似表示方法，由于该近似方法的运用，检测方法只需要从哈尔前缀树的根节点开始广度遍历，哈尔表示的字符串长度会随着深度的增加而增加，所以检测不需要在开始时决定哈尔表示的长度，而是在检测的过程中动态的决定所需要的长度。Luo 等人[48]提

出了 DDS(Direct Discord Search)的算法用于寻找周期性或伪周期性时间序列中的 k 个异常，这种检测方法不需要事先建立索引，所以不需要调整索引的参数。DDS 是基于复原图(recurrence plot)[51]开发的，复原图可用于诊断和可视化时间序列。对于周期或伪周期的时间序列，与 HOTSAX 相比，DDS 能用更少的计算时间找到时间序列中的异常。Luo 等人[22]随后通过对 DDS 做了改进提出了 GDS(General Direct Search)。与 DDS 相比，GDS 的异常检测方法无需假设对时间序列的周期性。GDS 将周期性假设替换为一个新的参考函数，该参考函数采用新的采样策略从而使 GDS 检测方法的速度比 DDS 更快，鲁棒性更好。

2.4.4 硬盘感知的异常检测

大部分现有的时间序列异常检测方法都假设数据的规模小到足以存放在一台计算机的内存中，在读取数据时采用随机读取。而在实际生活中，时间序列的规模往往超出了单机内存的存储能力，即数据必需存放磁盘中。随机读取磁盘效率低下，严重影响到异常检测的速度。为了解决磁盘 I/O 问题，Yankov 等人[10]提出了一个两阶段异常检测方法。磁盘感知的检测方法将时间序列异常检测转换为异常的范围检测，即找到时间序列中最近邻距离大于 r 的所有序列，其中 r 为阈值。当 r 足够大时，从时间序列集合中筛选出期望数量的序列，以达到检测出最不寻常的 k 个序列的目的。

该算法的第一阶段找出候选时间序列，第二阶段从候选中找到异常。该算法需要对磁盘进行两次线性扫描。与以前随机读取的检测方法相比，该算法可以处理的时间序列数据规模高出许多数量级。

给定一个阈值 r，算法的第一阶段用于生成一个时间序列异常的候选集合，集合中的伪异常的数量较少。第一阶段的形式定义可以用表 2-6 描述。

表 2-6 磁盘感知异常检测方法的候选选择阶段[10]

Table 2-6 The candidates selection phase of disk aware discord discovery method [10]

输入：	$\mathbb{C}$ ：存储在磁盘上的所有时间序列，r ：异常范围
输出：	$\mathbb{C}_c$异常列表
1	$\mathbb{C}_c = \{C_1\}$
2	For $i = 2$ to $\lvert\mathbb{C}\rvert$ do
3	isCandidate = true
4	For $\forall C_j \in \mathbb{C}_c$ do
5	If $(\mathrm{dist}(C_i, C_j) < r)$ then
6	$\mathbb{C}_c = \mathbb{C}_c \backslash C_j$
7	isCandidate = false
8	End if
9	End for
10	If (isCandidate) then
11	$\mathbb{C}_c = \mathbb{C}_c \cup C_i$
12	End if
13	End for

第一阶段线性扫描整个数据集，对于每一个时间序列C_p，第一阶段验证在$\mathbb{C}_c$中的所有序列为异常的可能性（第 5 行）。如果一个候选没有通过验证，那么就把它从候选集合中去除（第 6 第 7 行）。最后，当前如果当前时间序列C_p可能是一个异常，就把它加到候选集合中（第 11 行），否则把它忽略。

硬盘感知算法的第二阶段是异常候选列表的提炼阶段。该阶段的输入包括第一阶段输出的异常候选列表。列表中包括所有最近邻距离大于 r 的序列，同时可能也包括少量最近邻距离小于 r 的序列。如果确实存在最近邻距离小于 r 的序列，那么提炼阶段会把它去除。

表 2-7 磁盘感知异常检测方法的异常提炼阶段[10]

Table 2-7 The discord refinement phase of disk aware discord discovery method [10]

输入：	$\mathbb{C}$ ：存储在磁盘上的所有时间序列，$\mathbb{C}_c$：候选时间序列异常，r ：异常范围
输出：	$\mathbb{C}_c$异常列表，$\mathbb{C}_c.\mathrm{dist}$异常距离列表

```
1   For j = 1 to |C_c| do
2       C_c.dist_j = ∞
3   End for
4   For ∀C_p ∈ ℂ do
5       For ∀C_j ∈ C_c do
6           If C_i == C_j then
7               continue
8           End if
9           d = EarlyAbandon(C_i, C_j, C_j.dist)
10          If (d < r) then
11              C_c = C_c\C_j
12              C_c.dist = C_c.dist\C_j.dist
13          Else
14              C_c.dist = min(C_c.dist, d)
15          End if
16      End for
17  End for
```

该阶段首先将异常候选列表$\mathbb{C}_c$中的所有序列的最近邻距离设置为正无穷大（第 2 行），随后扫描整个时间序列集合，以计算异常候选列表$\mathbb{C}_c$中的所有序列的最近邻距。实际距离的计算使用了优化的过程，可以使用上界提前中止当前的距离计算（第 9 行）。

根据对最近邻的计算，每个时间序列C_p存在以下三种情况：1. 异常候选序列$\mathrm{C_j}$与磁盘上的序列$\mathrm{C_i}$的距离大于$\mathrm{C_j.dist}$ ，如果出现该情况，那么我们什么都不做；2. 如果$\mathrm{C_j}$与

C_i之间的距离小于 r，那行意味着C_j肯定不是异常，我们可以把C_j从$\mathbb{C}_c$中去掉（第 11 行和第 12 行）；3. 如果C_j与C_i之间的距离小于$C_j.dist$，但是仍然大于 r，那么我们可以更新C_j的最近邻距离（第 14 行）。

我们可以直观地认识到，第一阶段执行完后，所有最近邻距离大于 r 的异常都会在$\mathbb{C}_c$中，当第二阶段执行时，所有到最近邻距离大于 r 的异常都会被保留在$\mathbb{C}_c$中。算法的计算复杂度主要与$|\mathbb{C}_c|$有关，当$\mathbb{C}_c = \mathbb{C}$时，异常检测方法退化为基本检测方法。显然，这样的集合$\mathbb{C}_c$可以通过将 r 设置为 0 生成。然而，当$\mathbb{C}_c$中只含有一个时间序列，那行第二阶段的计算复杂度将会是线性的。当$\mathbb{C}_c$中的时间序列的数量为 2 个或三个时，计算时间也不会快速增加，主要原因是磁盘读写的时间比计算时间更长。

通过设置足够大的 r 可以使$|\mathbb{C}_c|$足够小，然而过大的 r 会使检测方法无法检测到指定数量的异常。参数 r 的设置是算法成功的关键。通过研究分析数据集中每个时候序列的最近邻距离分布可以发现，“采样法”可以在消耗较少计算和存储资源的前提下获得比较精准的对异常最近邻距离的估计。

“采样法”可以被描述为：从时间序列集合$\mathbb{C}$中随机抽取少量的序列，形成新的集合$\mathbb{C}_s$，检测$\mathbb{C}_s$中的前 k 个异常。第 k 个异常的最近邻距离即为对于原始集合$\mathbb{C}$中第 k 个异常最近邻距离的估计 r。

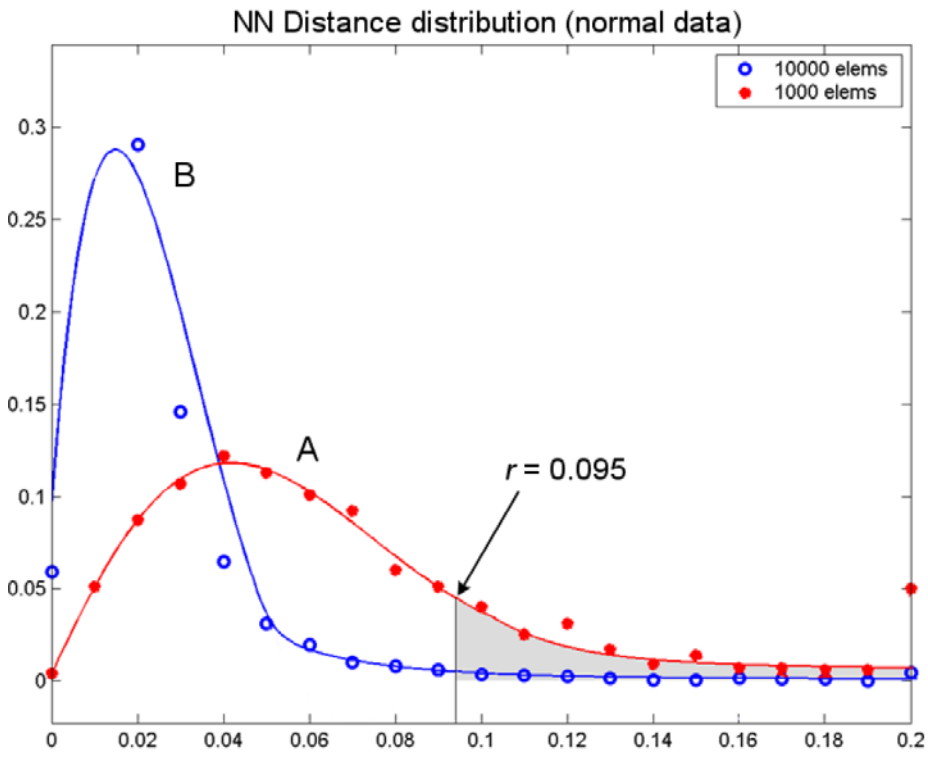

图 2-6. 最近邻距离和采样率的关系[10]

Fig. 2-6 Points sampled from the same normal distribution produce different nearest neighbor distance distributions. The mean and the volume of the tail cut by r decrease with adding more data [10]

[10]给出了采样比例也估算精确度之间的关系，给定时间序列数据集，其中的第个序列的长度为 2，它们可以被看作是 2 维空间中的点，这些点服从正态分布。图 2-6 展示了数据集规模为$|\mathbb{C}| = 10^3$和$|\mathbb{C}| = 10^4$时的概率密度函数。随着数据集规模的增加，最近邻距离的分布的均值向平移，同等采样率下最近邻距离会降低。随着采样数量的增加，最近邻距离会降低。当采样率为 1%左右时，最近邻距离可以增多到 0.095。因此该估算方法适合于估算超大规模时间序列集合的序列最邻距离。

由于$|\mathbb{C}_s|$小到足以存放在内存中，因此随后可以使用传统的基本于内存的检测方法进行检测$\mathbb{C}_s$中的 k 个异常。以$|\mathbb{C}| = 10^4$为例，只需要计算$|\mathbb{C}_s| = 10^2$个时间序列中的异常最近邻距离即可获得对原始时间序列异常最近邻距离的精确估算。

2.5 本章小结

本章首先介绍了各种异常检测方法以及优点与局限；其中数据驱动的异常检测方法可以适应面超超大规模时间序列异常检测的需求。其次本章介绍了本文使用的基本符号、概念和方法，并对现有时间序列异常检测方法进行调研，为第三至第七章的数据驱动的异常检测系统、方法与方案的阐述提供必需的背景介绍。

第三章 面向超大规模时间序列异常检测系统的结构与框架

本章介绍面向超大规模时间序列的异常检测系统的设计思想，简单描述系统中的各种方法与方案，描述方法之间的相互关系，以及系统中各种方法与方案之间的作用。

3.1 异常检测系统的设计思想

为了应对由超大规模时间序列为异常检测带来的检测效果与检测性能方面的挑战，本文提出面向超大规模时间序列的异常检测系统。该检测系统的设计思想有如下四点。

- 通过改进异常定义改善检测效果

异常检测的效果取决于对于时间序列异常的定义，与检测性能无关。以双胞胎怪胎问题 (Twin Freak Problem)为例，根据原始时间序列异常的定义，即时间序列异常是指一个时间序列中与其他子序列最不相似的子序列，如果当前子序列有相似的邻居，即使只与一个邻居相似，被不会被识别为异常。现有的相关异常检测工作只是从检测方法复杂度的角度改进检测方法，检测方法复杂度方面的改进对于捕捉异常的有效性没有帮助。我们应该从定义的角度出发，寻找既有可操作性，又能捕捉更多实际异常的定义。

本文提出一种适用于时间序列数据的异常定义，该定义可以与面向超大规模时间序列异常检测系统中的多种检测方法相结合，使用该异常定义的检测方法都能在检测效果上得到提升，且在检测性能上不会受到明显的影响。本文也提出适用新定义的检测方法，以展示该定义在异常检测在改善检测效果方法的作用。

- 通过结合多种异常定义改善检测效果

单个异常定义的检测效果有限：现有时间序列异常的定义（如 discord），只适用于检测异常发生的时候，无法发现异常发生的原因；而传统点异常定义（如 LOF），可以对来自于不同维度的传感器数据异常作出响应，实现异常的溯源，且计算复杂度低，但是不能用于检测时间序列异常。单独使用这些异常定义进行检测会受到异常定义的局限性。我们应该利用其他异常定义的优势，在系统中结合不同异常定义检测结果，从而进一步

改善面向超大规模时间序列异常检测的检测效果。

本文提出一种能够结合多种异常定义的检测方案，该方案应能够结合本异常检测系统中的多种检测方法，如点异常与时间序列异常。本系统通过结合多种异常定义的检测结果，同时实现异常的检测与溯源，从而提升异常检测效果。

- 通过近似与降维改善检测性能

时间序列异常检测方法的用途受到计算复杂度的制约。根据时间序列异常的定义，检测方法的计算复杂度与数据规模的平方成正比。由于异常检测的许多异常检测应用领域对于检测的准确性要求并不高，适当降低异常检测的准确性可大幅度降低时间序列异常检测的计算复杂度。多维时间序列可用于描述受监测事物的多个方面状态与情况，然而随着维度的增长，多时间序列异常异常检测的计算时间会快速增加。通过观察发现当时间序列存在与异常发生原因无关的维度时，去除无关维度不会对异常检测的准确性产生决定性影响。

本文提出利用近似与降维提高面向超大规模时间序列异常检测系统的检测性能的方法。通过检测性能与检测结果准确性之间的权衡，适应不同应用领域对于检测性能与检测结果准确性的需求。本文将首先使用降维的方法，通过筛选找到对异常程度贡献最大的维度，达到减少数据规模的目的。降维方法是对输入数据的预处理，可与其他任何时间序列异常检测方法结合，具有较好的灵活性。本文其次将利用近似的方法，通过检测每个时间序列分段中的异常得到近似的全局异常，通过近似方法实现异常检测的并行化计算，进一步提面向高超大规模时间序列异常检测系统的检测性能。

- 通过并行化改善检测性能

精确的时间序列异常检测的计算时间与数据规模的平方成正比。超大规模时间序列对于检测方法提出了存储能力与计算能力的新要求。单个计算结点的存储能力与计算能力无法满足这些要求，我们需要利用并行化计算的方法改善异常检测方法的检测性能。

本文提出一种可以用并行化的时间序列异常检测方法，其检测结果完全符合异常定义。该检测方法可以驾驭多个计算节点的存储能力与计算能力，具有良好的通讯效率与计算资源利用率。该方法可以与任何时间序列异常定义结合，同时实现检测性能的提升

与检测效果的改善，也通过与降维方法结合，进一步提升异常检测方法的检测性能。

3.2 异常检测系统的组成

如图 3-1 所示，面向超大规模时间序列的异常检测系统主要由以下四部分结合而成：

- J 距离时间序列异常定义及其检测方法（J-distance Discord，JDD），该异常定义与检测方法的相关论文已在数据挖掘领域的国际会议 ICDM2015 上发表[121]；
- 多维时间序列异常检测方案（Multi-dimensional Discord Discovery Scheme，MDD），该异常检测方案的相关论文已在计算软件应用领域的国际会议 COMPSAC2013 和计算机系统结构领域的国际期刊 FGCS（影响因子: 2.786）上发表[67][122]；
- 并行化的时间序列异常检测方法（Parallel Discord Discovery，PDD），该章节的相关论文已经被数据挖掘领域的国际知名会议 PAKDD 2016 录用，该章节中互连感知的并行化可行性分析方法已在高性能计算领域的国际期刊 Supercomputing（影响因子: 0.858）上发表[73]。
- 近似的并行化时间序列异常检测方法（Approximated Parallel Discord Discovery，APDD），该近似检测方法的串行化版本已在高性能计算领域的国际会议 HPCC2015 上发表[123]；

JDD 与 MDD 方法主要用于改善异常检测方法的检测效果问题。JDD 对时间序列异常重新定义，改善了异常检测中的“双胞胎怪胎”问题；MDD 结合多种异常检测方法，通过异常溯源发现异常发生的原因。JDD 和 MDD 能够从超大规模时间序列中找到最有意义的数据数据实例，找到对应用最有帮助的信息。

APDD 与 PDD 主要用于改善异常检测方法的检测性能问题，APDD 通过近似化方法减少异常检测的计算复杂度，使异常检测可以被并行化加速；PDD 通过并行化方法对异常检测进行加速。同时，MDD 方法又可以通过数据降维的方式减少时间序列异常检

测的计算时间。APDD、PDD 和 MDD 有效改善了超大规模时间序列异常检测的性能问题。

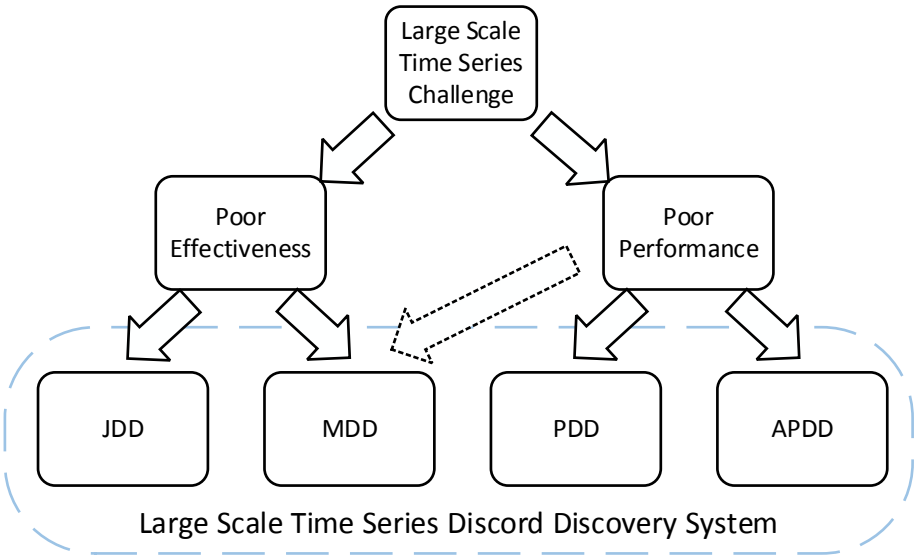

图 3-1. 面向超大规模时间序列的异常检测系统的组成。

Fig. 3-1 The composition of the large time series discord discovery system.

3.3 异常检测系统的结构与框架

面向超大规模时间序列的异常检测系统结合 JDD、MDD、PDD、APDD，达到同时改善检测效果与检测性能的目的，如图 3-2 所示。JDD、MDD、PDD、APDD 以系统组件的形式存在，它们之间不是相互独立的，它们通过相互合作实现异常检测系统。

JDD 是改善超大规模时间序列的异常检测效果的基本方法(kernel function)，MDD、APDD 和 PDD 方法都能通过与 JDD 结合达到改善检测效果的目的。例如，MDD 与 JDD 定义和检测方法结合，可以快速检测多维时间序列中的“双胞胎怪胎”异常。APDD/PDD 可以也 JDD 结合，如图 3-2 的左半边所示，在每个计算结点中，APDD 或 PDD 使用 JDD 定义及检测方法代替传统的异常定义与检测方法，可以快速检测超大规模时间序列中的“双胞胎怪胎”异常。

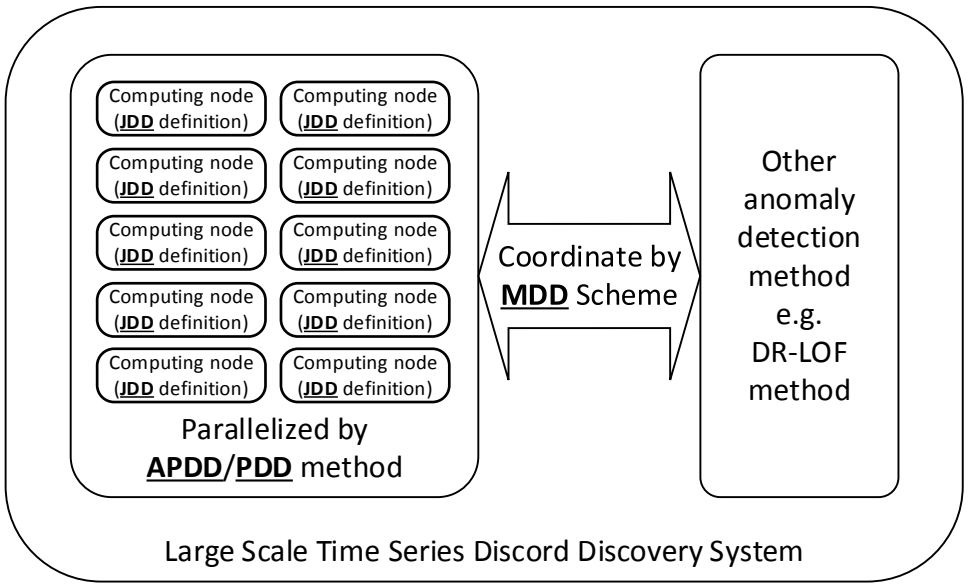

图 3-2. 面向超大规模时间序列异常检测系统的结构

Fig. 3-2 The structure of of the large time series discord discovery system

MDD 是将多种异常检测方法结合的方案。MDD 结合传统时间序列异常检测方法，通过异常溯源改善异常检测效果，通过数据降维改善异常检测性能。MDD 结合 JDD 定义及检测方法与其他检测方法，可以通过改善在“双胞胎怪胎”问题上的检测效果。MDD 结合 APDD 或 PDD 与其他检测方法，可以通过并行化方法减少计算时间，进一步改善检测性能。

APDD 通过近似与并行化的方法改善面向超大规模时间序列的异常检测性能，PDD 通过并行化也改善了异常检测的性能。APDD 与 PDD 是可以互换的方法，当对精确度要求不高时可以选择 APDD，当要求检测结果的绝对正确性时可以选择 PDD。PDD 与 APDD 可以与 JDD 结合，即在每一个计算节点中使用 JDD 定义与检测方法，从而改善异常检测效果。

总之，面向超大规模时间序列的异常检测系统通过结合 JDD、MDD、PDD 和 APDD 改善异常检测的效果与性能。系统结合 JDD 与 PDD 或 APDD，达到改善检测效果与性能的目的，系统通过 MDD 结合 PDD 或 APDD 与其他异常检测方法，通过异常溯源改善检测效果的目的，通过数据降维达到改善检测性能的目的。

3.4 本章小结

本章介绍面向超大规模时间序列的异常检测系统，异常检测系统由 JDD、MDD、PDD 和 APDD 结合而成，本章简单描述系统中的各种方法与方案，描述方法之间的相互关系，以及系统中各种方法与方案之间的作用。系统中的各方法与方案不是独立存在的，它们通过相互结合达到同时改善超大规模时间序列异常检测中检测效果与检测性能问题的目的。

接下来，本文将分章节详细描述 JDD、MDD、PDD 和 APDD 的原理和实现方法。

第四章 J 距离异常检测

本章主要针对时间序列异常(Discord)定义的缺陷，通过改进时间序列异常的定义以及改进异常检测方法，从而提高时间序列异常检测方法的检测效果。

4.1 时间序列定义缺陷

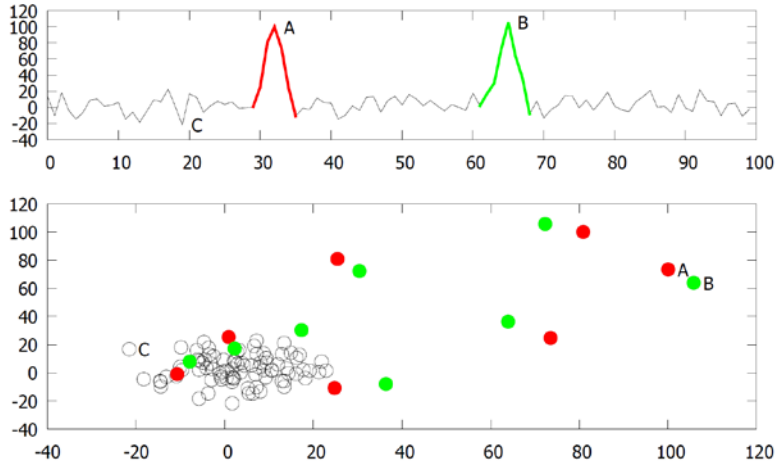

图 4-1. 时间序列中双胞胎怪胎问题的例子。上图为一个人工合成的时间序列，其中包含两很形状相似的异常。下图为宽度为 2 的所有子序列在二维空间中的点，两个异常分别以红色与绿色在上下图中标注出

Fig. 4-1 Upper: Synthetic example of time series discord discovery in the case of multiple occurrence of an anomalous or interesting pattern. The lower part plot all subsequences of length two on a 2D figure and color the anomalies correspondingly

时间序列异常的定义可以用来捕捉时间序列中的异常或有趣的模式。然而在有些时候，同一个形状的异常可能会在时间序列中出现多次，这种现象被称为“双胞胎怪胎”问题(“Twin Freak” Problem)。双胞胎怪胎问题是现有的时间序列异常定义不能捕捉的。在此类的问题中，不寻常的时间子序列由于双胞胎的存在，不能成为“与其他子序列最不相似的子序列”，此类子序列虽然很少见，但是往往无法被识别为异常，有时甚至不会

出现在最异常的 k 个子序列的列表中。图 4-1 展示了一个人工合成的例子以说明时间序列异常定义的缺陷。

图 4-1 的上半部分为一个符合高斯分布的时间序列，其中有两个形状相似的异常，这些异常分别以绿色和红色高亮标出。图的下半部分是所有长度为 2 的子序列在二维空间中的点，与异常存在重叠的子序列使用红色或绿色高亮标出。

从图 4-1 的上半部分可以看出，子序列 A 与其他的子序列存在较大差异，所以直观地说它应该是一个异常。但是根据时间序列异常的定义，子序列 A 不是异常，因为它有一个非常相似的最近邻子序列 B。如图 4-1 的下半部分所示，A 和 B 是非常靠近的。由于同样的原因，根据时间序列异常的定义，B 也不是异常。除此之外，从直觉判断子序列 C 应该是正常的子序列，但根据时间序列异常的定义，C 将在 k 异常的列表中拥有非常高的排名，因为它与最近邻之间的距离比 A 的和 B 的都要近。

就我们所知，目前与时间序列异常检测相关的算法主要从计算复杂度方面对检测进行改进，而没有从是否能出检测的角度对异常检测进行改进。我们相信后者的主要困难在于，解决双胞胎怪胎问题需要为每个子序列求解第 J 个最近邻问题，该问题将引入额外的计算复杂度。当时间子序列之间存在重叠时，求解第 J 个最近邻问题将引入更高的计算复杂度。超高的计算复杂度将降低时间序列异常检测方法的效用。

本章将提出新的时间序列异常的定义，用于缓解双胞胎怪胎问题。同时也将提出新的检测方法，用于缓解新定义引用的额外的计算复杂度。

4.2 J 距离异常（J-distance discord）

如上一节所示，当有两个或多个形状相似的异常出现时，这些异常可能不会被识别为现有时间序列异常定义中的 k 个最异常的子序列。我们认为时间序列异常检测结果的质量与异常检测的计算复杂度一样重要。我们的目标是使这些形状相似的异常也能出现在 k 个异常列表中，换句话说，我们的目标是提高异常检测结果的质量。由此我们需要对时间序列异常重新定义。

4.2.1 J 距离异常的概念

在原始的时间序列异常的定义（定义 8）中，一个时间序列的异常与否取决于它与不重叠最近邻之间的距离。一个异常在时间序列中多次出现时，这些异常是相互的最近邻。形状相似的异常之间的距离可能很小，这使得它们不能被原有的时间序列异常定义所捕获。新的异常定义应该能感知多个相似的异常的存在。

我们从局部异常因子(Local Outlier Factor, LOF[52])算法获得灵感，并提出了 J 距离异常(J-distance Discord, JDD)。J 距离(J-distance)是一个时间子序列与其第 J 个不重叠最近邻之间的距离。一个正常的子序列至少有 J 个与其非常相似的邻居。以 J 距离作为判断标准的异常是时间序列中拥有最大 J 距离的子序列。

4.2.2 形式定义

在给出 J 近邻异常(JDD)定义之前，我们首先给出 J 近邻的相关定义。

定义10. 第 J 个不重叠最近邻(J^{th} nearest neighbor)： *对于任何整数$J > 1$, $1 \le i < J$，假设C_{o_i}是C_p的第 i 个最近邻。C_p的第 J 个最近邻，记为$Jnn(C_p)$，是与C_p距离最近且不重叠的任何子序列C_o，且C_o与C_p的前$J-1$个最近邻没有重叠，即$|o-p| \ge n$，且当$1 \le i < J$，满足$|o-o_i| \ge n$。*

我们对子序列与其第 J 个最近邻之间的距离感兴趣，我们将该距离定义为：

定义11. J 距离(J-distance)： *对于任何正整数 J，子序列C_p的 J 距离，记为$Jdist(C_p)$，是C_p与$Jnn(C_p)$之间的距离。*

J 距离异常建立在 J 距离定义的基础之上，我们将 J 距离异常定义为：

定义12. J 距离异常(J-distance Discord, JDD)： *给定一个时间序列 T，C_d是从位置 d 开始的长度为 n 的子序列，若C_d是 T 的 J 距离异常，则C_d拥有最大的 J 距离。即，对于 T 的任意子序列C_o，$|d-o| \ge n$，且$Jdist(C_d) \ge Jdist(C_o)$。*

我们可能会对时间序列中的k个最异常的子序列感兴趣，我们将第k个异常定义为：

定义13. 第 k 个 J 距离异常(k^{th} JDD)： *给定一个时间序列 T，$1 \le i < k, C_{d_i}$是 T 的第 i 个异常。C_d是从位置 d 开始的长度为 n 的子序列，若C_d是 T 的第 k 个 J 距离异常，那么C_d拥有第 k 个最大的 J 距离，且与前$k-1$个异常不重叠，即$|d-d_i| \ge$*

n。

到目前为止我们省略了对距离的定义，这是为了使异常的定义变得更通用。为了下文的具体化与可操作性，我们在下文中使用无处不在的欧氏距离，其定义如下：

定义14. 时间子序列的欧氏距离(Euclidean Distance)： *给定两个时间子序列C_p和C_q，它们的起始位置分别为p和q，长度都为n，那么它们之间的欧氏距离定义为：*

$$dist(\mathrm{C}_p, \mathrm{C}_q) = \sqrt{\sum_{i=0}^{n-1} (t_{p+i} - t_{q+i})^2} \tag{4-1}$$

比较幅值与均值不同的时间序列是没有意义的[8]。我们在调用距离函数之前会对每个子序列进行归一化处理，使其拥有 0 均值与 1 的标准差。

4.2.3 JDD 的性质

本章中，我们使用 J 距离作为判断时间序列异常的标准。原始的时间序列异常的定义是新的异常定义的一个特例，即$\mathrm{J} = 1$。J 距离标准允许我们发现时间序列中多次出现的形状相似的异常，但也引入了额外的计算复杂度。在时间序列异常检测过程中，我们的 JDD 定义与原始定义有以下两点不同：

1. 相对于原始异常定义中的最近邻距离，J 距离标准要求寻找 J 个最近邻，增加了额外的$\mathrm{J} - 1$次距离函数的调用。为了排除最近邻搜索过程中的子序列重叠情况，需要增加额外的调用距离函数的次数。
2. 对于具有对称性质的距离函数(例如定义 14)，若子序列C_o是子序列C_p的不重叠最近邻，则C_p与C_o之间的距离一定大于或等于的C_o与其最近邻之间的距离，即$\mathrm{nnDist}(\mathrm{C}_p) \geq \mathrm{nnDist}(\mathrm{C}_o)$。当$\mathrm{C}_p$被识别为正常模式时，它的最近邻$\mathrm{C}_o$也一定是正常的。对于 J 距离异常检测来说，我们无法对C_p的 J 个最近邻做正常与否的假设。

这些性质说明现在有的异常检测方法对于寻找 J 近邻异常没有帮助。这也就是我们要在下一节提出 J 近邻异常检测方法的动机。

4.3 JDD 检测方法

首先，我们开发一个基本的 JDD 检测方法，我们取出所有可能的子序列，找到它们的 J 距离，拥有最大该距离的子序列便是 J 距离异常。这个基本搜索方法是通过双层循环实现的，外层循环用于遍历所有的候选子序列，内层循环用于搜索当前候选子序列的第 J 个最近邻。表 4-1 展示了 JDD 的基本检测方法。

表 4-1 JDD 的基本检测方法

Table 4-1 Basic J-distance discord discovery

输入：长为 m 的时间序列 T，滑动窗的宽度为 n，启发式外层循环与内层循环

输出：时间序列异常的起始位置$bsfPos$及其最近邻距离$bsfDist$

```
1   bsfPos=null; bsfDist=0;     //初始化
2   for p in 0 to m-n ordered by Heuristic outer order     //启发式外层循环
3       Jdist*(C_p)=infinity;  //最近邻距离初始化为无穷
4       for q in 0 to m-n ordered by Heuristic inner order     //启发式内层循环
5           if (|p − q| < n) continue; //忽略重叠情况
6           dist = dist(C_p, C_q)  //计算当前子序列对的距离
7           Update Jdist*(C_p) with dist;    //更新C_p的最近邻距离
8           if (Jdist*(C_p)< bsfDist) break;    //提前放弃C_p的相关计算
9       end for
10      if (bsfDist <Jdist*(C_p))      //更新当前时间序列异常
11          bsfDist = Jdist*(C_p);
12          bsfPos = p;
13  end for
```

T 是一条时间序列。n 是滑动窗的宽度。J 是我们引入的用于计算 J 距离的参数 J。启发式外层循环与内层循环是由 Trie 索引提供的优化的子序列访问次序（如章节 2.4.1 所示）。$Jdist$和p是待检测异常的 J 距离与起始位置。在内层循环中，我们需要不断计算

当前候选子序列C_p与不同子序列之间的距离以寻找 J 距离，如第 6 和第 7 行所示。

$Jdist^*(C_p)$是内层循环过程中到目前为止找到的C_p的当前的 J 距离。为了计算$Jdist^*(C_p)$，需要完成以下步骤：（1）保留一个列表用于保存当前子序列C_p与任意C_q的距离；（2）当新的$dist(C_p, C_q)$被计算时，将C_q与$dist(C_p, C_q)$记录到列表中，根据列表中的当前内容找到$Jdist^*(C_p)$；（3）舍弃列表中小于$Jdist^*(C_p)$的所有C_q及其距离$dist(C_p, C_q)$，从而节省存储空间。

在内层循环中，我们可以通过剪枝技术减少计算。我们未必要找到每个候选子序列的真正的 J 距离。当我们发现当前候选子序列C_p拥有比$bsfDist$更小的$Jdist^*(C_p)$时，当前候选子序列一定不是异常，我们可以通过第 8 行提前放弃寻找当前子序列的 J 距离。剪枝技术的效用取决于访问子序列的次序，若外层循环在前几次能访问到异常，内层循环的前几次序列能访问到最近邻，则剪枝技术能起到很好的效用。我们应用了 2.4.1 提到的启发式外层循环（第 2 行）、启发式内层循环（第 4 行），从而发挥剪枝技术的效用。

4.3.2 加强的剪枝技术

因为 J 距离标准引入了额外的计算复杂度，我们开发了加强的剪枝技术用于缓解额外的计算压力。加强的剪枝技术是建立在以下的观察上的。

观察 1. 一个与正常子序列相似的子序列很可能也是一个正常的子序列。

这个想法可以被翻译为以下的规则：给定一个子序列C_p和它的不重叠的第 J 个最近邻C_q，如果$J = 1$，那么我们可知，$Jdist(C_q) \leq Jdist(C_p)$。但是这一特征在$J > 1$时并不成立。我们需要对以上推论进行改进使其能被应用于 JDD 检测方法：

推论 1. 给定一个子序列C_p和它的不重叠的第 J 个最近邻C_q，在使用对称距离的情况下，对于任何正整数 J，有$Jdist(C_q) \leq 2 \times Jdist(C_p)$

证明过程请见附录。有了C_p的 J 距离上界以后，我们可以使用该推论改进 JDD 的基

本检测方法。

表 4-2 增强的 JDD 检测方法

Table 4-2 Enhanced JDD discovery method

输入：长为 m 的时间序列 T，滑动窗的宽度为 n，启发式外层循环与内层循环

输出：时间序列异常的起始位置 bsf_loc 及其最近邻距离 bsf_dist

```
1  bsf_loc=-1; bsf_dist=0;    //初始化
2  for p in 0 to m-n ordered by Heuristic outer order     //启发式外层循环
3      if C_p marked as normal, continue;
4      Jdist*(C_p)=infinity;  //最近邻距离初始化为无穷
5      for q in 0 to m-n ordered by Heuristic inner order     //启发式内层循环
6          if (|p − q| < n) continue; //忽略重叠情况
7          dist = dist(C_p, C_q)   //计算当前子序列对的距离
8          if (Jdist*(C_p)>dist)  Jdist*(C_p)=dist;   //更新C_p的最近邻距离
9          if (Jdist*(C_q)>dist)  Jdist*(C_q)=dist;   //更新C_q的最近邻距离
10         if (2 × Jdist*(C_p)<bsf_dist)
11             Mark  N_J(C_p)  as normal        //放弃C_p的 J 个最近邻的相关计算
12         if (2 × Jdist*(C_q)<bsf_dist)
13             Mark  N_J(C_q)  as normal        //放弃C_q的 J 个最近邻的相关计算
14         if (Jdist*(C_p)<bsf_dist) break;      //提前放弃正常子序列相关计算
15     end for
16     if (bsf_dist<Jdist*(C_p))   //更新当前时间序列异常
17         bsf_dist =  Jdist*(C_p);
18         bsf_loc = p;
19 end for
```

增强的 JDD 检测方法如表 4-2 所示，其中$N_J(C_p)$表示C_p的 J 个最近邻。在内循环寻

找当前候选子序列的J距离时，当发现当前子序列C_p的当前J距离$Jdist^*(C_p)$小于bsf_dist的一半时，根据推论 1我们可以肯定C_p的J个最近邻肯定都不是异常。所以我们可以安全地将C_p的 J 个最近邻标记为正常子序列（第 10 和 11 行），并在今后的外循环迭代时忽略它们（第 3 行）。同样，我们也可以对C_q的 J 个最近邻做相同的判断（第 12 和 13 行）。

4.3.3 重用中间结果

通过观察我们发现，当距离具有对称性时，保留双层循环的中间计算结果可以从以下两个方面节省 JDD 检测的计算量：（1）计算C_p与C_q之间的距离不但对寻找Jdist(C_p)有帮助，同时也对寻找Jdist(C_q)有帮助；（2）在检测第 k 个异常的过程中，计算C_p的当前J距离$Jdist^*(C_p)$对减少检测第 k+1 个异常的计算量有帮助。所以我们通过使用备忘录技术[53]继续减少由 J 近邻搜索带来的额外计算复杂度。

我们维护一个数据结构用存放所有子序列的当前 J 距离$Jdist^*(C_p)$。在计算C_p与C_q之间的距离后，我们不但更新$Jdist^*(C_p)$，同时也更新$Jdist^*(C_q)$（如表 4-2 第 8 和 9 行所示）。更新过程将舍弃大于$Jdist^*(C_p)$或$Jdist^*(C_q)$的距离信息从而降低检测方法对存储的需求。最后我们使用剪枝技术跳过拥有足够小 J 距离的正常子序列。（第 3 和 14 行）

4.3.4 检测 k 个 JDD

为了寻找时间序列中的 k 个异常，我们检测第一个异常、第二个异常、……、第 k 个异常时需要重复同样的步骤。在检测第 i 个异常时，$1 \le i \le k-1$，记录和更新每个候选子序列C_p的当前 J 距离$Jdist^*(C_p)$。随着检测的进行，$Jdist^*(C_p)$会逐步逼近真实的 J 距离，JDD 检测方法将这些计算结果保存下来，在下一个异常的检测过程中重用这些中间结果，从而检测下一个异常所需要的计算量。

4.4 JDD 及其检测方法的经验评估

本章节中，将 JDD 及其检测方法应用于一个合成数据集以及三个真实世界数据集，从而对 JDD 及其检测方法进行经验评估。我们从两方面比较 JDD 检测方法与经典的时间序列异常检测方法，即 HOTSAX：（1）检测结果的质量；（2）计算复杂度。我们也会经验评估 JDD 及其检测方法对于参数 J 的敏感程度。如不明确指出，我们以$J = 3$作为默认的实验设置。我们使用经典的时间序列异常检方法 HOTSAX 作为基准比较方法主要由于两个原因：（1）现有的时间序列异常检测方法建立在同一异常定义的基础上，选用任意方法将得到相同的检测结果；（2）大多数文献的实验都以 HOTSAX 为基准比较方法，为了更好的可比性，我们也与 HOTSAX 进行比较。

检测率（Detection Rate, DR）与误报率（False Alarm Rate, FAR）是衡量异常检测方法有效性的通用指标。然而我们我们决定不 DR 或 FAR 来评价时间序列异常定义的优劣，主要原因有两个：（1）DR 与 FAR 不反映异常的异常程度。（2）时间序列异常能够捕捉异常含义，但并不完全等同于应用相关异常的定义。我们的目标是把最异常的子序列得到异常排名列表中的较高排名。

4.4.1 合成数据集中的异常检测

为了快速展示 JDD 的含义，我们首先评估 JDD 在一个合成数据集上的性能。如图 4-2 上图所示是一个合成数据集，该时间序列中包含一个非稳态的正弦波，周期为 60。我们在位置 70、133 和 761 人工注入了三个长度为 20 的异常，并用红色粗线将异常标出。这些异常切断了正弦波的波峰并且看起来形状非常相似。

我们使用 JDD 检测方法和 HOTSAX 检测合成数据集中的异常。一般来说，滑动窗的长度可以被设置为与正弦波周期一致的长度。然而在某些应用场景中我们并不知道合适的滑动窗长度。因此我们使用范围从 20 至 100，间隔为 20 的不同长度滑动窗进行异常检测。我们分别用 JDD 检测方法与 HOTSAX 检测该合成数据集中的前三个异常。

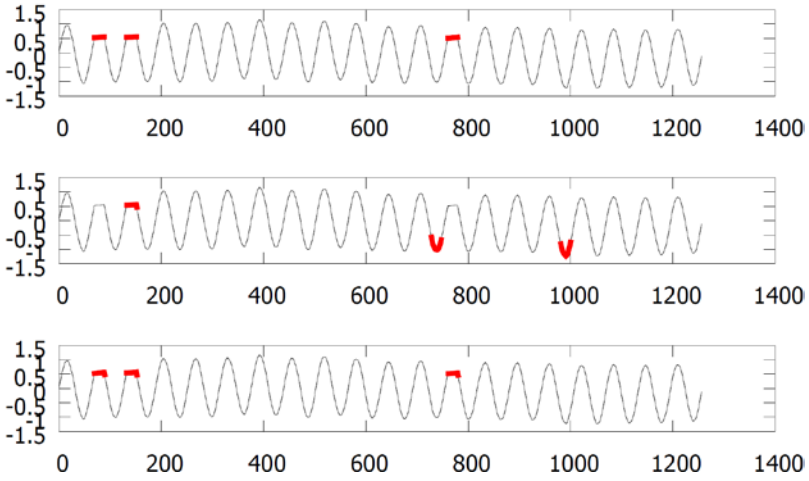

图 4-2. 上图：向正弦波中注入三个形状相似的异常形成的合成数据集，异常使用红色粗线标出。中图与下图是原始时间序列异常定义与 JDD 定义检测出的前 3 个异常。滑动窗的宽度为 20。

Fig. 4-2 Upper: A Synthetic dataset generated by manually injecting three similar anomalies into a non-stationary sine wave. The anomalies are plotted in red. The middle and the lower parts respectively show the top three discords discovered by HOTSAX and JDD method. The size of sliding window is 20.

图 4-2 的中图与下图分别展示了 HOTSAX 与 JDD 检测方法检测出的时间序列中的前 3 个异常。这些检测结果都用红色的粗线标出。为了使图简洁一些，我们只标出了滑动窗长度为 20 时的结果。从中图我们可以看出，HOTSAX 只找出了三个异常中的一个。两个正弦波的波峰被错误地识别为异常，这主要是由于该合成数据集的非平稳特性。下图为 JDD 检测方法的结果。相比之下，JDD 将三个异常全部检测出来。因此，当滑动窗设置为长度 20 时，JDD 的检测效果超过了 HOTSAX。

表 4-3 列出了不同滑动窗长度情况下 HOTSAX 和 JDD 在合成数据集上的检测结果。它显示了两种方法所检测出的异常的个数以及检测结果中每个异常的起始位置。JDD 在所有 5 种情况下将三个异常全都检测了出来。HOTSAX 在滑动窗小于正弦波周期的情况下没能把所有的异常检测出来。在滑动窗长度为 40 的情况下，HOTSAX 将起始位置为 813 的正常子序列识别为最为数据集中最异常的子序列，使得检测结果的质量进一步

下降。不适当的滑动窗长度设置将使原有时间序列异常定义的效用变差[54]。这个图表表明 JDD 对于滑动窗的长度相较于 HOTSAX 而言更不敏感。

表 4-3 不同滑动窗长度情况下 HOTSAX 和 JDD 在合成数据集上的检测结果。

Table 4-3 The discovery result on synthetic dataset with different window length.

Window length	JDD	HOTSAX
20	3 (133,761,70)	1 (133,981,728)
40	3 (121,60,750)	2 (813,148,59)
60	3 (111,740,50)	3 (81,771,142)
80	3 (126,714,21)	3 (78,690,769)
100	3 (61,688,905)	3 (61,689,281)

下一步我们将展示在真实世界数据集上，即使滑动窗的长度设置合适，原有时间序列异常的定义也未必能检测出所有的异常。JDD 的检测结果总是等同于或好于 HOTSAX 的检测结果。

4.4.2 心电图的异常检测

心电图（Electrocardiogra, ECG）是一个描述心跳活动的时间序列。Lin 等人[18]已经展示了时间序列异常检测在心电图方面的效用。在他们的实验中使用到的心电图数据集包含有限数量的异常。而且这些异常在形状方面相差很大，因此原始的时间序列异常定义可以非常容易的识别这些异常。然而形状相似的异常很有可能会在短期内在同一病人身上发生多次。

例如图 4-3 的上图为从 MIT-BIH 心率失常数据库[55]sel102 数据集中摘到的一段拥有多个异常的心电图时间序列，X 轴代表时间秒，y 轴代表电压毫伏，该图中的时间序列包含有连续出现的心率失常。其中有 6 次红色高亮的心跳是由心脏病医生手工标注的。我们将滑动窗长度设置为 300，因为每个心跳平均被采样 300 次。我们分别使用 HOTSAX 和 JDD 检测出该时间序列中的 10 个异常。

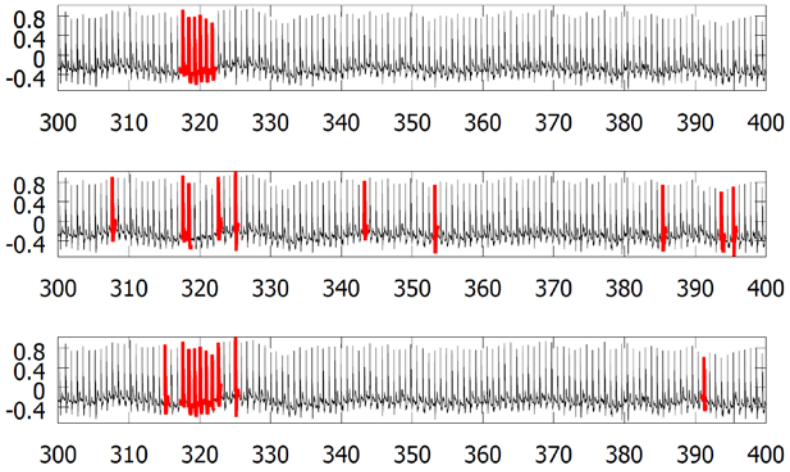

图 4-3. 上图：从 MIT-BIH 心率失常数据库[55]sel102 数据集中摘到的一段拥有多个异常的心电图时间序列，X 轴代表时间秒，y 轴代表电压毫伏，红色粗线是由心脏病医生标的异常。中图与下图分别为 HOTSAX 与 JDD 的检测结果。

Fig. 4-3 Upper: An ECG time series extracted from sel102 dataset in MIT-BIH Arrhythmia Database [55]. The x axis stands for the time in second, while the y axis stands for the voltage in mV. The red thick lines are annotated by cardiologists. Middle: the result of HOTSAX. Lower: the result of JDD method.

图 4-3 的中图和下图分别显示了 HOTSAX 和 JDD 的检测结果。如图 4-3 的中图所示，HOTSAX 识别出该时间序列中的 2 个异常。这两个异常在检测结果中排名第 7 和第 9 名，这意味着 HOTSAX 的检测结果的前 6 名中不包含任何实际的异常。由于异常的连续发生，我们也可以从模式切换的角度考察 HOTSAX 的检测性能，因为模式切换检测也是时间序列异常检测的效用之一[15]。事实上 HOTSAX 检测捕捉到了心率进入异常状态，但是没能捕捉到心率从异常状态中恢复的情况。

图 4-3 的下图显示了 JDD 的检测结果。JDD 检测方法在其检测结果中的前 6 名识别出了所有 6 个异常。其余结果中的 3 个的分布也围绕着实际异常，表示 JDD 识别出异常状态的开始与结束。

4.4.3 心脏出血漏洞的异常

我们也展示 JDD 在健康领域以外的应用的效用。心脏出血（Heartbleed）漏洞在 2014

年引起了广泛的关注，它是广受欢迎的 OpenSSL 加密软件库的一个漏洞。这个漏洞可能允许攻击者在未曾授权的情况下获取运程 HTTPS 服务内部的机密信息[56]。得到机密信息的方法之一就是重复发送恶意的心跳请求给远程服务器[57]。

在这个实验中，我们构造了一个心脏出血攻击以及异常检测的场景。我们建立了一台网页服务器并从客户端向服务器发起 20 个 TLS 链接。我们向服务器发送 100 个恶意心跳请求，每次 10 个。每个请求只要求服务器发回 30 个额外的字符，这种方式可以巧妙地躲避 BPF 包过滤的检测[58]。我们计算每 5 分钟心跳请求与回复数据包大小的差异并将它们以时间序列的方式保存下来。我们使用 HOTSAX 与 JDD 检测该时间序列中的异常。

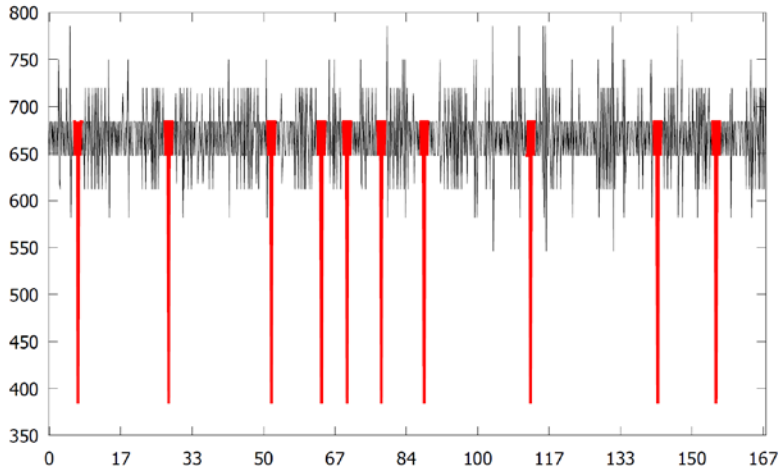

图 4-4. 每五分钟心跳请求与回复数据包大小的差异的时间序列。每个用红色粗线标出的尖峰是一次连续 10 个心脏出血的攻击。

Fig. 4-4 The difference of packet size between total heartbeat requests and replies every five minutes. Each spike plotted in red represents a ten-times heartbleed attack.

表 4-4 在不同滑动窗长度情况下，HOTSAX 与 JDD 在心脏出血攻击数据集上的检测结果。

Table 4-4 The discovery result of Heartbleed detection with different window length.

Window length	JDD	HOTSAX
10	1,2,4,9	1,2
12	1,2,3,5,8,10	1,2,3,7
14	1,2,3,5,7,8	1,2,3
16	1,2,3,4,5,8	1,2,3,4,8

图 4-4 展示了每五分钟心跳请求与回复数据包大小差异的时间序列，x 轴的每个单位表示 5 分钟的时间段。y 轴表示请求与回复数据包大小的差。每个用红色粗线标出的尖峰表示一次连续的心脏出血攻击。我们分别使用 HOTSAX 与 JDD 检测该时间序列中的 10 个异常。

表 4-4 展示了 HOTSAX 与 JDD 的检测结果。我们使用了不同长度的滑动窗进行异常检测。从该表我们可以看出，两种方法都没能把所有的攻击检测出来。但是在任何滑动窗长度设置下，JDD 检测方法总能检测出更多的异常，并且这些实际异常在 JDD 检测结果中的排名比 HOTSAX 的更靠前。

4.4.4 人体活动变化检测

在无处不在的计算技术[59]中，人类活动识别是一项重要的技术。该技术在人身各处放置加速度传感器，通过传感器的读数判断与识别人类活动。在本次实验中，我们在人类活动数据集[60]上进行模式变化检测。

该数据集是由 15 位参与者在胸口佩戴未经校正的加速度传感器，并从事诸如走路静坐跑步等活动而产生的。加速度传感器分别在 x,y 和 z 方向上以 52Hz 的频率记录数据。我们计算$g = \sqrt{x^2 + y^2 + z^2}$获得时间序列用于模式变化检测，该时间序列如图 4-5 所示。

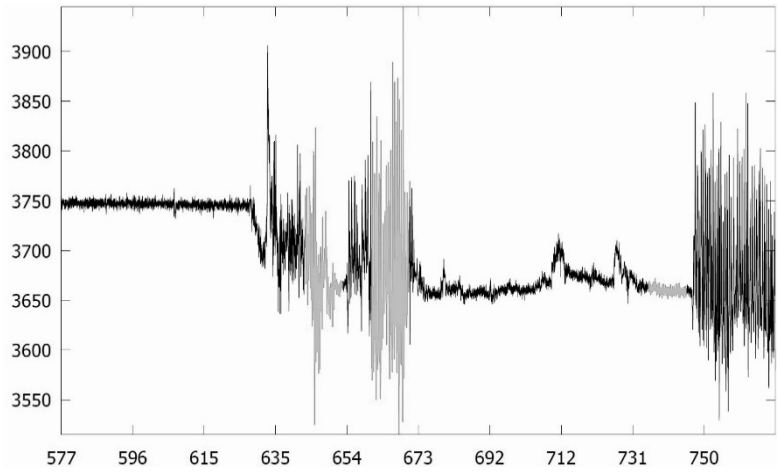

图 4-5. 在参与者胸口佩戴未经校正的加速度传感器所获得的加速度幅值时间序列。该图中包含 4 个人类活动，活动切换的部分使用浅色曲线标出。

Fig. 4-5 The time series of g acceleration measured by the accelerometer mounted on the chest of the number one participant. Four activities were included in this figure. Activity changes are plotted in gray. Courtesy of Casale et al. [60].

图 4-5 中，x 轴表示时间，每个单位为秒。y 轴为加速度。在该图中有四个人类活动，他们按顺序分别是：（1）在电脑前工作；（2）上下楼梯；（3）站立；（4）行走。活动与活动之间的切换使用红色粗线标。我们分别使用 HOTSAX 与 JDD 检测该时间序列中的 10 个异常。

表 4-5 在不同长度滑动窗的情况下，HOTSAX 与 JDD 在人类活动数据集上的检测结果。

Table 4-5 The discovery result of human activity change detection with different window length.

Window length	JDD	HOTSAX
5s	1,7	1,8
7.5s	1,5	1,6
10s	1,4,10	1,5

在不同长度滑动窗的情况下，HOTSAX 与 JDD 在人类活动数据集上的检测结果如表 4-5 所示。为了易读性，我们使用了秒作为滑动窗的长度单位。从该表我们可以看出，当滑动窗长度为 10 秒时，JDD 可以检测出所有的活动切换。除此之外，在滑动窗长度为 5 秒和 7.5 秒的情况下，JDD 总能将活动切换识别为比 HOTSAX 的结果的排名更高

的异常。

4.4.5 计算复杂度

本节我们将评估 JDD 检测方法的计算复杂度。诸如编程能力、计算平台差异等因素会影响检测方法的计算时间，同时也会降低实验结果的可重现性。我们将通过距离函数（表 4-2 第 7 行）的调用次数来评估检测方法的计算复杂度。距离函数的执行以及相关 I/O 操作占到了检测方法总运行时间的 99%以上，距离函数的调用次数是更好的衡量计算复杂度的方法。

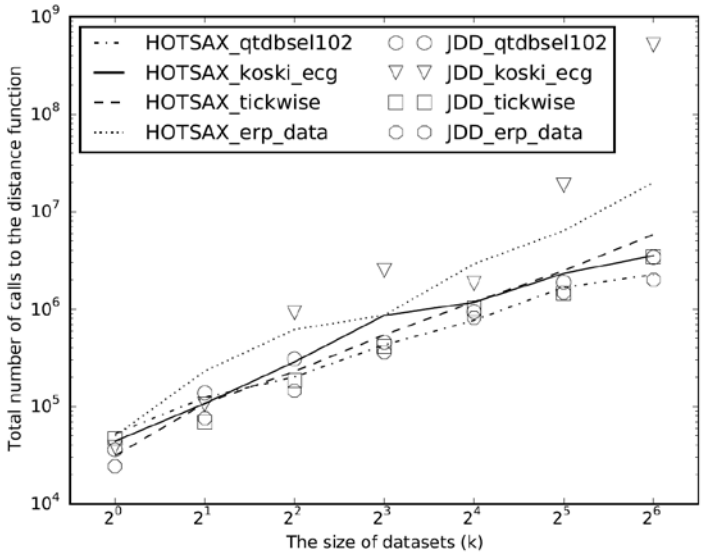

图 4-6. HOTSAX 和 JDD 检测方法的计算复杂度，该实验使用了四个具有代表性的数据集[14]。X 轴代表数据集的规模，y 轴代表距离函数的调用次数。滑动窗的长度为 100。

Fig. 4-6 The computational efficiency of HOTSAX and JDD discovery method on four representative datasets [14]. X axis stands for the size of the datasets. Y stands for the total number of calls to the distance function. The size of sliding window is set to n = 100.

我们分别用 HOTSAX 和 JDD 在四个表有代表性的数据集中检测 3 个异常。这四个数据集由[14]提供。滑动窗的长度被设置为 100。因为 HOTSAX 在检测过程中使用了随机方法，因此每次执行结果都反应了不同的计算量。我们通过计算每 10 次运行结果的平均值来评价 HOTSAX 的计算复杂度。

图 4-6 展示了本实验的结果。x 轴代表数据集的规模，y 轴代表距离函数的调用次数。通过该图我们可以观察到 JDD 检测方法与 HOTSAX 的曲线是非常贴近的。这表明 JDD 与 HOTSAX 在计算复杂度方面是相似的。

4.4.6 JDD 检测方法对参数 J 的敏感度

我们已经展示了 JDD 在参数 J 设置为 3 时的实验结果。在本小节我们展示J = 3不是一个为我们的实验精心调整所得的结果。我们 JDD 检测方法在计算复杂度与检测性能方面对参数 J 的敏感度。在这一实验中，我们使用 JDD 检测方法在上一小节中涉及的四个数据集中检测前 10 个异常，数据集的大小都设置为 8k。我们将参数 J 设置为从 2 至 8，间隔为 1。我们将滑动窗的长度设置为 100。

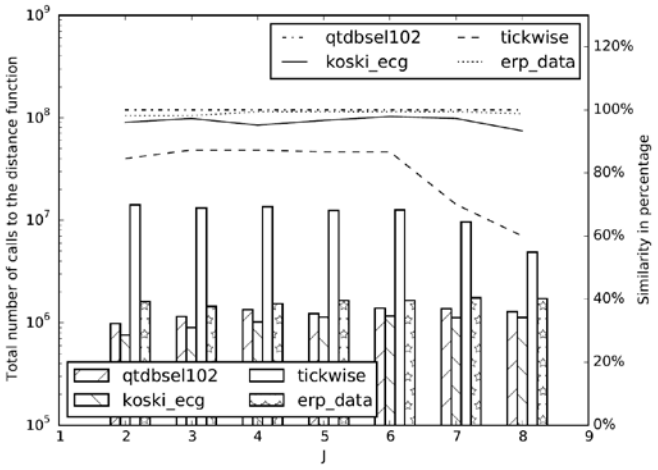

图 4-7. 条状图：JDD 检测方法在参数 J 的不同设置下的距离函数调用次数。线图：在参数 J 的不同设置下 JDD 方法检测结果的一致性趋势。x 轴表示参数 J 的不同设置，左侧的 y 轴标明距离函数的调用次数，右侧的 y 轴标明一致性百分比。

Fig. 4-7 Bars: The total number of calls to the distance function required by JDD discovery method under different settings of the parameter J. Lines show the consistency of the results discovered by JDD under different settings of the parameter J.

图 4-7 的条状图展示了 JDD 在不同数据集和参数 J 设置的情况下调用距离函数的次数。正如我们期望的那样，qtdbsel102, koski_ecg 和 erp_data 的方条随着参数 J 的变大而慢慢变长。然后 tickwise 的方条却在随着参数 J 的变大而开始变小。通过深入地调查

我们发现，tickwise 数据集中存在一个较小的子序列簇，它们之间比较相似，但与大多数其他子序列不相似。在参数 J 较大的情况下，JDD 将该簇内所有的子序列识别为异常，所以更快得跳过其他候选子序列的计算。总之，可以从图 4-7 的条状图看出，JDD 检测方法的计算复杂度不会随着参数 J 的变大而快速变大。

我们也对评估 JDD 检测方法在检测性能方法对参数 J 的敏感度。为了展示估计结果，我们首先要定义检测结果一致性的测量方法。我们定义$\mathbb{U}$为参数 J 所有整数设置的集合，定义$\mathbb{V}$为所有检测结果中异常的排名的集合。在本例中，$\mathbb{U} = [2,3,\dots,8]$，$\mathbb{V} = [1,2,\dots,10]$。$\mathbb{U}$和$\mathbb{V}$中元素的总数分别为$|\mathbb{U}| = 7, |\mathbb{V}| = 10$。对于任何正整数$j \in \mathbb{U}, \mathrm{k} \in \mathbb{V}$，我们使用$\mathrm{D^j}$表示 JDD 在参数$\mathrm{J = j}$的情况下检测到的异常集合，使用$d_k^j$表示$D^i$中的第$k$个异常。对于任意正整数$\mathrm{i} \in \mathbb{U}$，如果$\mathrm{d_k^j}$与$\mathrm{D^i}$中的任意异常存在重叠，记作$Ovlp(d_k^j, D^i) = 1$，那么我们认为$d_k^j$被$\mathrm{J = i}$情况下的 JDD 检测方法捕获，否则$Ovlp(d_k^j, D^i) = 0$。那么异常$d_k^j$被不同参数 J 设置情况下的 JDD 检测方法捕获的百分比，记作$P(d_k^j)$，可以表示为：

$$P(d_k^j) = \frac{\sum_{h \in \mathbb{U}, i \neq j} Ovlp(d_k^j, D^i)}{|\mathbb{U}| - 1} \tag{4-2}$$

为了测量D^j与其他所有$D^i, i \in \mathbb{U}, i \neq j$的相似度，记作$S(D^j)$，我们可以取$D^j$中每个异常$d_k^j$的捕获百分比$P(d_k^j)$的均值。同时，我们将异常排名考虑在内，给于每个百分比以相应的权重，即：

$$S(D^j) = \frac{\sum_{k \in \mathbb{V}} (|\mathbb{V}| - k + 1) \times P(d_k^j)}{\sum_{k \in \mathbb{V}} k} \tag{4-3}$$

根据定义，$S(D^j) \in [0\%, 100\%]$。该定义表明了D^j与其他参数 J 设置情况下 JDD 检测结果的相似性，如果D^j中包含的大多数异常被他参数 J 设置情况下 JDD 检测方法捕获，那么$S(D^j)$将会非常接近 100%，这一定义粗略表明了 JDD 检测方法在不同参数 J 设置情况下检测结果的一致性。

图 4-7 的线状图展示了根据(4-3)计算得到的 JDD 检测方法在不同参数 J 设置情况

下检测结果的一致性趋势。qtdbsel102, koski_ecg 和 erp_data 的曲线都非常靠近 100%，并且没有随着参数 J 的变化而剧烈变化。当参数 J 小于等于 6 时 tickwise 曲线维持在 80%以上，当参数 J 增长到 8 时，相似度下降至 60%。通过深入调查我们发现 tickwise 中包含着一个较小的子序列簇，它们被使用较大参数 J 的 JDD 检测方法识别为异常，但是使用较小参数 J 的 JDD 检测方法未能将该簇识别为异常。这一现象表明使用较大参数 J 的 JDD 检测方法可以检测出由异常构成的小型簇，这是原有时间序列异常定义做不到的。

4.5 本章小结

为了改善单维时间序列异常检测方法的检测效果，我们对异常的定义提出了改进。即 J 距离异常。我们也提出了相应的异常检测方法用于缓解新定义引入的计算复杂度方面的压力。实验表明，与原始时间序列异常相比，JDD 能捕获更多实际上的异常，且实际异常的排名更为靠前。我们的 JDD 检测方法与经典的异常检测方法 HOTSAX 相比，具有相近的计算复杂度。此外，当使用应用相关知识时，通过合理设置参数 J，JDD 检测方法可以更快获取更好质量的检测结果。

本章提出的 J 距离异常定义与检测方法已经在数据挖掘领域的国际会议 ICDM 2015 发表[121]。

第五章 多维时间序列异常检测

本章研究多维时间序列异常检测方法。为改善检测效果和计算复杂度问题，提出了多种异常检测方法结合的检测方案，通过实验比较本方案与各类异常检测方法的性能差异，并通过实验分析本方案的参数敏感度。

5.1 问题描述

目前在时间序列分析相关的技术方面，大部分研究成果集中在针对一维时间序列的处理上[34][36][61][62][63]，然而在各领域的实际应用中， 来自于多维时间序列处理的需求显得越来越重要。例如在移动计算领域，用户随身携带的移动设备在不断地移动，他们在不同时间序列通过无线连接向时间序列空间数据库发送各自的位置；在云计算平台中，为了了解每台虚拟机的健康情况，需要需要统计每台虚拟机的各种性能指标，包括 CPU 系统时间、CPU 用户时间、磁盘读写次数、网络收发数据数量、内存读写次数等多种信号源进行监测与记录[65]；并在自然环境信息系统中，跟踪动物迁移、飓风轨迹等应用保存被观测对象随着时间变化的位置，生成庞大的高维数据集合[64]；在人体的各个部分安装各类传感器，通过等时间间隔捕获的动作相关数据也是多维数据。总之，任何包含多个变量的时间序列都可以被视为多维时间序列来处理。

与单维时间序列相比，多维时间序列能对受监测的系统进行更细致具体的描述，为复杂系统的异常溯源提供了可能。异常溯源是指在检测系统异常的同时给出异常发生原因的提示[67]。传统的时间序列异常检测方法只给出异常发生与否的提示，而不给出发生异常的原因。异常的发生可能会在受监测系统中快速地传播开来，造成大范围的损失。知道异常的原因可以提早发现故障，缩短从发现故障到采取行动进行维修的平均时间。例如在水环境监测任务中，与水质相关的指标有数十种，由水环境监测产生的时间序列的维度高达几十维。当水质发生异常时，我们可以通过分析水环境的各种指标波动时的相似关系，找出与异常存在密切相关的几个指标，然后针对这些重要指标进行着重分析

实际行动。

5.1.1 计算复杂度问题

时间序列的高维度特性给异常检测带来以下两个困难：（1）多维时间序列数据集合的规模通常处于高速增长的过程中，其维度也随着传感器技术的进步而不断增加。多维数据往往需要较大的存储空间，较高的传输带宽，较高的计算复杂度，这些需求使得时间序列异常检测的成本不断增加，（2）部分维度可能包含噪音，维度与维度之间存在相关性，保留所有维度会降低时间序列索引方法的性能，从而影响时间序列分析的性能。

5.1.2 异常溯源与降维方法

在多维时间序列中，不同维度可用于描述受观测事物不同方面的状态与情况。现在检测方法只能发现异常发生的时间，但并不能异常溯源，即发现事物的哪方面存在异常。

传统的降维方法（如 PCA[66]，DFT[30]，DWT[31]，SVD[32]）可以降低时间序列的维度，并起到降低算法复杂度的作用，但是这些方法产生的降维数据与原始数据之间不存在维度的对应关系。这为多维时间序列异常检测的异常溯源问题带来了困难。

图 5-1 为经典的 PCA 降维方法的简单例子。假设有一个二维时间序列$\mathrm{T_{d,m}} = \begin{pmatrix} t_{1,1} & \cdots & t_{1,m} \\ t_{2,1} & \cdots & t_{2,m} \end{pmatrix}$，若把每个时刻的采样看作二维图中的一个点，则图 5-1 中的蓝色圆展示了$\mathrm{T_{d,m}}$的每个采样点的分布。对这些点进行 PCA 分析可得$\mathrm{T_{d,m}}$的主成份，如图中的绿色箭头所示。通过忽略 PCA 第二维度，我们可以将以 x 轴与 y 轴为坐标轴的二维空间中的点，近似表达为以 PCA 第一维度为坐标轴的一维空间中的点。

该近似表达具有以下两个特征：（1）PCA 近似表达是原始数据所有维度，即 x 轴与 y 轴线性的表示；（2）原始数据维度与 PCA 近似表达在应用场景中不一定存在对应关系。这两个特征给异常溯源带来困难，当检测方法发现异常时，我们不能通过 PCA 的近似表达的维度断言原始数据中的哪个维度对于异常负有主要责任，维度对应关系的缺失使得异常溯源问题在原始数据维数较高时变得尤其突出。

本章从异常溯源入手，研究适用于多维时间序列异常的溯源方法，通过异常溯源改

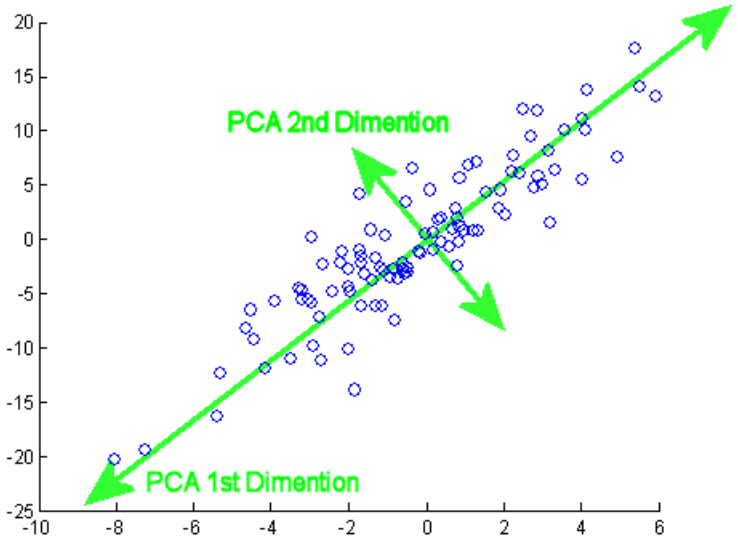

图 5-1. PCA 降维方法的简单例子

Fig. 5-1 A simple two-dimension example for PCA dimension reduction

善异常检测效果，并通过溯源降低时间序列维度，降低多维时间序列异常检测的计算复杂度

5.2 相关工作与背景知识

本文首先给出多维时间序列的定义。

定义15. 多维时间序列(Multi-dimension Time Series)： *多维时间序列是 d 个有序的分别由 m 个实数组成的组合，表达为* $T = \begin{pmatrix} t_{1,1} & \cdots & t_{1,m} \\ \cdots & \cdots & \cdots \\ t_{d,1} & \cdots & t_{d,m} \end{pmatrix}, t_{i,\mathrm{j}} \in R$，*设 A 为所有维度的有序子集，* $\mathrm{A} = \{a_1, \ldots, a_s\}, a_i \neq a_j, s \in [1, d]$。$T(A)$ *为 T 的子集，即* $T(A) = \begin{pmatrix} t_{a_1,1} & \cdots & t_{a_1,m} \\ \cdots & \cdots & \cdots \\ t_{a_s,1} & \cdots & t_{a_s,m} \end{pmatrix}$

通常我们只对多维时间序列的局部感兴趣，多维时间序列中子序列的定义为：

定义16. 多维子序列(Multi-dimension Subsequence)： *给定维度为 d 长度为 m 的时间*

序列T，子序列$C(A)_{p,n}$是$\mathrm{T(A)}$中从位置p开始连续采样n次获得的子集合。表达为$C(A)_{p,n}=\begin{pmatrix} t_{a_1,p} & \cdots & t_{a_1,p+n-1} \\ \cdots & \cdots & \cdots \\ t_{a_s,p} & \cdots & t_{a_s,p+n-1} \end{pmatrix}$。当$\mathrm{A}=\{1,\ldots,d\}$时，将$C(A)_{p,n}$记为$\mathbb{C}_{p,n}$。通常我们讨论时间序列的异常检测时只涉及一个$n$，所以$C(A)_{p,n}$可以缩略地表示为$C(A)_p$。

我们可以通过多维滑动窗获得一条多维时间序列中的所以可能的多维子序列：

定义17. 多维滑动窗(Multi-dimension Sliding window)： *给长度为m的时间序列T，对T使用多维滑动窗可生成所有可能的长度为n的多维子序列。*

我们还需要比较多维子序列之间的相似性，我们将多维子序列之间的相似性定义为：

定义18. 多维时间序列之间的距离 (Multi-dimension Time series Distance)： *距离dist的输入为两个多维序列S和T，维度与长度相同，输出为一个非负实数R，该输出可用于表征S和T的距离或相似度，即$dist(S,T)$。*

时间序列异常检测方法中最常用的距离定义为欧氏距离，给定两个长度为 n，维度为 d 的多维时间序列 S 和 T，则 S 和 T 之间的欧氏距离为：

$$dist(S,T)=\sqrt{\sum_{i=1}^{d}\sum_{j=1}^{n}\left(s_{i,j}-t_{i,j}\right)^2} \tag{5-1}$$

除了关注多维时间序列的时间特性之外，我们也需要关注多维数据的空间特性，即在同一时刻多维时间序列的所有采样点：

定义19. 时间序列中的点(Point in Time series)： *我们将时间序列中的点定义为时间序列在某一特定时刻所有维度的采样点，若$T=\begin{pmatrix} t_{1,1} & \cdots & t_{1,m} \\ \cdots & \cdots & \cdots \\ t_{d,1} & \cdots & t_{d,m} \end{pmatrix}$是维数为$d$长度为$m$的时间序列，那么时间序列在第$\mathrm{i}$时刻的点可以表示为$\mathbb{t}_{\mathrm{i}}=\begin{pmatrix} t_{1,i} \\ \cdots \\ t_{d,i} \end{pmatrix}, \mathrm{i}\in[1,\mathrm{m}]$。*

定义20. 点距离 (Point Distance)： *点距离可用于表征空间中两个点的距离或相似度，若$\mathbb{t}_{\mathrm{i}}$和$\mathbb{t}_j$是时间序列T中的两个点，则它们之间的距离表示为$dist(\mathbb{t}_{\mathrm{i}},\mathbb{t}_{\mathrm{j}})$。*

欧氏距离是最常用的两点之间的距离，给定一个时间序列中的两个点$\mathbb{t}_{\mathrm{i}}$和$\mathbb{t}_j$，则$\mathbb{t}_{\mathrm{i}}$和

$\mathbb{t}_j$之间的欧氏距离为：

$$dist(\mathbb{t}_i, \mathbb{t}_j) = \sqrt{\sum_{k=1}^{d} (t_{k,i} - t_{k,j})^2} \tag{5-2}$$

我们还需要关注多维时间子序列以及点的最近邻相关问题。由于多维时间子序列的最近邻定义与 0 至定义 7 一致，在此不再赘述。由于空间中的点与点之间不存在重叠问题，0 至定义 7 同样适用于点的最近邻问题，点的最近邻定义在此不再赘述。

5.2.1 异常

异常是数据集中与正常定义不一致的模式。异常从数据的组织结构上可以点异常和时间序列异常。

1. 点异常。在一个时间序列数据集中，单个数据点（定义 19）与其余数据点相比较为异常，则可以将这样的数据点称为点异常。这最简单的一类异常，绝大多数异常检检测方法都用于检测这类异常。
2. 上下文异常或时间序列异常。如果一个数据点在数据上下文中呈现异常，但其本身的幅值不存在异常，则该类数据点被称为上下文异常，也称为时间序列异常，是本文着重研究的一类异常。

点异常与时间序列异常时常存在关联，点异常的检测对时间序列异常的检测具有辅助作用。

5.2.2 LOF

点异常的检测方法多种多样，包括支持向量机(Support Vector Machine, SVM)，人工神经网络(Artifical Neural Network)等其他主流的异常检测方法。在基于相对密度的众多异常检测方法中，Local Outlier Factor (LOF) [52]是一种简单有效的半监督异常检测方法，它的检测性能超过了其他主流的异常检测方法[68]。LOF 本身是对待检测点的异常程度的量化表示，这是其他检测方法不具备的。本章将利用 LOF 的这些特性，在检测方案中将 LOF 作为时间序列异常检测的辅助方法。

LOF算法给每个待测试数据点一个表征异常程度的值。对于任何一个给定的数据点，LOF 值等于其 k 个最近邻的平均局部密度以及待测数据点本身的局部密度的比值。为了找到一个数据点的局部密度，首先要分别找到该点与其所有 k 个最近邻之间的可达距离(Reachability Distance)：

$$rd_k(A,B) = max\big(kdist(A), dist(A,B)\big). \tag{5-3}$$

其中 A 是一个数据点，B 是 A 的邻居，$kdist(A)$是 A 的 k 距离（定义 11），$dist(A,B)$表示 A 与 B 之间的距离。在找到 A 与 B 之间的可达距离后，局部可达密度(Local reachability density)可表示为：

$$lrd_k(A) = \frac{|N_k(A)|}{\sum_{B\in N_k(A)} rd_k(A,B)} \tag{5-4}$$

其中$N_k(A)$表示 A 的 k 个最近邻，$|N_k(A)|$为$N_k(A)$中包含的最近邻的数量。通过计算数据点 A 和其 k 个最近邻的局部可达密度，LOF 值可表达为：

$$LOF_k(A) = \frac{\sum_{B\in N_k(A)} lrd_k(B)}{|N_k(A)| \times lrd_k(A)} \tag{5-5}$$

根据 LOF 值的定义，若 A 的 k 个最近邻的平均局部密度大于等于 A 的局部密度，则 LOF 值大于等于 1，A 可能是一个异常，且其异常程度由两者的比值决定。否则，A 是正常数据点的可能性更大一些。图 5-2 展示了 LOF 异常检测的一个例子。

图 5-2 中有两个大小与密度不同的簇C_1和C_2，簇C_1的密度比C_2小一些。直观地看，点p_1和p_2为异常，而C_1和C_2为正常数据点的集合。在使用基于全局密度方法进行异常检测时，如果使用C_2的密度为阈值，那么C_1中的所有点都会被误认为是异常；如果使用C_1的密度为阈值，p_2不会被识别为异常。该例子的局部密度的不同使得基本全局密度的异常检测方法失效。LOF 异常检测方法通过分析待测数据的邻居的局部密度来判断待测数据本身的局部密度是否正常。因此 LOF 不需要设置全局密度阈值。在本例中，由于p_2的局部密度比其最近邻的局部密度小很多，所以 LOF 方法能将p_2检测为异常，并通过赋予该点以 LOF 值来表达其异常程度。

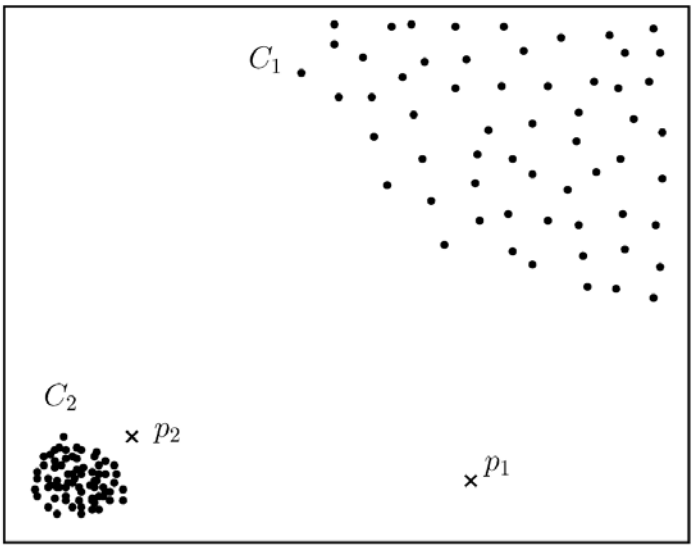

图 5-2. 基于局部密度的异常检测方法 LOF 可以检测出异常p_1和p_2，而基于全局密度的异常检测方法无法检测出异常p_2。

Fig. 5-2 Advantage of local density-based techniques over global density-based techniques.

5.3 多维时间序列异常检测方案

本文要提出多维时间序列异常检测方案（Multi-dimensional Discord Discovery scheme，MDD）建立在以下观察的基础上：在多维时间序列中，时间序列异常往往伴随着点异常的发生。在使用多个指标组成的时间序列描述一个复杂系统的状态时，当系统中发生异常时，异常对多个指标带来程度不同的影响，使得多维时间序列中出现点异常。我们可以通过实时检测点异常来发现潜在的时间序列异常以及异常的原因。下面给出一个网页服务器的例子。

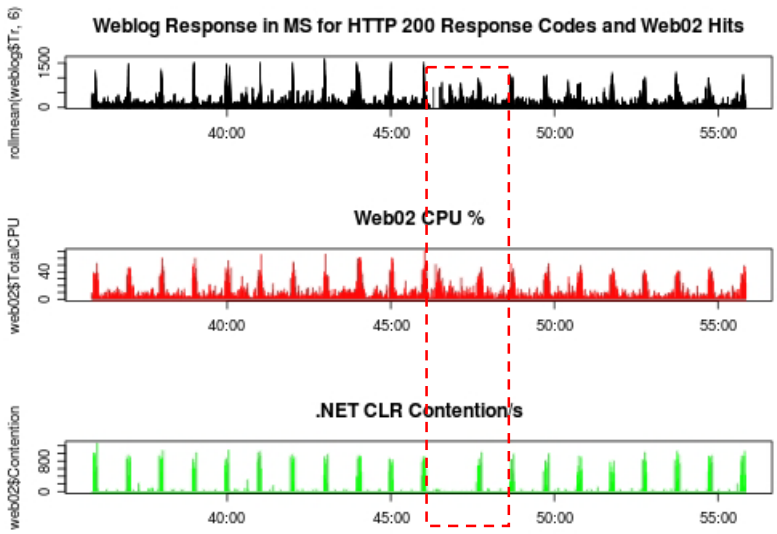

图 5-3. 网站服务器的性能统计数据组成的三维时间序列[69]。在红色虚线框内，CLR 的周期性争抢发生了推迟，暗示着异常的存在。而在相对应的时段内，三维时间序列的点也存在点异常。

Fig. 5-3 Three-dimension time series for web logs and performance monitoring data [69]. During the time period in the red dotted frame, the period of CLR contention/s changes, indicating the occurrence of an anomaly of the web server. There are also point anomalies in this period.

图 5-3 展示了网站服务器的性能统计数据组成的三维时间序列[69]。第一维是网页请求的响应时间，第二维是 CPU 利用率，第三维是.NET CLR 库的争抢率。在大部分情况下，争抢率符合周期性时间序列的特性，然而在红色虚线框内峰值争抢率的到来出现了延迟，这是一种时间序列异常。从三维时间序列的角度看，虽然所有的性能统计数据的幅值都处于正常范围内，但是三个维度的统计值的组合呈现异常，这些是点异常。

在多维时间序列中，点异常与时间序列异常时常结伴出现的。上述多维时间序列就是该现象的例子。事实上，许多相关文献[10][15][18][19][20][21][22][47][48]涉及的多维时间序列集合都存在这些现象。如心电图[55]、家庭用电监测数据[71]、太空梭发动机监测数据[70]等。

我们可以通过以下步骤检测数据集中的时间序列异常：首先通过检测点异常发现潜在的时间序列异常。根据时间序列中每个点的 LOF 值判断该点是否为点异常。若该点

为异常则分析各个维度对该异常的贡献程度，否则检查下一点。根据各维度对点异常的累计贡献来解决对点异常最多的维度。然后按照累计贡献程度对时间序列的不同维度进行筛选，以达到降维目的。一般的，在贡献程度越高的维度中越有可能找到时间序列异常，应该使用时间序列异常检测方法进行分析；反之则较不可能反映异常的根本原因，可以在后续的分析过程中忽略这些维度。最后，对选定的一个或少数几个维度进行时间序列异常的检测。我们可以根据多维时间序列距离（定义 18）判断时间序列异常的发生与否。

5.3.1 检测方案概述

MDD 的概述如图 5-4 所示。虚线箭头表示数据流向，实线箭头表示方案流程。该方案的输入由两部分组成：（1）待检测的多维时间序列的输入；（2）多维时间序列的历史数据库，该数据库不包含异常数据。

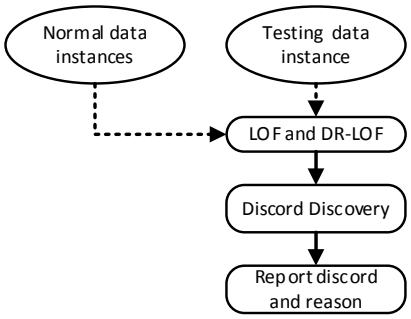

图 5-4. 多维时间序列异常检测方案概述

Fig. 5-4 Overview of Multi-dimensional Discord Discovery scheme

MDD 的输入为一个待测多维时间序列和一个历史多维时间序列数据库。历史数据库包含受监测系统在无异常的情况下产生的具有代表性的多维时间序列采样。

MDD 首先根据时间序列数据库计算待测多维时间序列中每个点的 LOF 值，若 LOF 值超过事先设定的阈值，则将其视为点异常，并使用 DR-LOF(Dimension reasoning LOF)

方法分析每个维度对该异常点的贡献程度；若 LOF 值未超过事先设定的阈值，则不进行 DR-LOF 值的计算。若所有点的 LOF 值均未超过阈值，则可选取具有最高 LOF 值的点进行 DR-LOF 值的计算。

然后 MDD 通过计算不同维度对异常点的累计贡献程度决定对异常的贡献程度最高的维度，并对贡献程度最高的一个或少数几个维度进行时间序列异常检测。最后，在将待检测维度中寻找最不寻常的子序列，并将该子序列发生的位置以及相关维度报告给用户。

MDD 使用 DR-LOF(Dimension reasoning LOF)分析各个维度对点异常的贡献程度，下面对 DR-LOF 作详细介绍。

5.3.2 溯源的 LOF（DR-LOF）

为了分析数据点的各个维度对点异常的贡献程度，我们提出了对经典 LOF 算法在异常溯源方面的扩展，记为 DR-LOF (Dimension Reasoning LOF)。DR-LOF 的基本想法是，把数据点的其中一个维度去掉并重新计算 LOF 值，新 LOF 值与原 LOF 值的比例反映了被去掉维度对异常的贡献程度。如果被去掉维度是点异常的主要表现，那么新的 LOF 值将会下降。所以我们为数据点的每个维度都计算其 DR-LOF 值，最小的 DR-LOF 值表明该维度对点异常的贡献程度最大。

每个数据点的每个维度都有一个对应的 DR-LOF 值，我们将 DR-LOF 定义为：

$$DR\text{-}LOF_k^i(A) = \frac{LOF_k^i(A)}{LOF_k(A)} \tag{5-6}$$

其中$LOF_k^i(A)$表示除去第 i 维的数据点的新 LOF 值。根据 DR-LOF 的定义，我们可以得到以下两个性质：

性质 1. 如果数据点 A 是正常的，那么移除数据点 A 的任何一个维度都将得到与原 LOF 值相近的新 LOF 值。

除此之外我们还可以得到如下性质：

性质 2. 如果数据点 A 是异常的，那么 DR-LOF 值按照如下规律变化：
当 A 的第 i 维是异常的主要原因，那么除去第 i 维的新 LOF 值将变小
当 A 的第 i 维不是异常的主要原因，那么除去第 i 维的新 LOF 值将变大

性质 1 与性质 2 的证明请见附件二。

在 DR-LOF 的实际使用过程中，我们计算一个数据点关于每个维度的 DR-LOF 值，然后找到使 LOF 值降低最多的维度，该维度包含最明显的异常信息，MDD 将对该维度进行后续的时间序列异常检测。

5.3.3 结合 JDD 与 DR-LOF 方法

MDD 的时间序列异常检测阶段对异常累计贡献最高的一个维度进行时间序列异常检测。MDD 可以使用传统的时间序列异常定义（定义 8）以及单维时间序列距离的定义（定义 4 或定义 11）。该定义的输入为一段较长的时间序列，它的输出为拥有最大最近邻距离的子序列。MDD 可以使用 HOTSAX 或者 JDD 检测方法加速单维时间序列中异常检测。

5.4 经验评估

我们使用实际生活中的数据对 MDD 进行经验评估。云计算环境中虚拟机性能统计数据是反映虚拟机健康状况的时间序列，当虚拟机中出现异常时，异常会通过性能指标的波动表现出来。虚拟机常用的性能统计指标多达数十种，而计算机环境中的异常可能只对其中一种或多种产生影响，因此，我们可以通过降维方法选出与异常相关度高的指标，从而减少异常检测的计算复杂度，减少其他不相关指标中噪声对异常检测产生的影响。

实验数据是从我们搭建的虚拟机环境中产生的。我们在一台装备有 Intel CPU E5606 的服务器上运行装有 CentOS 的 Xen 虚拟机，我们在这些虚拟机上运行与并行计算相关的应用，使用 Sysstat 工具采集 9 个具有代表性的性能统计指标，包括%user（CPU 时间用户态百分比），kbmemused（已使用内存），intr/s（每秒中断次数），rxpck/s（每秒接收包数量）等。

5.4.1 DR-LOF 示例

我们首先使用一个简单的例子验证 DR-LOF 的有效性。在本例中我们使用 Sysstat 工具采集了虚拟机的 9 个具有代表性的性能统计指标的数据，采集间隔为 5 秒，持续 1 个小时。在采集过程中有工作负载在后台运行，我们在采集过程中注入了两个简单的异常以测试 DR-LOF 的有效性。由于原始时间序列的维度较多且杂乱，在此我们不展示原始数据的曲线图，仅展示简化的分析结果，以增强可读性。

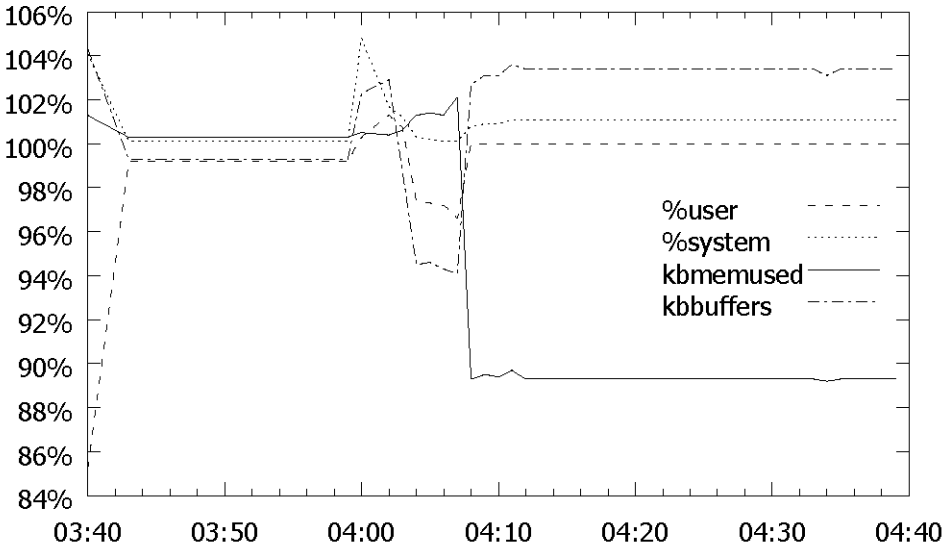

图 5-5. DR-LOF 的简单例子

Fig. 5-5 Ratio of LOF values with/without specific dimensions of data and all dimensions of data in the experiment to evaluate DR-LOF.

图 5-5 展示了 9 个维度中具有代表性的 4 个维度的 DR-LOF 值的变化趋势。这 4 个维度分别是%user，%system，kbmemused 和 kbbuffers。X 轴代表时间，y 轴代表 DR-LOF 的值。DR-LOF 值的波动是由虚拟机中工作负载以外的状况造成的，这些状况代表着潜在的异常。虚拟机器的事件列表如表 5-1 所示。

表 5-1 DR-LOF 的简单例子

Table 5-1 Anomaly list and the Results of DR-LOF

时间	虚拟机状况	DR-LOF 值最低的维度
03:40	CPU 竞争	%user
03:44-03:59	无	无
04:00-04:07	系统维护例程	kbbuffers
04:08-04:40	内存泄漏	kbmemused

本例开始时，我们注入了 CPU 竞争异常，即在用户态运行一个以 CPU 计算为主的应用程序，与正常的工作负载争抢 CPU 时间。从图 5-5 可以看出，%user 在 03:40 具有最小 DR-LOF 值，这意味着该时的异常主要是由%user 引起的。在本例的 04:08，我们注入了内存泄漏异常，即在用户态运行一个不断申请堆内存且从不释放内存的应用程序。从图 5-5 可以看出，kbmemused 在 04:08 到 04:40 期间拥有最小的 DR-LOF 值，表明内存使用情况是引起当前虚拟机异常的主要原因。

5.4.2 云计算环境中的异常检测

我们将 MDD 应用于云计算环境中的多维时间序列异常检测。MDD 以虚拟机的性能统计数据为多维时间序列，通过寻找时间序列中的异常及其原因。本章将通过检测云计算环境中两种典型异常验证 MDD 有有效性，它们是端口扫描和心脏出血异常。

MDD 在点异常的检测过程中使用正常数据集。正常数据集是受监测系统在正常工作的情况下通过采集各种性能统计数据而获得的数据集。在本例中，我们部署并运行带有正常工作负荷的虚拟机环境，并采集了间隔为 1 分钟，时长为 24 小时的数据集。根据 LOF 算法的作者所述， LOF 算法的相关参数 k 设置为 10 到 50 之间在各应用领域方面都能效地进行点异常检测，我们将 k 设置为 30。通过计算正常数据集中所有时间点的 LOF 值，我们将 LOF 阈值设置为 2.5，使得正常数据库中 99%的点的 LOF 值低于该阈值。

5.4.2.1 端口扫描

端口扫描是全面扫描计算机网络端口的行为。端口扫描在网络管理方面有合理的用

途。但是端口扫描由于其本质也可以被用于恶意目的，如寻找和突破目标计算机的薄弱环节以及管理员权限等。本例中，我们对虚拟机进行持续时间长达 5 分钟的恶意端口扫描，记录了端口扫描前后一小时的 9 种性能统计数据，并检测该多维时间序列中的异常及其原因。

首先，我们计算多维时间序列数据中每个点的 LOF 值。如图 5-6 的上图所示，除 18:28-18:36 之间的 LOF 峰值之外，其他时间点的 LOF 值均低于 2。18:28-18:36 之间的端口扫描给虚拟机的性能统计数据带来偏差，使得这个时间段内的点的 LOF 值大于预设的阈值 2.5。LOF 值超出阈值的时间点需要进行 DR-LOF 值的计算。虽然我们只需要计算 LOF 值超出阈值时间段内点的 DR-LOF 值，即 18:28-18:36 分之间的点的 DR-LOF 值，但是为了方便阅读与理解，我们计算了所有点的 DR-LOF 值，并挑选出具有代表性的几个维度进行展示。

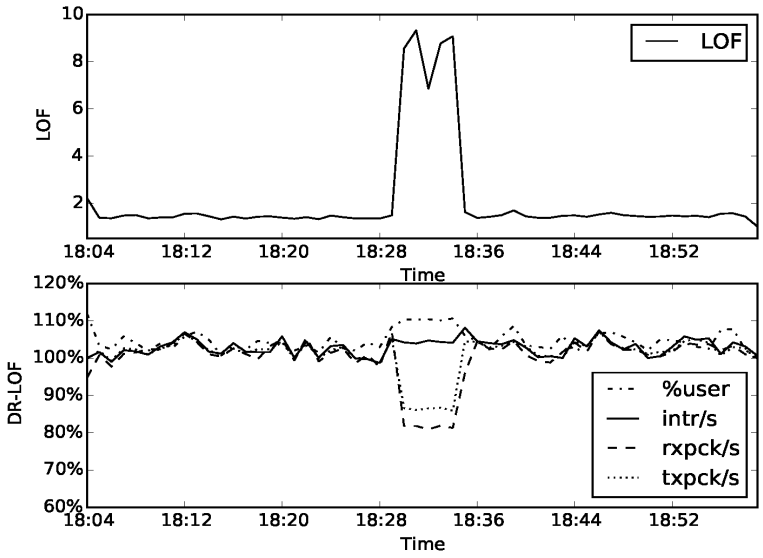

图 5-6. 端口扫描案例的点异常检测与异常溯源

Fig. 5-6 Point anomaly detecting and dimension reasoning in the case of port scanning

如图 5-6 的下图所示为待测数据集的 DR-LOF 值，由图可知 18:28-18:36 以外的点的各维 DR-LOF 值均在 100%附近波动，说明各维度对当前时间点的异常程度贡献较少

或贡献相当。在 18:28-18:36 之间 rxpck/s 和 txpck/s 的 DR-LOF 值最小，这表明相对于其他性能统计指标而言，如 intr/s，rxpck/s 和 txpck/s 对当前时间点的异常程度贡献更大，有更多可能从 rxpck/s 和 txpck/s 发现时间序列异常。我们可以对 DR-LOF 值最低的 rxpck/s 的时间序列进行异常检测。

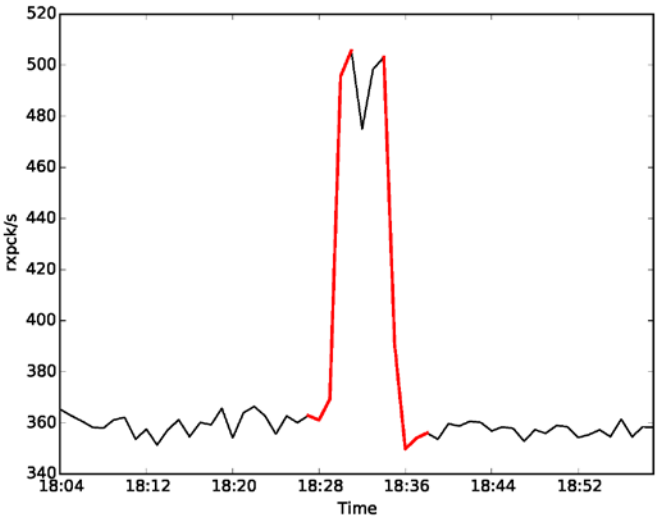

图 5-7. 端口扫描案例中的 rxpck/s 时间序列异常检测

Fig. 5-7 The time series discord discovery of rxpck/s in the case of port scanning

我们使用在上一章提出的 JDD 及其检测方法检测 rxpck/s 时间序列中的异常，如图 5-7 所示为 rxpck/s 时间序列及时间序列异常的滑动窗长度为 5 的情况下的检测结果。JDD 捕捉到了端口扫描开始与结束时 rxpck/s 的时间序列异常。我们也测试了滑动窗长度为 5 到 10 的任意整数情况下的检测结果。每一种滑动窗的长度设置都能使 JDD 及其检测方法发现 rxpck/s 中 18:28-18:36 之间的时间序列异常。

根据以上的检测结果，我们可以确定待测多维时间序列中异常程度最高的事件发生在 18:28-18:36 前后，并且是由能够引起网络设备数据包收发速率 rxpck/s 和 txpck/s 波动的异常产生的。

5.4.2.2 心脏出血

心脏出血（Heartbleed）漏洞 OpenSSL 加密软件库的一个漏洞。这个漏洞可能允许

攻击者在未曾授权的情况下获取运程 HTTPS 服务内部的机密信息[56]。得到机密信息的方法之一就是重复发送恶意的心跳请求给远程服务器[57]。我们使用与章节 4.4.3 相同的方法再现心脏出血漏洞，在一小时内对一台虚拟机发起 5 次攻击，记录下该虚拟机 9 种性能统计标准的每分钟均值，并利用 MDD 检测该多维时间序列中的异常。

如图 5-8 上半部分所示为心脏出血案例中所有时间点的 LOF 值。从图中可以看出，除 5 个时间点的 LOF 峰值大于事先设定的阈值 2.5 以外，其他时间点的 LOF 值均小于阈值。事实上这 5 个峰值在时间上正好对应心脏出血的攻击时间。我们需要对这 5 个时间点进行 DR-LOF 值的计算。为了易于阅读和理解，我们同时也计算出了其他时间点的各个维度的 DR-LOF 值，并选出了具有代表性的维度进行展示。

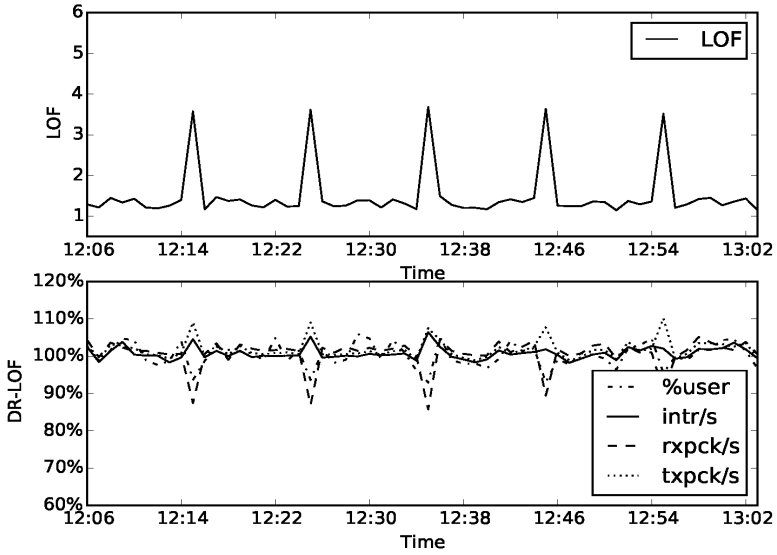

图 5-8. 心脏出血案例的点异常检测与异常溯源

Fig. 5-8 Point anomaly detecting and dimension reasoning in the case of Heartbleed

如图 5-8 下半部分所示为待测时间序列时间点的 DR-LOF 值。除上述 5 个时间点以外，所有时间点的各维度的 DR-LOF 值均在 100%周围波动，表明这些维度对当前点的异常程度的贡献较低或相当。而上述 5 个时间点的%user 和 rxpck/s 具有较小的 DR-LOF 值，表明它们对这 5 个点的异常程度的贡献较大，较有可能从这两个维度中发现应用相

关异常。因此，我们可以对 DR-LOF 值最小的 rxpck/s 作时间序列异常检测。

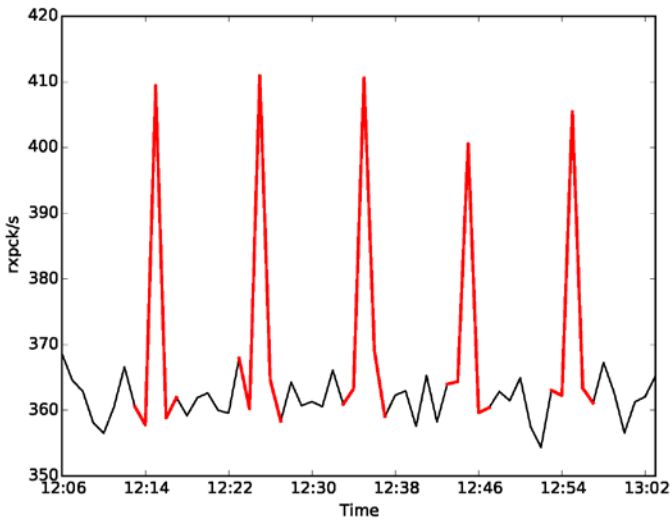

图 5-9. 心脏出血案例中的 rxpck/s 时间序列异常检测

Fig. 5-9 The time series discord discovery of rxpck/s in the case of Heartbleed

我们使用 JDD 及其相应的检测方法检测以及出血案例中 rxpck/s 的时间序列异常。如图 5-9 所示为 rxpck/s 时间序列以及使用长度为 5 的滑动窗的检测结果，检测结果已使用红色曲线高亮表示。JDD 检测方法在排名前 5 的结果中捕获所有心脏出血攻击。另外，我们分别使用滑动窗长度为 6 至 10 的所有整数作为滑动窗长度，也能在前 5 名检测出这些异常。

根据以上的检测结果，我们可以确定待测多维时间序列中异常程度最高的 5 个事件的发生时间，并且这些异常是由能够引起 rxpck/s 和%user 波动的异常产生的。

5.4.3 方案对参数的敏感度

本检测方案的正常运作涉及到的参数包括 LOF 阈值。MDD 的检测效果并不依赖于这些参数的精确设置，接下来我们讨论 MDD 对该参数的敏感度。

5.4.3.1 LOF 阈值

MDD 在 LOF 与 DR-LOF 的计算阶段需要使用到 LOF 阈值，即计算时间序列中所有

时间点的 LOF 值，选出所有 LOF 值大于阈值的点，计算它们的各维度的 DR-LOF 值。因此，阈值用来选出异常程度较高的点。在上述经验评估的实验设置中，我们通过计算正常数据集中所有时间点的 LOF 值，将 LOF 阈值设置为 2.5，使得正常数据库中 99%的点的 LOF 值低于该阈值。为了证明这种阈值设置方法的有效性，我们在一天内虚拟机环境中分别重现了 CPU 竞争、内存泄漏、端口扫描以及心脏出血的异常，我们采集性能统计时间序列并分析了时间序列中所有时间点的 LOF 值直方图。

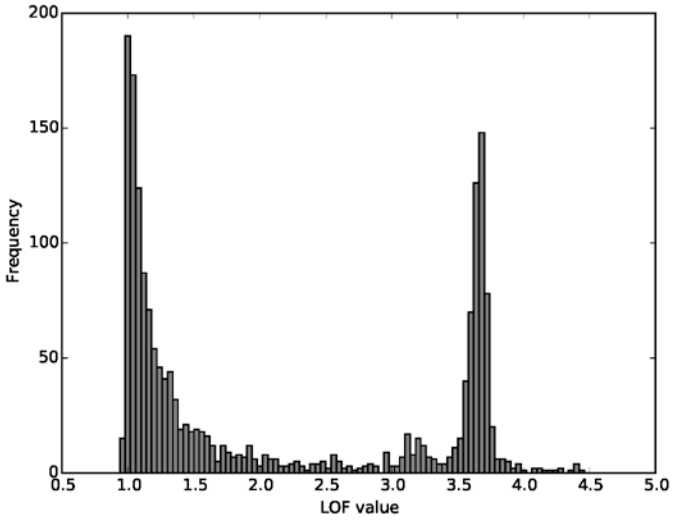

图 5-10. LOF 值分布情况

Fig. 5-10 Histogram of LOF value

如图 5-10 所示为 LOF 值的直方图，正常数据的 LOF 值主要分布在靠近 1.0 的位置，而与异常有关的统计数据的 LOF 值可能会靠近甚至超过阈值 2.5。阈值 2.5 能大致将正常数据与异常数据区分开来。因此，通过正常数据库中所有点的 LOF 确定阈值的方法是有效的。

此外，阈值的确定会影响到 MDD 的计算复杂度，但对方案的检测效果的影响是有限的。例如，若阈值未能排除一部分正常数据，这部分正常数据只会增加计算 DR-LOF 所需的时间，正常数据的各个维度对异常的贡献基本相当，因此不会影响 DR-LOF 方法选出对点异常贡献最多的维度；反之，若阈值未能过滤出一部分异常数据，部分异常数

据的缺失只会减少计算 DR-LOF 所需要的时间，DR-LOF 依然能选出对异常贡献最多的维度。总之，LOF 阈值的设置主要与 MDD 的计算复杂度有关，与方案的检测效果的关联较少。

5.5 本章小节

本章研究多维时间序列异常检测问题，提出可用于异常溯源的多维时间序列异常检测方案（Multi-dimensional Discord Discovery scheme，MDD），通过结合多种异常检测方法减少不相关维度对异常检测的影响，通过提出降维的方法减少异常检测的算法复杂度，同时通过降维达到异常溯源的目的。

本章提出的异常检测方案已在计算机系统结构领域的著名国际会议 COMPSAC2013 和计算系统结构领域的国际期刊 FGCS（影响因子: 2.786）发表[67][122]。

第六章 并行化时间序列异常检测方法

本章研究了时间序列异常的并行化检测方法，通过分布式存储与内存计算的方法减少了单机计算节点对于异常检测可扩展性方面的限制，并且研究了提高计算资源利用率和降低计算复杂度的方法。

6.1 研究背景

超大规模时间序列的异常检测受到计算节点的计算能力限制，同时也受到存储设备性能的约束。

6.1.1 单个计算节点能力限制

时间序列异常检测的规模受到单个计算节点的计算能力与存储能力的限制。一般单个计算节点的计算能力有限，以采样频率为 360Hz 的心电图数据为例，现代桌面处理器大约需要半小时的计算时间检测时长半小时的心电图数据，而心电数据的长度往往超过半小时，所以检测速度受到单台计算节点的计算能力的限制。此外，单个计算节点的内存空间有限制，可能无法满足超大规模时间序列的存储需求而不得不将数据集存放在读写性能低下的磁盘中。

6.1.2 磁盘读写问题

大部分时间序列异常检测方法需要随机读取数据集中的子序列，这些检测方法的相关文献涉及的数据集规模都在 64k 以下，现代计算机内存可以轻松容下这些数据集以及相关的计算辅助数据，如索引，从而缓解了检测方法在数据访问性能和效率方面的问题。

当数据集规模增大时，现代计算机内存可能无法容下所有数据，因此在数据访问时需要考虑磁盘读写效率的问题。Yankov 等人[10]提出硬盘感知的异常检测方法，通过顺序读取磁盘的方法改进了操盘的访问效率。然而当数据集处于磁盘中时，磁盘访问依然

严重影响着时间序列异常检测的耗时。如表 7-1 所示，磁盘访问花费的时间超出异常检测总耗时的一半。为了减少由于数据访问对异常检测造成的负面影响，我们应该研究基于内存计算(in memory computing)的异常检测方法。

6.2 基于互连感知 Amdahl 定律的并行化可行性分析

本文根据[73]的指导思想对时间序列异常检测进行并行化设计。检测方法的并行化设计问题涉及到计算需求的并行化以及通讯需求的并行化，并行化效率问题涉及到检测方法的计算通讯比。

并行化计算要求原问题可以被分解为互相独立的子问题，在不同的计算节点中进行计算。然而 Keogh 等人证实不能仅通过划分时间序列将异常检测分解为互相独立的子问题[15]。我们需要寻找其他方法以分解异常检测问题。

6.2.1 计算需求的并行化

为了研究时间序列异常检测计算需求并行化的可行性问题，我们首先需要分析异常检测的数据依赖关系。根据时间序列异常的定义，时间序列异常的基本检测方法的过程可以用如下公式表示：

$$p^{(1)} = \underset{p}{\operatorname{argmax}}\{nnDist(C_p) | 1 \le p \le m - n + 1\} \tag{6-1}$$

其中$p^{(1)}$为第一个异常的起始位置。$nnDist(C_p)$为子序列C_p的最近邻距离，n 为滑动窗的长度，m 为整个时间序列的长度。

公式(6-1)表达的是找到使$nnDist(C_p)$最大的 p，这需要分别求解每个子序列的最近邻距离。从该公式可以看出，求解每个$nnDist(C_p)$之间没有任何数据依赖关系，这些求解过程是完全互相独立的。因此我们可以将异常检测分解为多个分别求解子序列最近邻距离的子问题。

根据 Amdahl's law，算法的最大加速比与算法的计算并行度有关。由于时间序列异常检测可以被分解成相互独立的小问题，其中每个小问题即为寻找一个子序列在整个时

间序列中的最近邻。因此，时间序列异常检测的计算并行度与数据规模有关，随着数据规模的增大，时间序列异常检测的计算并行度也会增大。计算资源的数量一般远远小于数据集中子序列的数量，时间序列异常检测在计算任务方面的加速比主要受到计算资源数量与性能的限制，计算资源性能越强，数量越多，计算任务加速比越高。

6.2.2 通讯需求的并行化

除了计算需求的并行化之外，传统的 Amdahl 定律忽视了算法通讯需求与计算平台对于算法整体加速比的影响[73]。当时间序列的规模相对较小时，串行的异常检测方法将所有数据存储在内存中，不需要考虑数据传输对算法性能造成的影响。当时间序列的规模变大时，数据不能容入单个计算节点的内存，数据需要分布存储在不同的计算节点，为了保证分布式异常检测结果的正确性，计算节点之间必需通讯。当通讯需求足够大时，检测方法的加速比不仅仅取决于检测方法并行度与计算节点的数量和性能，还取决于检测方法的通讯需求以及计算节点之间的互连。

并行化时间序列异常检测的通讯需求与计算节点的数量有关。由于时间序列数据集分布存储在多个计算节点中，为了寻找一个子序列的最近邻，该子序列需要访问所有的计算节点。子序列相继访问不同的计算节点可以看作是由两两计算节点之间进行子序列的传输，计算节点间的通讯需求总量会随着计算节点数量的增加而增加。因此，计算需求的加速比的增加会使通讯需求的加速比下降，这是异常检测方法并行化问题的主要矛盾。

本文首先从互连的角度评估通讯需求并行化可行性。拓扑结构是影响互连的主要因素。而拓扑结构的确定与子序列访问计算节点的顺序有关。互连设计的主要思想是，在具有通讯需求的两个计算节点之间建立互连，在没有通讯需求的计算节点之间不建立互联，在计算节点之间的互联应该尽量避免使用星型互连或总线互联，影响并发通讯带宽，也应尽量避免使用全互连方式，降低互连利用率。为提高通讯效率与互连利用率，本文使用环状互连拓扑结构。

如图 6-1 所示，所有计算节点以环的形式连接在一起，相邻两个计算节点之间的互

连是单向的。这种结构的优点有以下两点：1）当计算节点的数量增加时，互连的数量以及并发带宽成线性比例增加；2）并行化异常检测方法的通讯需求能适应这种拓扑结构，即所有子序列以固定的顺序访问所有计算节点。例如，当一个计算节点找到该子序列在本地的最近邻后，将其传递给下一个计算节点，直到该子序列遍历所有的计算节点，则停止对该子序列的传输；3）所有具有互连的计算节点之间的通讯可以并发完成。

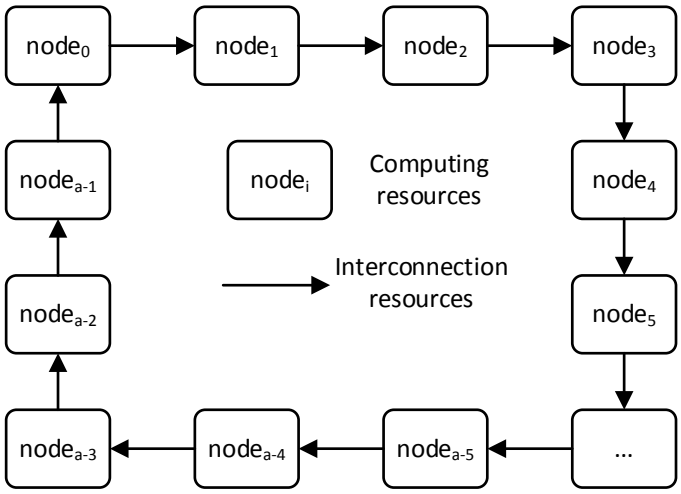

图 6-1. 计算节点的互连拓扑结构

Fig. 6-1 The topology of the interconnections among computing nodes

该互连结构改善了通讯效率，使得通讯时间取决于单次传输数量的数量，然而并行化检测方法的整体加速比还与计算需求与通讯需求的比例有关，接下来我们将量化分析并行化检测方法的计算通讯比。

6.2.3 计算通讯比

为了量化分析并行化异常检测方法的计算需求与通讯需求的比例，我们需要具体分析每一个子问题的计算需求与通讯需求。

- 计算需求

计算需求主要为在时间序列的分段中寻找给定子序列的最近邻距离。该计算需求与

分段的大小有关，同时也与搜索方式有关。若使用线性搜索法，即比较候选子序列与分段内所有子序列的距离，这样做计算复杂度过高。若使用经典的时间序列索引技术，可大大减少计算需求。例如使用 iSAX 索引技术，在大小为一百万的随机时间序列中搜索给定子序列的最近邻，至多需要调用 100 次距离函数[76]。

距离函数的 CPU 时间开销可由表 6-1 表示：

表 6-1 归一化操作与欧式距离函数的 CPU 时间开销

Table 6-1 The CPU time consumption of normalization and Euclidean distance function

	Add/Sub	Mul	Sqrt
Delay in Intel 64-bit instruction set[74]	1 cycles	10 cycles	66-80cycles
Cycles for Euclidean dist(C_p, C_q)	2n-1	n	1

其中 n 为滑动窗的大小。假设 CPU 的时钟频率为 3GHz，所有种类的计算都可流水化，且所有数据都存放 cache 中，那么距离的计算时间与滑动窗的大小近似成正比。设滑动窗长度为 100，那么计算一次两个子序列之间的距离至少需要 0.4us。假设一次最近邻搜索需要完成 10 次距离计算，那么传输一个子序列需要 4us 的计算时间。

- 通讯需求

一个子序列在计算机之间的传输时间可以近似由以下公式表示：

$$time_{transmission} = time_{latency} + \frac{n}{bandwidth} \tag{6-2}$$

其中$time_{transmission}$表示总共的传输时间，$time_{latency}$表示以太网的延时，$\frac{n}{bandwidth}$表明总传输时间随着数据规模的增长而增长，随着带宽的增长而减少。

一般而言，企业级以太网交换机的延时在一微秒数量级[75]，此外，假设集群架设在万兆以太网内，若滑动窗的大小为 100，每个数值使用双精度浮点数（64 位）表示，则传输一个时间子序列的数据需要$\frac{64b\times100}{10Gb/s} = 0.64us$。因此每传输 1 个子序列所需要的时间至多为 2 微秒。

- 计算通讯比

下面计算计算时间与通讯时间的比值。传输长度为 100 的子序列需要至多 2us 的时间，一次传输 1 个子序列需要 4us 的通讯时间，因此计算通讯比为$\frac{4\text{us}}{2\text{us}} = 2$。传输时间与

计算时间处于同一个数量级。因此并行化检测方法不适合随机访问时间序列中的子序列。

我们可以利用线性扫描访问子序列的方法提高计算通讯比。由于相邻子序列之间存在重叠，线性扫描访问子序列可以节约大量通讯时间，以长度为 100 的滑动窗为例，传输前 100 个数值只能形成 1 个子序列，而接下来每传输一个数值，即可形成一个新的子序列。连续传输数值的数量越大，传输效率越是接近 1 子序列/每数值。

以长度为 100 的滑动窗为例，一次传输 299 个连续的数值可得到 299-100+1=200 个长度为 100 的子序列，假设传输时间约为 4us，那么计算通讯比为$\frac{4\text{us}\times 200}{4\text{us}}=100$。计算时间与传输时间相比高出两个数量级。由此我们可以得到结论，线性扫描访问子序列以及分布式存储的方法改善了并行异常检测方法的计算通讯比。

6.3 并行化检测方法

根据上一节分析，使用线性扫描方法的时间序列异常检测从问题分解以及计算通讯比的角度看是非常适合并行化的。然而独立求解每个子问题会使计算复杂度等同于基本检测方法的计算复杂度，即$O(m^2)$。我们还需要使用剪枝技术，进一步降低并行化异常检测的计算复杂度。

并行化的异常检测方法的伪代码如表 6-2 所示，其中输入 b 为批量处理子序列的数量。该检测方法主要由两部分构成，第一部分通过采样少量子序列的最近邻距离来估算时间序列异常的最近邻距离（第 3-4 行）；第二部分计算所有子序列的最近邻距离，每次计算 b 个子序列的最近邻距离（第 6 行），更新目前最大最近邻距离$bsfDist$（第 7-9 行），并且在计算过程中利用剪枝技术减少距离函数的调用次数（第 6 行）。该并行化检测方法的计算时间主要集中在估计异常的最近邻距离（第 3 行)和线性扫描（第 5 到第 9 行）。下面介绍检测方法的具体过程。

表 6-2 并行的时间序列异常检测

Table 6-2 Parallel discord discovery

1	[bsfPos, bsfDist] = Parallel Discord Discovery (T, b)
2	Initialize $\mathrm{bsfDist} = 0$, $\mathrm{bsfPos} = \mathrm{null}$ // best so far position and distance
3	Find the estimation $\tilde{d}$ of the nnDist of discord,
4	Update $\mathrm{bsfDist} = \tilde{d}$
5	For every b subsequences $C_p, \ldots, C_{p+b-1}$ of all subsequences
6	$i = \underset{i}{\mathrm{argmax}}\{\mathrm{nnDist}(C_{p+i}) \mid 0 \le i \le b-1\}$ // enhanced with early abandon
7	If $\mathrm{bsfDist} \le \mathrm{nnDist}(C_{p+i})$
8	Update $\mathrm{bsfDist} = \mathrm{nnDist}(C_{p+i})$
9	Update $\mathrm{bsfPos} = p + i$
10	Return [bsfPos, bsfDist]

6.3.1 估算异常的最近邻距离

如表 6-2 第 3 行所示，并行检测方法的第一步是估算异常的最近邻距离。对于异常的最近邻距离的估算越是精准，能通过剪枝技术省去更多距离函数的调用，从而减少计算时间。

现有的异常最近邻距离估算方法建立在单个计算机节点的基础上，不适用于分布式计算环境。例如使用 HOTSAX 方法或 WAT 方法为整个时间序列建立索引，并通过索引获取模式罕见的子序列；然而本章中时间序列被划分成若干个分段，存储在不同的计算机节点中，为分布式数据建立索引存在通讯方面的困难，因此上述串行方法无法适用于分布式环境下时间序列的异常检测。

本文提出适用于分布式时间序列的异常的估计方法（Distributed-time-series Discord Estimation, DDE）。DDE 的伪代码如表 6-3 所示：

表 6-3 分布式时间序列中异常最近邻距离的估算方法的伪代码。

Table 6-3 Pseudocode of Distributed-time-series Discord Estimation

1	$\tilde{\mathrm{d}} = \mathrm{DDE}\ (\mathbb{C},\ \mathbb{S})$
2	For each $\mathrm{C_p}$ in $\mathbb{C}$ // in parallel
3	Calculate $\mathrm{A_p}$ of $\mathrm{C_p}$
4	Group $\mathrm{C_p}$ by $\mathrm{A_p}$
5	Find $\mathrm{A}_{\tilde{\mathrm{d}}}$ that is the $\mathrm{A_p}$ with the smallest group of $\mathrm{C_p}$
6	For each $\mathrm{C_p}$ in $\mathrm{A}_{\tilde{\mathrm{d}}}$
7	For each S in $\mathbb{S}$ // in parallel
8	Calculate lNNDist of $\mathrm{C_p}$ in S
9	Calculate gNNDist of $\mathrm{C_p}$ by finding its min lNNDist
10	$\tilde{\mathrm{d}} = \max\{\mathrm{gNNDist}(\mathrm{C_p}) \mid \mathrm{C_p}\ \text{that can be approximated as}\ \mathrm{A}_{\tilde{\mathrm{d}}}\}$
11	Return $\tilde{\mathrm{d}}$

DDE 的输入由两部分组成，$\mathbb{C}$是所有子序列的集合，$\mathbb{S}$为时间序列的分段信息。DDE 的输出为对异常最近邻距离的估计$\tilde{d}$，如果实际异常的最近邻距离为$\mathrm{nnDist}(\mathrm{C_d})$，那么总是有$\tilde{d} \leq \mathrm{nnDist}(\mathrm{C_d})$。

如第 2 和第 3 行所示，DDE 首先计算所有子序列的近似表示，子序列C_p的近似表示记为A_p。近似表示的种类有很多（如 2.2.2 所示），任意一种符号化近似都可以在 DDE 中使用。形状相似的子序列具有相同的近似表示。DDE 根据子序列的近似表示对所有子序列进行分组（第 4 行）。每个分组中子序列的数量反映了该分组近似表示的常见程度或异常程度。DDE 在第 5 行找到子序列数量最少的一个分组$A_{\tilde{d}}$（或几个分组）。对于$A_{\tilde{d}}$分组中的每一个子序列C_p（第 6 行），DDE 在每一个时间序列分段 S 中找到C_p的最近邻距离$lNNDist(C_p)$（第 7 和第 8 行），并取最小值作为C_p的全局最近邻距$gNNDist(C_p)$（第 9 行）。最后将最大的$gNNDist(C_p)$作为全局范围内时间序列异常的最近邻距离的

估算值（第 10 行）。

$A_{\tilde{d}}$分组中子序列的数量会影响到估算的准确性。若时间序列具有周期性，那么$A_{\tilde{d}}$分组可能包含很多子序列；若时间序列由随机数生成，则$A_{\tilde{d}}$分组内子序列的数量可能过少，影响到估算的准确性。为了能够更准确地估算异常的最近邻距离，我们可以取包含最少子序列的多个分组，以获得足够数量的子序列。

DDE 可在分布式计算平台上并行完成。如第 2 到第 3 行中，不同子序列的近似表示的计算过程是相互独立的，因此可以由各计算节点并行计算完成，随后通过汇总找到较异常的近似表示（第 5 行）；如第 7 和第 8 行，不同子序列在不同分段内寻找最近邻距离的计算是相互独立的，因此也可以由各计算节点并行计算完成，随后通过汇总找到子序列的全局最近邻距离。

6.3.2 线性扫描

如表 6-2 第 5 到第 9 行所示，并行检测方法的第二步为计算所有子序列的最近邻距离，并更新$bsfPos$和$bsfDist$。这是并行检测方法中最耗时的部分。为了提高计算通讯比，我们每次传输一个数据块(bulk)并计算其中所有子序列的最近邻距离；为此减少一个数据块的计算时间，我们使用剪枝技术跳过具有足够近的邻居的子序列。

6.3.2.1 数据块的处理

首先我们从数据块的角度描述并行化的检测方法。处理每个数据块的伪代码如表 6-4 所示：

数据块扫描方法的输入由数据块$\mathbb{C}_b$，时间序列分段的集合$\mathbb{S}$以及当前异常的最近邻距离$bsfDist$组成。该方法通过计算若干子序列在所有时间序列分段中的最近邻距离，决定是否更新当前异常$bsfPos$及其最近邻距离$bsfDist$。

该方法首先初始化数据块$\mathbb{C}_b$中的每一个子序列C_p的最近邻距离（第 2 行），然后将这些子序列传输到每一个时间序列分段所在的计算节点（第 3 行），计算C_p在当前时间序列分段 S 中的最近邻距离（第 4 到第 7 行）。如果在遍历过程中发现C_p当前的最近邻

距离$nnDist[p]$小于全局的当前异常最近邻距离$bsfDist$（第 8 行），则可以把当前子序列C_p从数据块$\mathbb{C}_b$中去掉（第 9 行），并跳过当前子序列C_p的后续计算（第 10 行），启发式的访问顺序可以更好地发挥剪枝技术的作用（第 5 行）。当该方法计算了C_p与所有分段中其他子序列C_p的距离且没有通过剪枝技术退出时，则$nnDist[p]$为C_p在全局范围内的最近邻距离，且必有$nnDist \geq bsfDist$，因此需要将$bsfPos$指向当前子序列（第 11 到第 12 行）。由于并行化计算的原因，$bsfDist$可能由多个计算共享，因此需要在遍历完所有时间序列分段后对$bsfDist$进行更新（第 13 行）。

表 6-4 一个数据块的扫描方法

Table 6-4 Scanning method of a bulk of data

1	$[\text{bsfPos}, \text{bsfDist}] = \text{Bulk Scan}\ (\mathbb{C}_b,\ \mathbb{S}, \text{bsfDist})$
2	$\text{nnDist}[:] =$ positive infinity
3	For each S in $\mathbb{S}$
4	For each C_p in $\mathbb{C}_b$
5	For each C_q in S // sorted by heuristic ordering
6	If $(\lvert p - q \rvert < n)$ continue
7	If $\left(\text{nnDist}[p] > \text{dist}(C_p, C_q)\right)\text{nnDist}[p] = \text{dist}(C_p, C_q)$
8	If $(\text{nnDist}[p] < \text{bsfDist})$
9	$\mathbb{C}_b = \mathbb{C}_b / C_p$
10	continue to next C_p
11	If $(\text{nnDist}[p] \geq \text{bsfDist})$
12	$\text{bsfPos} = p$
13	$\text{bsfDist} = \text{nnDist}(\text{bsfPos})$
14	Return $[\text{bsfPos}, \text{bsfDist}]$

当进行数据块扫描时，需要把数据块$\mathbb{C}_b$及相关中间结果从一个计算节点传输至另一个计算节点（第 3 行），中间结果包括每个子序列的当前最近邻距离$nnDist[:]$。我们已

经在上一节中介绍了 DDE 方法，使得$bsfDist$足够大，并更频繁地触发剪枝技术（第 8 到第 10 行）。因此在访问第一个计算节点中的时间序列分段之后，$\mathbb{C}_b$中的大部分子序列被剪枝。每当访问一个计算节点之后，$\mathbb{C}_b$中的子序列数量将继续减少。因此该扫描方法在计算节点之间传输的数据块与中间结果的数量较小。

6.3.2.2 计算节点的并行计算

接着本文从计算节点或时间序列分段的角度描述并行化的检测方法。在每一轮扫描过程中，每一个计算节点将收到一个新的数据块，以及从其他计算节点发送过来的数据块及其中间结果。该计算节点要在这一轮扫描内计算这些数据块中所有子序列在当前时间序列分段内的最近邻距离，并将非空的数据块传输给下一个数据块，随后进入下一轮扫描。所有计算节点同步进入一轮扫描，先完成数据块的计算节点必须等待其他计算节点完成，才能进入下一轮扫描。

我们举一个例子用于说明计算节点处理数据块的顺序。本文首先将时间序列分成a个分段$\mathrm{S}_0 \ldots \mathrm{S}_{a-1}$，分别存储在计算节点$node_0 \ldots node_{a-1}$内。所有子序列被分成数据块$b_0, b_1, b_2 \ldots$，每个数据块中含有数量相同的连续的子序列。表 6-5 显示了计算节点处理数据块的时序。

表 6-5 计算节点处理数据块的时序

Table 6-5 The timing of scanning multiple bulk of data

	node_0（S_0）	node_1（S_1）	…	node_{a-1}（S_{a-1}）
R_0	b_0	b_1	…	b_{a-1}
R_1	$\mathrm{b}_a + \mathrm{b}_{a-1}$	$\mathrm{b}_{a+1} + \mathrm{b}_0$	…	$\mathrm{b}_{2a-1} + \mathrm{b}_{a-2}$
R_2	$\mathrm{b}_{2a} + \mathrm{b}_{2a-1} + \mathrm{b}_{a-2}$	$\mathrm{b}_{2a+1} + \mathrm{b}_a + \mathrm{b}_{a-1}$	…	$\mathrm{b}_{3a-1} + \mathrm{b}_{2a-2} + \mathrm{b}_{a-3}$
…	…	…	…	…
R_{a-1}	$\mathrm{b}_{a(a-1)} + \sum_{i=0}^{a-2} \mathrm{b}_{ai+i+1}$	$\mathrm{b}_{a(a-1)+1} + \sum_{i=0}^{a-2} \mathrm{b}_{ai+i+2}$	…	$\mathrm{b}_{a(a-1)+a-1} + \sum_{i=0}^{a-2} \mathrm{b}_{ai+i+a}$
…	…	…	…	…
R_t	$\mathrm{b}_{at} + \sum_{i=t-a}^{t-1} \mathrm{b}_{ai+i+1}$	$\mathrm{b}_{at+1} + \sum_{i=t-a}^{t-1} \mathrm{b}_{ai+i+2}$	…	$\mathrm{b}_{at+a-1} + \sum_{i=t-a}^{t-1} \mathrm{b}_{ai+i+a}$

为计算节点分配新数据块：本检测方法在每轮扫描开始时为每个计算节点分配一个新数据块，该数据块先前未被传输到其他计算节点中，其中的所有子序列的最近邻距离

刚被初始化为正无穷（表 6-4 的第 2 行）。例如计算节点$node_{a-1}$分别在$R_0, R_1, R_2, \ldots$被分配新的数据块$b_{a-1}, b_{2a-1}, b_{3a-1}, \ldots$，直到所有新数据块都已被分配为止。

计算节点之间的数据传输：每个数据块都需要被传输到各个计算节点，以计算数据块中每个子序列的全局最近邻距离。当一个计算节点（如$node_0$）在计算完一个数据块（如b_0）中所有子序列在一个分段（如S_0）范围内的最近邻距离之后，需要把该数据块传输给下一个计算节点（如$node_1$）。计算节点之间数据块的传输路径可以有很多选择。我们选择使用旋转式移位(Rotate)在计算节点之间传输数据块。例如：以第 1 轮扫描过程T_1中$node_1$内的数据块$b_{a+1} + b_0$为例，在T_2时刻，这些数据块会被传输到$node_2$，在T_{a-1}时刻会被传输到$node_{a-1}$，由于此时b_0已经遍历了所有的计算节点（或时间序列分段），所以不会被继续传输到其他计算节点。在T_a时刻，剩下的b_{a+1}会被继续传输到$node_0$。

每个计算节点在任一时间刻最多需要处理a个数据块，例如$node_{a-1}$在T_t时需要处理的数据块为$b_{at+a-1} + \sum_{i=t-a}^{t-1} b_{ai+i+a}$，其中第一项$b_{at+a-1}$为新数据块，其余几项$\sum_{i=t-a}^{t-1} b_{ai+i+a}$为来自其他计算节点的数据块。然而，随着数据块遍历更多的时间序列分段，数据块中的子序列数量因剪枝技术（表 6-4 的第 9 行）而减少。因此，与第一项相比，计算其余几个数据块所需要的时间较少。

6.3.3 改善计算资源利用率

计算节点的差异与数据块的差异会造成负载均衡问题。不同计算节点处理不同数据块所需的时间可能不同，因此在同一轮扫描过程中有些计算节点在处理完其数据块之后会因等待其他计算节点而闲置，闲置时间会降低计算资源的利用率。

我们通过**多发射**，即分配额外的新数据块以及使用共享输入队列的方式缓解这一问题。在每一轮扫描过程中，并行检测方法准备数倍于计算节点数量的数据块，并通过共享输入队列进行缓冲。当有计算节点处于空闲状态时，就从共享输入队列获取一个数据块进行处理。当共享输入队列中所有的数据块都已被处理时，本轮扫描结束。

6.3.4 JDD、MDD 与 PDD

为了同时改善超大规模时间序列异常检测的检测效果与检测性能，我们可以结合

JDD 与 PDD，即将 J 距离作为衡量时间序列异常程度的标准或基本方法，使用 PDD 检测超大规模时间序列中的 J 距离异常。JDD 与 PDD 的结合方式与 APDD 类似，主要在于通讯需求的改变。PDD 在使用最近邻距离作为衡量子序列异常程度的标准时，若当一个计算节点找到子序列在本地时间序列的最近邻，只需要将该子序列与其最近邻距离传输给下一个计算节点即可以，当使用 J 距离作为衡量子序列异常程度的标准时，每个计算节点在找到本地的 J 近邻距离后需要将本地的 J 近邻列表一起传输给下一个计算节点，保证当子序列访问过所有计算节点后能得到正确的全局 J 近邻距离。

当时间序列的规模变大时，MDD 方案可以结合 DR-LOF 方法与使用 JDD 异常定义的 PDD 方法，达到同时改善检测效果与检测性能的目的。具体结合方式与 APDD 类似：首先利用 LOF 和 DR-LOF 找出对于异常贡献程度最高的维度，该过程起到数据降维的作用；随后选出最异常的维度，使用 PDD 检测该维度中的异常时间序列。

6.4 经验评估

为了检测 PDD 的实际性能，本文使用 Apache Spark[78]实现了 PDD。Apache Spark 是当前流行的大数据处理模型。Spark 是基于内存的集群计算框架，保留了 MapReduce [77]的可扩展性、容错性，又弥补了 MapReduce 在迭代式机器学习算法和数据挖掘等应用方面的不足，Spark 在一些典型应用方面比 MapReduce 快 100 倍。本文利用了 Spark 基于内存计算的特点对异常检测进行加速。

我们搭建了具有 10 个处理器核心的 Spark 集群，每个处理器核心配有 512MB 内存。该集群由 hadoop 分布式存储系统提供时间序列相关数据的访问服务。PDD 的所有部分均由 Scala 与 Java 的程序混编而成。实验过程中使用的时间序列数据集使用随机数生成。

6.4.1 DDE 的准确性

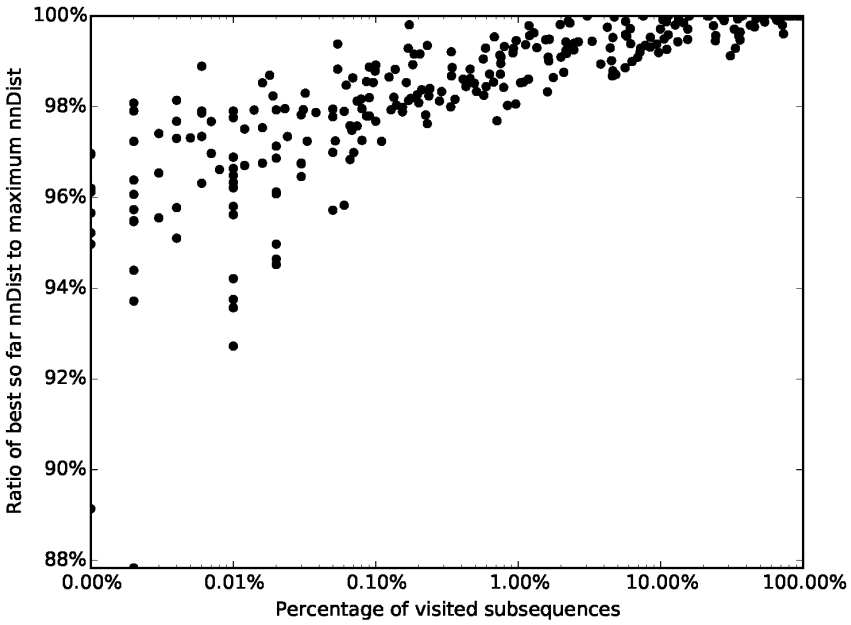

图 6-2. DDE 估算方法的准确性

Fig. 6-2 The accuracy of DDE method

首先我们评估 DDE 方法的准确性。我们生成 10 个大小不一的时间序列，数据集的规模从10^5到10^6不等，按照 DDE 方法采样不同数量的子序列，并计算这些子序列的最大最近邻距离。

图 6-2 展示了 DDE 估算方法的准确性。其中 x 轴表示采样子序列数量占整个数据集的比例，y 轴表示样估算最近邻距离$nnDist(C_{\tilde{d}})$与实际全局最近邻距离$nnDist(C_d)$的比例。通过图 6-2 我们可以看到，随着采样数量的增加，DDE 估算的精确性会逐渐增加。当采样比例超过 0.01%时，使估算距离$nnDist(C_{\tilde{d}})$达到实际距离$nnDist(C_d)$的 92%以上。因此，当数据规模达到10^6个子序列时，使用 DDE 方法采样 100 个子序列所得估算距离与实际距离的误差在 10%以内。

6.4.2 可扩展性

我们从并行度以及数据规模两个方面考查 PDD 的可扩展性，并且比较了经典的 HOTSAX 方法与 PDD 的计算时间。为了达到更好的可比性，我们同样使用 Java 实现了 HOTSAX 方法，使用 Java 虚拟机在相同配置的计算平台上运行。由于 HOTSAX 方法是众多时间序列异常检测方法的比较对象，因此将 HOTSAX 作为基准比较方法使得实验结果更具有可比性。

表 6-6 PDD 的可扩展性

Table 6-6 Scalability of PDD

Data size	HOTSAX time	PDD time	#node	Speedup	Speedup/node
1×10^5	3374s	2184s	2	1.54	0.77
		1190s	4	2.84	0.71
		761s	6	4.43	0.74
		570s	8	5.92	0.74
		500s	10	6.75	0.68
2×10^5	4.0h	2058s		7.00	0.7
4×10^5	11h	1.6h		6.88	0.69
6×10^5	28h	3.9h		7.18	0.71
8×10^5	60h	8.0h		7.5	0.75
1×10^6	82h	10.2h		8.04	0.80
1×10^7	NA	111h		NA	NA

我们首先比较在相同数据规模下并行度对 PDD 计算时间的影响。如表 6-6 的上半部分所示，数据集为包含1×10^5个点的随机时间序列（第一列），HOTSAX 方法检测该时间序列中的第一个异常需要使用 3374 秒（第 2 列）。随后我们使用 PDD 检测数据集中的第一个异常，使用计算节点的数量分别为 2、4、6、8、10 个（第 4 列），时间序列分段的数量与计算节点的数量相同。PDD 的计算时间随着计算节点的增加从 2184 秒减少到了 500 秒（第 3 列），PDD 相对于 HOTSAX 的加速比随着计算节点的增加从 1.54 增加到 6.75（第 5 行）。单位计算节点获得的加速比如最后一列所示，平均每个计算节点可以获得大约 0.7 的加速比。然而，随着计算节点或分段数量的增加，每计算节点加速比呈现出微弱的下降趋势。这主要是因为，若把时间序列分成更小的分段时，PDD 更不易在少数分段中找到足够近的邻居，更不易触发剪枝技术和放弃当前候选子序列。因此该实验数据表明，当数据规模不变时，更多的计算节点可以带来更高的加速比，但是

单位计算节点带来的加速比会随之降低。

随后，我们比较在相同并行度情况下不同数据规模对 PDD 计算时间的影响。我们使用 HOTSAX 检测不同规模的数据集（第 1 列，从1×10^5到1×10^6）中第一个异常，HOTSAX 的计算时间从 3374 秒快速增长至 82 小时（第 2 列），对于含有1×10^6个点的时间序列来说 82 小时的计算时间过长。与 HOTSAX 方法相比，PDD 使用 10 个计算节点时（第 4 列），将计算时间至 500 秒到 10.2 个小时（第 3 列），加速比约为 7（第 5 列）。从该表可以看出，随着数据规模的增加，单位计算节点的加速比显示出微弱的增长趋势。这是由于当数据规模增长而分段数量不变时，每个分段的长度增加，PDD 更易从少数分段中找到与当前候选子序列足够相似的邻居，更易触发剪枝技术并跳过候选子序列，从而达到减少时间复杂度的目的。因此该实验数据表明 PDD 在数据规模方面具有良好的扩展性。

最后，我们比较了 PDD 与 HOTSAX 在数据规模为1×10^7时的计算性能。我们使用了 Java 虚拟机默认的内存容量 512MB 执行 HOTSAX。如表 6-6 最后一行所示，HOTSAX 由于内存不足而无法正常运行。然后由于 PDD 使用了多台具有 512MB 内容的 Java 虚拟机，且将时间序列分段并分布式存储在这些 Java 虚拟机上，所以 PDD 能够被正常执行。这一实验结果说明 PDD 缓解了单台计算机内存容量对于超大规模时间序列异常的限制。

6.4.3 计算资源利用率

下面我们评估 PDD 在计算资源利用率方面的性能。PDD 在每一轮扫描之间需要同步全局范围内的当前异常的最近邻距离（表 6-2 中的$bsfDist$），同步之前要求所有计算节点完成本轮扫描的所有计算。有些计算节点提早完成计算，因等待其他计算节点而闲置。闲置造成计算资源的利用率低下。我们部署了 10 个计算节点，在规模为1×10^5的时间序列中检测第一个异常，并通过实验观察多发射方法对于计算资源利用率的改善效果。我们将每一轮分配的数据块的数量表示为 BPR（number of Bulks Per Round）。

表 6-7 显示了在不同 BPR 的情况下（第 1 列）PDD 的执行时间（第 3 列）和计算

节点的闲置时间（第 4 列）。第一行中，PDD 在每轮分配 10 个数据块，即每个计算节点一个数据块，闲置时间占到了 PDD 总时间的 18.9%。随着 BPR 的增加，闲置时间也随之下降。当$\mathrm{BPR}=500$时，闲置时间降至 4.71%，PDD 的总计算时间也从 580 秒降到了 500 秒。

表 6-7 计算资源的利用率

Table 6-7 Utilization of the computational resources

Bulk/round (BPR)	#subsequences/bulk	Total time	Idle time	Computation/running ratio of PDD	Computation/running ratio of Yankov [10]
10	200	580s	18.9%	96%	≤ 50%
100		527s	6.25%	95%	
500		500s	4.71%	95%	
1000		547s	3.04%	94%	

BPR 的数量不宜过大。如最后一行所示，当每轮分配数据块的数量从 500 升至 1000 时，PDD 的总运行时间从 500 秒升到了 547 秒，主要原因是剪枝技术未能充分起到作用。在每轮分配 1000 个数据块的情况下，PDD 每轮将分配$1000\times 200=2\times 10^5$个子序列，而大小为$1\times 10^5$的时间序列中仅有 99901 个子序列，PDD 将在一轮扫描内处理所有的子序列，这不利于剪枝技术发挥作用。因此，在设置 BPR 的时候应该考虑数据集规模与 BPR 之间的关系，使得剪枝技术的优势能得到充分的发挥。

表 6-7 的最后一列显示了计算节点的计算时间占到所有节点的总运行时间（除去闲置时间）的平均比例。由于计算所需要的数据都在内存中，所以计算时间占总运行时间的比例在 95%左右。其他时间用于完成 Spark 计算平台的后台工作，如调度、任务反序列化、Java 垃圾回收、计算节点间的数据传输等。与磁盘感知的异常检测方法[10]相比，计算资源的利用效率提高了将近一倍。

6.5 本章小节

本章提出了并行化时间序列异常检测方法（Parallel Discord Discovery，PDD），用于减少磁盘操作对于异常检测计算时间的影响，缓解单台计算机内存容量对数据集规模的限制。本章首先从数据依赖关系和计算通讯比方面考查了并行化可行性，然后设计了适

合于分布式计算框架的基于内存的时间序列异常检测方法 PDD。本章通过经验评估验证了 DDE 方法能通过采样 0.01%的子序列估算出与实际异常最近邻距离误差 8%以内估计距离。在拥有 10 个计算节点的情况下，PDD 能够实现 7 倍于 HOTSAX 在 1 个计算节点上的检测速度，且能够处理 HOTSAX 在单机环境下无法处理的更大规模的数据集。实验表明计算时间占到总运行时间的 90%以上，缓解了由于磁盘读写问题对于检测性能的影响。

本章节的相关论文已经被数据挖掘领域的国际知名会议 PAKDD 2016 录用，本章节中，互连感知的并行化可行性分析方法已在高性能计算领域的著名国际期刊 Supercomputing（影响因子: 0.858）上发表[73]。

第七章 近似的并行化时间序列异常检测方法

本章研究近似的并行化时间序列异常检测方法，通过近似的检测方法，降低时间序列异常检测的计算复杂度，并行并行化检测过程，缩短检测时间。本章还着重研究本方法下检测结果与原始定义检测结果的一致性问题，使检测方法在提高速度的情况下保证检测结果基本正确。

7.1 研究背景

超大规模时间序列的异常检测存在计算复杂度方面的困难，且由于时间序列异常定义本身的特点，检测问题不能直接通过分治法进行加速。

7.1.1 计算复杂度问题

时间序列异常检测的计算复杂度过高，它需要计算所有子序列之间的距离。若时间序列的长度为 m，则时间序列异常检测的计算复杂度为$O(m^2)$。虽然 HOTSAX 方法能将时间序列异常检测的计算时间降低多达 3 到 4 个数量级，但是依然不能降低$O(m^2)$的计算复杂度。为了验证这一事实，我们生成随机数数据集使用 HOTSAX 检测其中长度为 100 的第一个异常，并用距离函数的调用次数衡量 HOTSAX 的计算复杂度。

如图 7-1 展示了数据集规模与距离函数调用次数之间的关系。距离函数调用次数与数据集规模的二次项近似成正比。事实上，与时间序列异常检测有关的文献中涉及的绝大部分数据集的规模都在 64K 以下，这些文献回避了检测方法在数据规模方法的问题。当数据集规模超过百万时，现代处理器需要至少数小时检测第一异常。例如，一个典型的心律失常检测应用要求心电信号的时钟频率为 360Hz，许多情况下病人会被要求整天佩戴心率监测设备记录心电信号，一天一人的心电信号数据规模高达 31M 个采样点，HOTSAX 将花费数天时间完成第一异常的检测，检测时间过长。因此我们认为 HOTSAX 并不适合检测海量数据的异常。

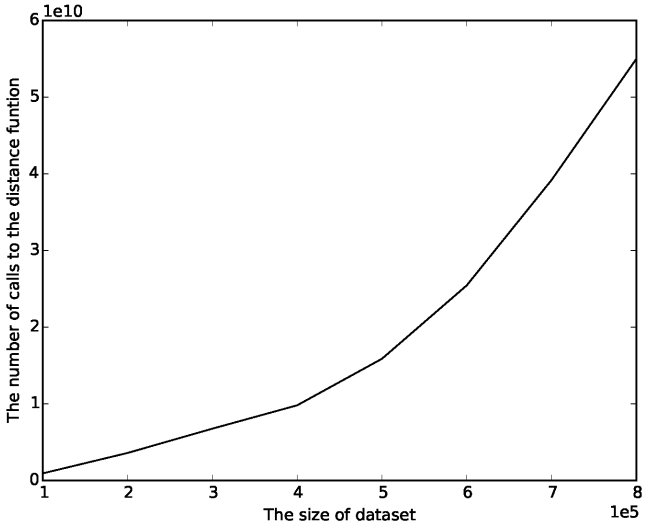

图 7-1. 随机数数据集规模与距离函数调用次数的关系

Fig. 7-1 The size of dataset to the number of calls to the distance function

Yankov 等人[10]提出了前沿的硬盘感知的异常检测方法，虽然该方法只需要两次线性读取数据集即可找到异常，然而它并未降低异常检测的计算复杂度。根据[10]提供的数据我们可以得到如下计算复杂度的相关信息。

表 7-1 硬盘感知的异常检测方法的计算复杂度[10]

Table 7-1 The computational complexity of disk-aware discord discovery method [10]

#sequence	Time on computation	Time on disk I/O
1 million	840 s (14 min)	27 min
10 million	12120 s (3 h 22 min)	4 h 30 min
100 million	163980 s (45 h 33 min)	45 h

表 7-1 展示了数据集规模与计算时间的关系。数据规模#sequence 为[10]生成的随机数时间序列的数量，而 Time on computation 为硬盘感知的异常检测方法在纯粹计算方法下所花费的时间。通过该表我们可以发现，数据规模以 10 倍的速度增长着，而计算时间却以大于 10 倍的速度更快地增长着。虽然该方法只需要两次线性读取数据集即

可找到异常，但是数据规模与计算时间依然成$O(m^2)$关系。

因此我们致力于提出一种并行化的时间序列异常检测方法，通过降低异常检测的计算复杂度改善时间序列异常检测在数据规模方面的可扩展性。

7.1.2 正确性问题

时间序列异常不能被分解为互相独立的子问题[15]。这意味着我们不能通过常用的分治法改善时间序列异常检测在数据规模方面的可扩展性。分治法的计算结果可能与事实有出入。我们将分治法结果与原定义检测结果不一致的问题称为“**正确性问题**”。

以图 7-2 为例，该图展示了一条完整的时间序列，依照分治法的思想将该时间序列分成两半，即S_1和S_2，分别检测其中的异常。从图中可以直观的看出，C_a和C_c分别是S_1和S_2中的第一异常。如果按照分治法的思想，那么全局的第一异常将在C_a和C_c之间选出。然而从全局的角度来看，C_a和C_c非常相似，它们不应该是全局最异常的子序列。在S_1中的C_b才是事实上的全局第一异常，使用分治法存在无法捕获正确结果的可能性。

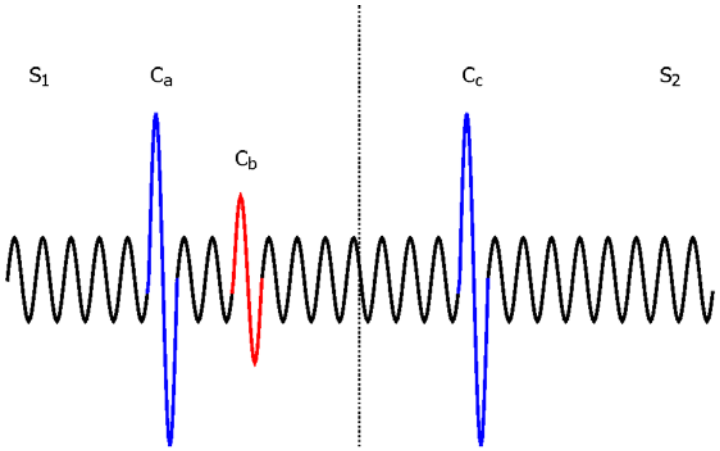

图 7-2. 一个全局第一异常不是局部第一异常的例子

Fig. 7-2 A counter-example where top global discord is not top local discord

这一问题在[15]中被称为“异常检测结果是不可合并的”。当数据规模较大时，局部第一异常在全局更有可能找到与其相似的子序列，因此发生上述情况的可能性更大。我们需要在新的检测方法中解决此类问题。

7.2 近似的并行化时间序列异常检测方法

时间序列基本检测方法需要调用大约m^2次距离函数，其中 m 是时间序列的长度，如果将时间序列分解为a段并且分别检测其中的异常，那么总共只需要调用$\frac{m^2}{a}$次距离函数，并且可以通过并行计算加快检测速度。从一个分段中检测出的异常是该分段内最不寻常的子序列，我们称之为本地异常(local discord)。相对的，我们将直接从整条时间序列中检测出的异常称为全局异常(global discord)。本地异常拥有在其分段内最大的最近邻距离。而全局异常在整个时间序列范围内拥有最大的最近邻距离。

分别检测每一段中的异常可能会遇到结果的正确性问题，因为本地异常未必拥有全局最大的最近邻距离。但是通过实验我们得到如下两个观察：

1. 全局第一异常很有可能是本地第一异常；
2. 一个分段中频繁出现的子序列较不可能是全局第一异常。

相关文献中用来展示时间序列异常定义的用途所涉及的大部分示例都符合以上两个观察。这意味着我们可以通过分治法轻易地将全局第一异常找出来，同时减少计算时间，并且通过并行化进一步减少计算时间。在每个分段中找到本地第一异常之后，我们通过遍历所有的本地异常选出具有最大本地最近邻距离的子序列，将其作为近似的全局第一异常。

然而上述观察不总是成立，例如全局第一异常不是本地第一异常。事实上可以通过检测更多的本地异常来解决这一问题，如检测本地第二异常或第三异常。在最糟糕的情况下，我们需要计算每个分段中所有子序列的最近邻距离以确保能够找到全局第一异常。最糟糕情况下并行化方法的计算复杂度将仅比基本检测方法(brute force method)快a倍，其中a为分段数量。然而根据第二条观察发现这种情况的概率的可能性是很低的，所以并行化方法将远远快于基本检测方法。

我们将该方法称为近似的并行化时间序列异常检测方法(Approximated Parallel Discord Discovery, APDD)，因为 APDD 不能检测出与原始定义保持一致的异常。该方法并未完全按照时间序列异常的定义进行异常检测。它为了保证本地异常之间不存在重叠

情况而放弃了某些子序列对之间距离的计算。例如，若本地第一异常与全局第一异常存在重叠，而该本地第一异常在分治法合并过程中未被识别为近似的全局第一异常，那么近似的全局第一异常可能与事实的全局第一异常不一致。然而根据上述的两条观察，这种情况在真实世界数据集中出现的可能性较小，APDD 在多数情况下能找到事实上的全局异常。

7.2.1 流程

在介绍了 APDD 的主要思想和原理之后，我们深入介绍 APDD 的具体实现方法。图 7-3 展示 APDD 的工作流程图。

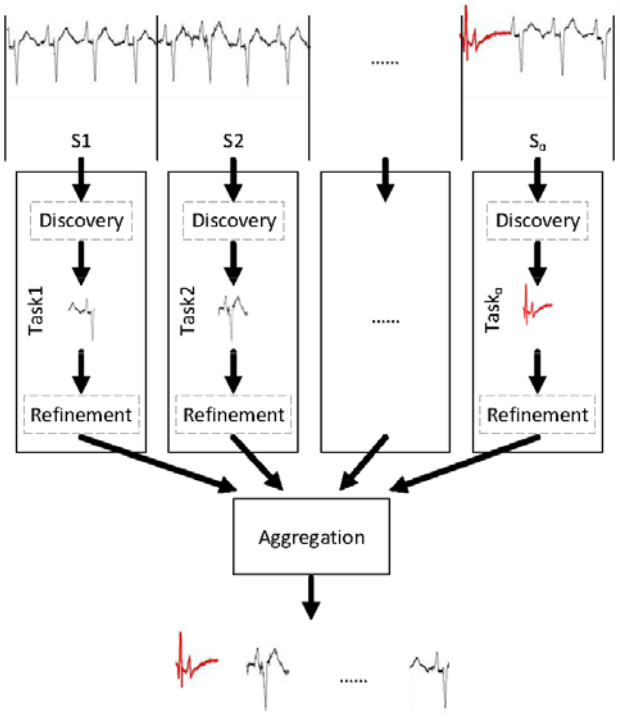

图 7-3. 近似的并行化时间序列异常检测方法的流程图

Fig. 7-3 Workflow of Approximated Parallel Discord Discovery

首先将时间序列划分成等长的分段，划分之后的分段是$S_1, S_2, \ldots, S_a$，其中a为分段的总数。如图 7-3 所示，异常可能出现在划分边界上，简单的划分可能会影响异常检测的正确性。为了避免这一情况的发生，我们扩展每个分段，使得任意两个相邻分段之间都

有一段长度为$n-1$的重叠，其中 n 为滑动窗的长度。

然后，我们将每个分段建立一个任务，每个任务由两步组成，即异常检测与验证。任务的伪代码如表 7-2 所示。任务的第一步是找出当前分段是的第一异常（第 2 行）。找异常的方法可以是任意时间序列异常检测方法，例如基本检测方法或者 HOTSAX。本地异常C_p拥有该分段中最大的最近邻距离，其中 p 是该异常在分段中的起始位置。任务的第二步是验证本地异常在全局范围内的最近邻距离。在这一步中，当前任务首先将C_p发送给其他任务（第 5 行），向其他任务查询C_p在其他分段中的最近邻距离（第 6 行），随后将C_p的本地最近邻距离更新为全局最近邻距离（第 7-9 行）。

表 7-2 任务的伪代码

Table 7-2 The pseudocode of a task

1	Function [discord, global nnDist]= Task(segment)
2	Find local discord C_p from segment
3	bsfNNDist = nnDist(C_p) //best so far nnDist
4	For each of the other tasks, labeled b
5	Send C_p to b
6	Receive b's nnDist(C_p)
7	If (bsfNNDist > b′s nnDist(C_p)
8	bsfNNDist = b′s nnDist(C_p)
9	End If
10	end For
11	Return[C_p, bsfNNDist]

最后，我们将找到的本地异常汇总，根据它们的全局最近邻距离的降序列排序，排名第一的本地异常即为近似的全局异常。

7.2.2 计算复杂度

APDD 的计算复杂度主要集中在可并行的任务中，受到并行度a的影响，a也就是分段或任务的数量。为了获得更好的加速比，我们对计算复杂度与并行度a的关系进行分析。

距离函数的调用次数是衡量计算复杂度的良好方法。我们将距离函数的调用次数记为 F。F 是一个与并行度a有关的函数。$F(a)$可以近似表示为：

$$F(a) \approx \left(\frac{m}{a}\right)^2 \times a + m \times a。 \tag{7-1}$$

$F(a)$由两项构成。第一项表示在所有分段中使用基本检测方法(brute force method)检测异常所需要的计算工作量。第二项表示验证所需要的计算。当a很小时，第一项主导$F(a)$。随着时间序列被分成更多的分段，第二项逐渐变大并主导$F(a)$。有一个关于a的最优解使得$F(a)$最小。为了找到a的最优解，我们求$F(a)$关于 a的一阶导数的零点：

$$\frac{\mathrm{d}F(a)}{\mathrm{d}a} = 0, \tag{7-2}$$

求得零点为：

$$a = \sqrt{\mathrm{m}}。 \tag{7-3}$$

如(7-3)所示，当$a = \sqrt{\mathrm{m}}$，$F(a)$有其最小值：

$$F(a)|_{a=\sqrt{\mathrm{m}}} \approx 2m^{\frac{3}{2}}。 \tag{7-4}$$

因此当满足上述两个观察时，APDD 可以将时间序列异常检测的计算复杂度从$\mathrm{O}(\mathrm{m}^2)$降到$\mathrm{O}(\mathrm{m}^{\frac{3}{2}})$。值得注意的是，(7-1)以及后续等式的计算都建立在使用基本检测方法检测异常的基础上，更好的异常检测方法，如 HOTSAX，也可以用于各分段中本地异常的检测，进一步减少计算时间。

7.2.3 加强的近似的并行化检测方法

上述流程中每个任务只检测其分段中的一个异常。然后全局第一异常在一个分段中可能不是本地第一异常，而可能是本地第二、第三，甚至排名更加靠后的异常。根据上述两项观察，我们可以在每个分段中检测足够数据的本地异常以确保能够找到近似的全

局第一异常。然而我们事先并不知道应该在每个分段中检测多少异常。检测过少的本地异常可能会遗漏全局第一异常，检测过多的异常可能会引起过高的计算工作量。

我们需要提出一种能自动决定每个任务检测分段中本地异常的数量的方法。该方法的主要思想是：决定一个分段是否还有未发现的本地异常，并且这些本地异常可能是前 k 个全局异常。为了实现这一目标，我们需要对每个被检测出的本地异常进行测试。根据最近邻距离、本地异常和全局异常的定义，我们知道有如下两个事实：

1. 子序列C_p的本地最近邻距离不小于C_p的全局最近邻距离；
2. 第 i 个异常的最近邻距离不小于第 i+1 个异常的最近邻距离。

通过这两个事实我们可以得到如下**推论**：

- 当一个任务找到一个本地异常C_p，且C_p的本地最近邻距离小于第 k 个近似的全局异常，那么C_p一定不是前 k 个全局异常，并且不可能从该任务所在分段中找到前 k 个全局异常的任意一个。

该推论保证每个任务不会错失找到更多的前 k 个全局异常。该推论要求我们知道事实的前 k 个全局异常，这是我们在检测开始时不知道的。但是本地异常是全局异常的候选，我们在检测过程中找到的近似的全局异常会逐步逼近事实的前 k 个全局异常，因此我们可以使用当前的第 k 个近似的全局异常的全局最近邻距离作为任务是否停止的标准。根据该推论我们提出**自适应停止条件(Adaptive Stop Criteria, ASC)**：

- 如果一个任务找到了一个本地异常，该本地异常的本地最近邻距离小于当前第 k 个近似的全局异常的全局最近邻距离，那么任务停止；
- 否则，该任务继续寻找相应分段中的下一个异常。

ASC 让每个任务知道它们各自的停止运行的时刻。为了把 ASC 运行到 APDD 中，我们对 APDD 的流程做了改进。

ASC 被插入到每个任务中。一个任务在检测分段中的一个本地异常之后，检查该本地异常是否符合 ASC。如果满足 ASC，那该任务停止工作；如果不满足 ASC，那么该任务对当前本地异常进行全局验证，并检测下一个本地异常。

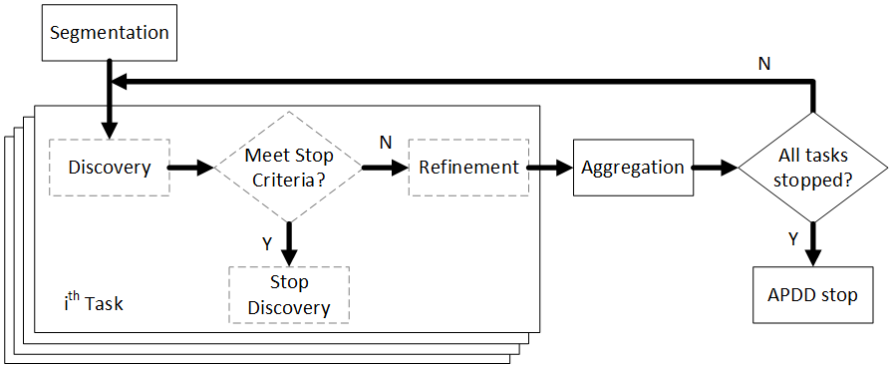

图 7-4. 带有自适应停止条件的 APDD 的流程

Fig. 7-4 The workflow of APDD with adaptive stop criteria

流程中的汇总阶段维护一个近似全局异常的列表，该列表保存了由每个任务检测出的各自的本地异常以及与其相关的全局最近邻距离。该列表中的每个本地异常都是最终全局异常检测结果的候选。ASC 使用该列表中排名第 k 的全局最近邻距离作为停止条件。每个一个任务检测验证完一个新的本地异常后，该本地异常都会用于更新汇总阶段的列表。

一个任务的停止意味着该任务不可能能从其分段内找到更多的前 k 个全局异常。当所有任务都停止时，APDD 方法终止。

在此流程中，任务与任务之间是相互独立的。当一个任务完成一次本地异常的检测与全局最近邻距离的验证后，它可以继续检测下一个本地异常，而不用等待其他任务完成它们当前的检测与验证工作。这一特性使得 APDD 在运行速度方面可以得益于以下两个方面：

- 对于一个含有前 k 个全局异常的分段，其相应的任务将更快地找到这些异常[15] 。对于不含有前 k 个全局异常的分段，汇总阶段将更早使用更有潜力的本地异常候选更新列表，从而相关任务将通过 ASC 更早地停止。因此 APDD 的计算时间可以进一步缩短。
- 计算资源的数量不需要与任务的数量等同。APDD 在计算资源的利用率方面非常灵活。例如将一个时间序列分成 64 个分段，使用 32 个计算资源进行异常检

测，64 个任务中的任意 32 个可以在 32 个计算资源上运行。如果一个任务因 ASC 停止，相应的计算资源可以随机抽取下一个任务进行计算。当所有任务停止时 APDD 终止。由于 ASC 保证结果的正确性，任务的运行顺序不会影响到结果的正确性。

7.2.4 JDD、MDD 与 APDD

到目前为止本文在描述 APDD 时，将最近邻距离指定为衡量时间序列异常程度的标准（criteria）或基本方法（kernel function）。为了同时改善超大规模时间序列异常检测的检测效果与检测性能，我们可以结合 JDD 与 APDD，即将 J 距离作为衡量时间序列异常程度的标准或基本方法，使用 APDD 检测超大规模时间序列中的 J 距离异常。具体的实现方式为：为每一个子序列维护一个数据结构，用于存放当前找到的该子序列的前 J 个最近邻，从而使 APDD 获该子序列当前的 J 距离。APDD 在验证结果正确性的阶段考察该序列的全局第 J 近邻距离时，首先在每个分段中找到该子序列的 J 个最近邻，然后在汇总时对检测结果进行排序，排除重叠的邻居，取前 J 个邻居并获得全局 J 距离。

当时间序列规模较小时，MDD 方案可以结合 DR-LOF 方法与 JDD 定义及其检测方法，改善时间序列异常检测的效果。当时间序列的规模变大时，MDD 方案可以结合 DR-LOF 方法与使用 JDD 异常定义的 APDD 方法，达到同时改善检测效果与检测性能的目的。具体结合方式为：首先利用 LOF 和 DR-LOF 找出对于异常贡献程度最高的维度，该过程起到数据降维的作用；随后选出最异常的维度，使用 APDD 检测该维度中的时间序列异常。

7.3 经验评估

本节我们通过实验对 APDD 的计算复杂度，正确性以及可扩展性进行经验评估。为了使实验结果具有可比性，我们将滑动窗的长度固定为$n = 360$。我们选用 HOTSAX 作为比较的基准方法，主要原因有如下两条：

1. 通过相关文献[20][48]提供的图表可以发现，HOTSAX 的计算时间相对于数据集

规模更不敏感。虽然这些文献声称其检测方法比 HOTSAX 更快，但是随着数据规模的增加，这些检测方法的计算时间的增长速度比 HOTSAX 更快，当数据规模到达 32K 或 64K 时，这些检测方法的计算时间与 HOTSAX 非常接近。在本章中我们使用的数据规模都超过了 64K，因此我们认为 HOTSAX 在这样的数据规模下能更好的代表前沿检测方法的性能。

2. 多数文献都以 HOTSAX 和基本检测方法作为基准方法。此外，HOTSAX 在实现方面更简单，较其他方法不易引入人为因素造成对计算复杂度的影响。为了更好的可比性，我们认为应该选择 HOTSAX 作为基准方法。

7.3.1 计算复杂度

之前我们已经分析过 APDD 的计算复杂度，下面我们通过测量 APDD 的计算时间来评估它的计算复杂度。我们使用 4 个具有代表性的真实世界数据，每个数据集中采样点的数量都大于 64K。我们使用 Java 实现 HOTSAX 与 APDD，并在一台装备有 Intel i5 的个人电脑上运行这些程序。APDD 的任务数量一律被设置为 64。我们使用一个线程运行 HOTSAX 和 APDD，并使用所有任务的总计算时间衡量 APDD 计算复杂度，总计算时间可以表示为：

$$Time_{APDD} = \sum_{i=1}^{a} Time_{task_i} \tag{7-5}$$

其中a是任务的数量，在这里$a = 64$。表 7-3 显示了 HOTSAX 和 APDD 为每一个数据集检测前 10 个异常的运行时间。

表 7-3 HOTSAX 与 APDD 在不同规模数据集情况下的运行时间

Table 7-3 Time consumption of HOTSAX and APDD

Dataset	Size(K)	HOTSAX(s)	APDD(s)	Speedup
Activity [60]	162	398	147	2.7
Buzz [72]	583	1661	197	8.4
Ecg102 [55]	650	2703	1148	2.3
House [71]	2049	20900	1448	14.4

表中的第二列描述了数据集的规模。第三第四列分别展示了 HOTSAX 和 APDD 的运

行时间。第五列是 APDD 相对于 HOTSAX 的加速比。在计算资源数量相同的情况下，APDD 相对于 HOTSAX 的加速比为 2.3-14.4 倍。当数据集规模较大时，APDD 相对于 HOTSAX 具有更高的加速比。

请注意 APDD 可以被并行化运行，但是我们在本章中仅使用一个计算资源运行 APDD，这使得实验能简单快速地验证 APDD 的有效性，而且实验结果具有更好的可重现性与可比性。APDD 的所有任务共享一个 CPU 线程的时间片。多个计算资源给 APDD 带来的加速比受到数据集的影响，由执行时间最长的任务决定。通过多次运行 APDD 并分析平均计算时间我们发现，不可并行的部分，即执行时间最长的任务，最多占到所有计算时间的 4%。这意味着如果计算资源的数量与任务数量相同，即为 64 时，APDD 能产生额外的$\frac{100\%}{4\%} = 25$倍的加速比[73]。也就是说，在给予充足数量计算资源的情况下，APDD 相对于 HOTSAX 将获得至少$25 \times 2.3 = 57.5$的加速比。

7.3.2 正确性

然后我们需要评估 APDD 的正确性，即 APDD 检测结果与原始定义检测结果的相似程度。如我们所知，时间序列异常(discord)不是一个二值属性。时间序列中的每个子序列都与至少一个异常重叠。异常的排名表明了子序列异常的程度。所以我们不能简单地通过检出率(detecting rate)与误报率(false alarm rate)来判断 APDD 的正确性。因此我们提出以下两个评估方法来衡量 APDD 的正确性。

检出得分（Discovering Score, DS）：用来表示检测方法找到的事实异常的数量以及排名的正确性。事实异常指的是根据原始定义及基本方法检测出的异常。DS 可以表示为：

$$DS_k = \sum_{i=1}^{k} \begin{cases} \frac{1}{i}, & \text{if } i^{th} \text{ reported discord is reported} \\ 0, & \text{otherwise} \end{cases} \tag{7-6}$$

其中 k 为希望检测出的异常的数量。

漏报得分（Missing Score, MS）：用来表示检测方法找到的假异常的数量以及排名。假异常即为未出现在根据原始定义及基本方法检测结果中的异常。MS 可以被表示为：

$$MS_k = \sum_{i=1}^{k} \begin{cases} \frac{1}{i}, \text{if } i^{th} \text{ reported discord is false} \\ 0, \text{otherwise} \end{cases} \tag{7-7}$$

DS 和 MS 不仅强调了异常的排名的重要性，同时也将检出率与误报率考虑在内。DS 的满分以及 MS 的 0 分表示了检测方法能够检测出与原始定义及基本检测方法结果完全一致的异常。

表 7-4 APDD 在不同数据集情况下的正确性

Table 7-4 The correctness of APDD on different datasets

Dataset	DS_{10}	MS_{10}
Ecg102	2.93	0.00
Activity	2.93	0.00
House	2.82	0.14
Buzz	2.93	0.00

我们用 APDD 检测数据集中的 10 个异常，相关参数设置与 7.3.1 保持一致。表 7-4 展示了 APDD 在 4 个真实世界数据集上的正确性。DS 与 MS 的满分为 2.93。通过该表我们知道 APDD 可以在 ECG，Activity 以及 Buzz 三个数据集上找到与原始定义和基本检测方法结果完全一致的异常。APDD 在 House 数据集上未能检测出完全正确的结果，在排名第 7 的位置报告了伪异常。总体而言，APDD 与原始定义和基本检测方法的结果非常接近。

7.3.3 任务数量的可扩展性

APDD 可以通过扩展任务数量缩短异常检测的计算时间。我们从计算时间与正确性两方面评估 APDD 在任务数量方法的可扩展性。我们使用 ECG102 作为测试数据集，因为它被广泛应用于相关文献中展示异常检测方法的效用。除任务数量以外，其他参数设置均与 7.3.1 保持一致。

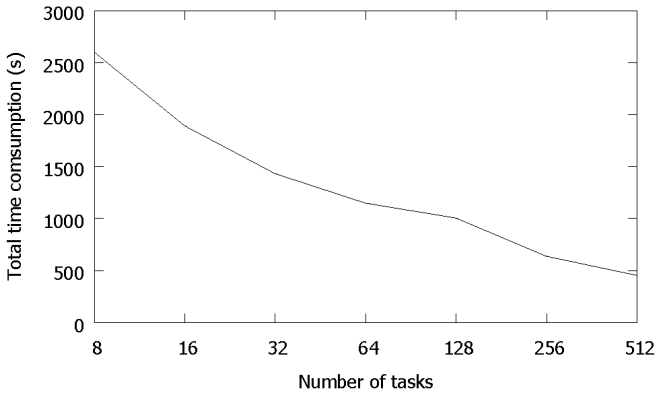

图 7-5. APDD 的计算时间对任务数量

Fig. 7-5 The number of tasks v.s. time consumption

图 7-5 展示了计算时间相对于任务数量的曲线，APDD 的计算时间随着任务数量的增加而逐渐减小，且下降速率越来越慢，这符合我们对 APDD 计算复杂度的预期。我们通过分析发现，最长任务执行时间占到总运行时间的 4%，因此至少有 96%的计算工作量是可以通过并行加速的。给于更多的计算资源，APDD 的计算时间将进一步减少。

表 7-5 展示了 APDD 的正确性与任务数量之间的关系。从表中可以看出，在任务数量从 8 到变化到 512 的范围内，DS_{10}都接近或者等于满分 2.93，而MS_{10}都接近或者等于 0 分。这意味着 APDD 在不同的任务数量下都能得到与原始异常定义和基本检测方法结果相近的结果，APDD 的正确性对任务数量不敏感。

表 7-5 APDD 的正确性对任务数量

Table 7-5 The number of tasks v.s. correctness

任务数量	DS_{10}	MS_{10}
8	2.93	0.00
16	2.83	0.10
32	2.93	0.00
64	2.93	0.00
128	2.79	0.10
256	2.93	0.00
512	2.83	0.10

7.4 本章小节

本章中我们提出了近似的并行化时间序列异常检测方法（Approximated Parallel Discord Discovery，APDD），用于减少时间序列异常检测的计算时间。APDD 将一个很长的时间序列分成若干份分别进行异常检测，最后通过将所有结果汇总得到近似的全局异常。APDD 降低了时间序列异常检测的算法复杂度，同时也允许使用多个计算资源对异常检测进行加速。给予足够数量的计算资源，设置有 64 个任务的 APDD 方法相对于前沿的 HOSAX 方法可以达到超过 50 倍的加速比，且其检测结果与原始定义和基本检测方法的结果基本一致。

本章提出的近似化异常检测方法的串行化版本已在高性能计算领域的国际会议 HPCC2015 上发表[123]。

第八章 全文总结

本章系统总结本文的核心工作和简要描述未来可能的工作计划。

8.1 工作总结

时间序列是描述事物发展情况的有效数据形式，而时间序列异常反映了事物的反常情况。随着传感器技术的发展，由传感器采集产生的时间序列的种类和规模都在不断地增加，如何快速有效地从各类时间序列中检测出异常是迫切问题。本文针对时间序列异常检测的两个核心挑战：1) 现有方法的检测效果无法满足超大规模时间序列异常检测的需求；2）现有方法在计算复杂度方面无法适应时间序列规模的快速增长，提出了一系列方法。核心创新点包括：

- 提出了完整的面向超大规模时间序列的异常检测系统，通过结合多种异常检测方法与方案，到达同时改善检测效果与检测性能的目的；
  - 提出了新的异常定义（J-distance Discord，JDD）以及相应的检测方法，将第 J 个最近邻距离(J-distance)作为判断子序列异常程度的标准，能够检测出多个形状相似的异常。相应的检测方法缓解了 JDD 带来的额外计算复杂度，使得 JDD 检测速度与经典检测方法 HOTSAX 相当；
  - 提出可用于异常溯源的多维时间序列异常检测方案（Multi-dimensional Discord Discovery，MDD），通过发现和选择多维时间序列中对异常贡献最大的维度，达到降低数据维度，降低时间序列异常检测计算时间的目的；
  - 提出了并行化时间序列异常检测方法（Parallel Discord Discovery），从数据依赖关系和计算通讯比的角度证明了异常检测的可行性，并在 Spark 分布式计算框架下实现了 PDD。
  - 提出近似的并行化时间序列异常检测方法（Approximated Parallel Discord

Discovery，APDD），将时间序列划分成多个分段分别进行检测，在不过度影响检测结果正常性的情况下，降低异常检测的复杂度，同时允许通过并行化计算对异常检测进行加速；

总之本文针对时间序列异常检测当前已有工作的难点和热点，从检测效果、计算复杂度两个方面都分别设计，实现了高效的解决方案，并通过丰富的实验结果证明了方案的有效性。

8.2 研究展望

本文的工作主要集中在没有专业知识的情况下检测静态的时间序列中的异常，为了改善时间序列异常检测在实际应用中的效果，拟开展以下工作：

1. 实时的时间序列异常检测方法

   本文研究的时间序列异常检测方法均为静态方法，即对历史中特定段落的时间序列进行分析并得出结果。静态的时间序列方法不能应用于实时的时间序列异常检测。然而在许多应用场景中时间序列是不断增长的，应用对于实时获得的时间序列中的异常的需求更加迫切，因此在本文的后续工作拟开展实时的时间序列异常检测的研究，研究内容包括 1）改进时间序列异常的定义，使定义适用于实时的异常检测；2）降低时间序列异常检测的算法复杂度 3）使用真实世界数据验证实时时间序列异常定义的有效性等等。

2. 交互式的时间序列异常检测方法

   本文研究的时间序列异常检测方法仅从数据的角度分析时间序列中的异常情况，这类检测方法属于数据驱动的非监督方法，虽然具有良好的泛化能力，但缺点是在专业应用领域的检测效果不如应用相关方法的准确性高。这主要是由于时间序列异常的定义与应用相关异常定义存在根本的不同。为了进一步改善异常检测方法在专业应用领域的检测效果，拟开展交互式的时间序列异常检测方法，即允许专家通过专业领域知识改善时间序列异常检测方法的检测效果。具体研究内容包括：1）研究建立和组织专家知识库的方法；2）利用库中的专家知识

为时间序列中的异常增加权重，弥补检测方法在应用相关专业知识方面的不足；3）选择具有代表性的应用，建立数据集，验证交互式检测方法的有效性。

3. 时间序列异常检测的理论与实际意义

时间序列异常检测的概念尚未推广到实际应用领域，主要原因是缺乏理论证明以及缺乏真实世界数据集的验证。为了将时间序列异常检测方法推广到更多应用领域，拟开展时间序列异常检测理论意义与实际用途的研究，具体研究内容包括：1）研究时间序列异常检测在概率统计学领域的意义；2）收集与建立更多时间序列相关数据集，为异常事件制作标记，运用这些数据集研究与验证时间序列异常检测的实际意义。

8.3 本章小节

本章对本文完成的工作进行了系统总结，并针对现在工作的局限性，提出了可能的未来工作计划。

参 考 文 献

[1] Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2015). Introduction to time series analysis and forecasting. John Wiley & Sons.

[2] Sakurai, Y., Faloutsos, C., & Yamamuro, M. (2007, April). Stream monitoring under the time warping distance. In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on (pp. 1046-1055). IEEE.

[3] Huijse, P., Estévez, P., Zegers, P., Príncipe, J. C., & Protopapas, P. (2011). Period estimation in astronomical time series using slotted correntropy. Signal Processing Letters, IEEE, 18(6), 371-374. IEEE.

[4] Jia, M., Liu, D., Song, K., Wang, Z., Jiang, G., Du, J., Zeng, L. (2010) Classification and Verification of Land Use/Cover in Australia Using MODIS Time-series Data. Remote Sensing Technology and Application, 25(3): 379~386. ACM & SIAM.

[5] Zhu, Y., & Shasha, D. (2003, June). Warping indexes with envelope transforms for query by humming. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (pp. 181-192). ACM.

[6] 吴邵春, 吴耿锋, 王炜等. (2006) 寻找地震相关地区的时间序列相似性匹配算法. 软件学报, .27(3): 185~192. Academic Journals Inc.

[7] Lu, C. J., Lee, T. S., & Chiu, C. C. (2009). Financial time series forecasting using independent component analysis and support vector regression. Decision Support Systems, 47(2), 115-125. Elsevier.

[8] Keogh, E., & Kasetty, S. (2003). On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Mining and knowledge discovery, 7(4), 349-371. Springer.

[9] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM

Computing Surveys (CSUR), 41(3), 15. ACM.

[10] Yankov, D., Keogh, E., & Rebbapragada, U. (2008). Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowledge and Information Systems, 17(2), 241-262. Springer

[11] Fox, A. J. (1972). Outliers in time series. Journal of the Royal Statistical Society. Series B (Methodological), 350-363. JSTOR

[12] Ma, J., & Perkins, S. (2003, August). Online novelty detection on temporal sequences. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 613-618). ACM.

[13] Wu, Q., & Shao, Z. (2005, October). Network anomaly detection using time series analysis. In Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on (pp. 42-42). IEEE.

[14] Keogh, E., Lin, J., Lee, S. H., & Van Herle, H. (2007). Finding the most unusual time series subsequence: algorithms and applications. Knowledge and Information Systems, 11(1), 1-27. Springer

[15] Keogh, E., Lin, J., & Fu, A. (2005, November). Hot sax: Efficiently finding the most unusual time series subsequence. In Data mining, fifth IEEE international conference on (pp. 8-pp). IEEE.

[16] NASA DashLink. (2007), Discovery in Aeronautics Systems Health. In https://dashlink.arc.nasa.gov. NASA.

[17] Jolliffe, I. (2002). Principal component analysis. John Wiley & Sons, Ltd. Chicago

[18] Lin, J., Keogh, E., Fu, A., & Van Herle, H. (2005, June). Approximations to magic: Finding unusual medical time series. In Computer-Based Medical Systems, 2005. Proceedings. 18th IEEE Symposium on (pp. 329-334). IEEE.

[19] Buu, H. T. Q., & Anh, D. T. (2011, October). Time series discord discovery based on iSAX symbolic representation. In Knowledge and Systems Engineering (KSE), 2011 Third

International Conference on (pp. 11-18). IEEE.

[20] Li, G., Bräysy, O., Jiang, L., Wu, Z., & Wang, Y. (2013). Finding time series discord based on bit representation clustering. Knowledge-Based Systems, 54, 243-254. Elsevier.

[21] Khanh, N. D. K., & Anh, D. T. (2012, August). Time series discord discovery using WAT algorithm and iSAX representation. In Proceedings of the Third Symposium on Information and Communication Technology (pp. 207-213). ACM.

[22] Luo, W., Gallagher, M., & Wiles, J. (2013). Parameter-free search of time-series discord. Journal of computer science and technology, 28(2), 300-310. Springer.

[23] Coomans, D., & Massart, D. L. (1982). Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-Nearest neighbour classification by using alternative voting rules. Analytica Chimica Acta, 136, 15-27. Elsevier.

[24] Bentley, J. L., & Sedgewick, R. (1997, January). Fast algorithms for sorting and searching strings. In SODA (Vol. 97, pp. 360-369). ACM.

[25] Chiu, B., Keogh, E. & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. In the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp 493-498. ACM.

[26] Kitaguchi, S. (2004). Extracting Feature based on Motif from a Chronic Hepatitis Dataset. In proceedings of the 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI).

[27] Tanaka, Y. & Uehara, K. (2004). Motif Discovery Algorithm from Motion Data. In proceedings of the 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI).

[28] Rombo, S. & Terracina, G. (2004). Discovering Representative Models in Large Time Series Databases. In proceedings of the 6th International Conference On Flexible Query Answering Systems. pp 84-97. Springer.

[29] Ruzzo, W.L, & Tompa, M. (1999). A linear time algorithm for finding all maximal scoring

subsequences. In Proc Int Conf Intell Syst Mol Biol.; pp 234-41. AAAI Press.

[30] Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases (pp. 69-84). Springer Berlin Heidelberg. Springer.

[31] Chan, K. P., & Fu, A. W. C. (1999, March). Efficient time series matching by wavelets. In Data Engineering, 1999. Proceedings., 15th International Conference on (pp. 126-133). IEEE.

[32] Korn, F., Jagadish, H. V., & Faloutsos, C. (1997). Efficiently supporting ad hoc queries in large datasets of time sequences. ACM SIGMOD Record, 26(2), 289-300. ACM.

[33] Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Dimensionality reduction for fast similarity search in large time series databases. Knowledge and information Systems, 3(3), 263-286. Springer

[34] Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. ACM SIGMOD Record, 30(2), 151-162. ACM.

[35] Keogh, E., Chu, S., Hart, D., & Pazzani, M. (2001). An online algorithm for segmenting time series. In Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on (pp. 289-296). IEEE.

[36] Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003, June). 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 (pp. 2-11). ACM.

[37] Keogh, E., Lonardi, S., & Chiu, B. Y. C. (2002, July). Finding surprising patterns in a time series database in linear time and space. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 550-556). ACM.

[38] Wei, L., Kumar, N., Lolla, V. N., Keogh, E. J., Lonardi, S., & Chotirat (Ann) Ratanamahatana. (2005, June). Assumption-Free Anomaly Detection in Time Series. In

SSDBM (Vol. 5, pp. 237-242). IEEE.

[39] Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching (Vol. 14, No. 2, pp. 47-57). ACM.

[40] Assent, I., Krieger, R., Afschari, F., & Seidl, T. (2008, March). The TS-tree: efficient time series search and retrieval. In Proceedings of the 11th international conference on extending database technology: Advances in database technology (pp. 252-263). ACM.

[41] Wang, Y., Wang, P., Pei, J., Wang, W., & Huang, S. (2013). A data-adaptive and dynamic segmentation index for whole matching on time series. Proceedings of the VLDB Endowment, 6(10), 793-804. VLDB Endowment. Chicago.

[42] Shieh, J., & Keogh, E. (2009). iSAX: disk-aware mining and indexing of massive time series datasets. Data Mining and Knowledge Discovery, 19(1), 24-57. Spinger

[43] Camerra, A., Palpanas, T., Shieh, J., & Keogh, E. (2010, December). iSAX 2.0: Indexing and mining one billion time series. In Data Mining (ICDM), 2010 IEEE 10th International Conference on (pp. 58-67). IEEE. Chicago

[44] Bu, Y., Leung, O. T. W., Fu, A. W. C., Keogh, E. J., Pei, J., & Meshkin, S. (2007, April). WAT: Finding Top-K Discords in Time Series Database. In SDM (pp. 449-454). SIAM.

[45] Fu, A. W. C., Leung, O. T. W., Keogh, E., & Lin, J. (2006). Finding time series discords based on haar transform. In Advanced Data Mining and Applications (pp. 31-41). Springer Berlin Heidelberg. Springer.

[46] Ameen, J., & Basha, R. (2007, September). Higherrarchical data mining for unusual sub-sequence identifications in time series processes. In Innovative Computing, Information and Control, 2007. ICICIC'07. Second International Conference on (pp. 177-177). IEEE.

[47] Basha, R., & Ameen, J. (2007). Unusual sub-sequence identifications in time series with periodicity. International Journal of Innovative Computing, Information and Control, 3(2), 471-480. Citeseer.

[48] Luo, W., & Gallagher, M. (2011). Faster and parameter-free discord search in quasi-

periodic time series. In Advances in Knowledge Discovery and Data Mining (pp. 135-148). Springer Berlin Heidelberg. Springer.

[49] Jones, M., Nikovski, D., Imamura, M., & Hirata, T. (2014). Anomaly Detection in Real-Valued Multidimensional Time Series. Academy of Science and Engineering (ASE), U.S.A, ASE.

[50] Jin, X., & Han, J. (2010). K-Medoids Clustering. In Encyclopedia of Machine Learning (pp. 564-565). Springer US.

[51] Eckmann, J. P., Kamphorst, S. O., & Ruelle, D. (1995). Recurrence plots of dynamical systems. World Scientific Series on Nonlinear Science Series A, 16, 441-446. World Scientific Publishing.

[52] Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000, May). LOF: identifying density-based local outliers. In ACM sigmod record (Vol. 29, No. 2, pp. 93-104). ACM.

[53] Michie, D. (1968). Memo functions and machine learning. Nature, 218(5136), 19-22.

[54] Teng, M. (2010, December). Anomaly detection on time series. In Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on (Vol. 1, pp. 603-608). IEEE. Chicago.

[55] Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., ... & Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. Circulation, 101(23), e215-e220. Am Heart Assoc.

[56] Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., & Paxson, V. (2014, November). The matter of Heartbleed. In Proceedings of the 2014 Conference on Internet Measurement Conference (pp. 475-488). ACM.

[57] Ruan, X. (2014). Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine. Apress.

[58] McCanne, S., & Jacobson, V. (1993, January). The BSD packet filter: A new architecture

for user-level packet capture. In Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (pp. 2-2). USENIX Association.

[59] Tak, Y. S., Kim, J., & Hwang, E. (2012). Hierarchical querying scheme of human motions for smart home environment. Engineering Applications of Artificial Intelligence, 25(7), 1301-1312. Elsevier

[60] Casale, P., Pujol, O., & Radeva, P. (2012). Personalization and user verification in wearable systems using biometric walking patterns. Personal and Ubiquitous Computing, 16(5), 563-580. Springer

[61] Keogh, E. J., & Pazzani, M. J. (2000). A simple dimensionality reduction technique for fast similarity search in large time series databases. In Knowledge Discovery and Data Mining. Current Issues and New Applications (pp. 122-133). Springer Berlin Heidelberg.

[62] Perng, C. S., Wang, H., Zhang, S. R., & Parker, D. S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. In Data Engineering, 2000. Proceedings. 16th International Conference on (pp. 33-42). IEEE.

[63] Keogh, E., Palpanas, T., Zordan, V. B., Gunopulos, D., & Cardle, M. (2004, August). Indexing large human-motion databases. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30 (pp. 780-791). VLDB Endowment.

[64] Chudova, D., Gaffney, S., Mjolsness, E., & Smyth, P. (2003, August). Translation-invariant mixture models for curve clustering. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 79-88). ACM.

[65] Johnson, M., McCraw, H., Moore, S., Mucci, P., Nelson, J., Terpstra, D., & Mohan, T. (2012, September). PAPI-V: performance monitoring for virtual machines. In 2012 41st International Conference on Parallel Processing Workshops (pp. 194-199). IEEE.

[66] Fodor, I. K. (2002). A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory

[67] Huang, T., Zhu, Y., Wu, Y., Bressan, S., & Dobbie, G. (2015). Anomaly detection and identification scheme for VM live migration in cloud infrastructure. Future Generation Computer Systems. Elsevier.

[68] Lazarevic, A., Ertöz, L., Kumar, V., Ozgur, A., & Srivastava, J. (2003, May). A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In SDM (pp. 25-36). SIAM.

[69] Kyle, B., (2011, July). Case Study: Web Logs and Perfmon Data. http://blog.serverfault.com/2011/07/14/the-three-monitoring-perspectives-part-1-ground-level/

[70] Ferrell, B & Santuro, S. (2005). NASA Shuttle Valve Data. http://www.cs.fit.edu/~pkc/nasa/data/. NASA.

[71] UCI repository of machine learning database [Online]. (2014) University of California, Irvine, Available: http:// archive.ics.uci.edu/ ml/ datasets/ Individual +household +electric +power +consumption

[72] Kawala, F., Douzal-Chouakria, A., Gaussier, E., & Diemert, E. (2014) Apprentissage d’ordonnancement pour la prédiction d’activité sur les réseaux sociaux. 1-15. Association francophone de recherche d'information et applications.

[73] Huang, T., Zhu, Y., Qiu, M., Yin, X., & Wang, X. (2013). Extending Amdahl’s law and Gustafson’s law by evaluating interconnections on multi-core processors. The Journal of Supercomputing, 66(1), 305-319. Springer.

[74] Intel 64 and IA-32 Architectures Optimization Reference Manual. (2012 April), Intel, http://www.intel.com/ content/dam/doc/manual/ 64-ia-32-architectures-optimization-manual.pdf

[75] Understanding Switch Latency. (2012 June) Cisco, http://www.cisco.com/c/en/us/products/collateral/ switches/nexus-3000-series-switches/white_paper_c11-661939.html

[76] Shieh, J., & Keogh, E. (2008, August). iSAX: indexing and mining terabyte sized time series. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 623-631). ACM.

[77] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113. ACM.

[78] Spark, A. (2014). Apache spark–lightning-fast cluster computing.

[79] Haykin, S., & Network, N. (2004). A comprehensive foundation. Neural Networks, 2(2004). Prentice Hall PTR.

[80] Ryan, J., Lin, M. J., & Miikkulainen, R. (1998). Intrusion detection with neural networks. Advances in neural information processing systems, 943-949. Morgan Kaufmann Publishers.

[81] Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on (Vol. 2, pp. 1702-1707). IEEE.

[82] Shah, B., & Trivedi, B. H. (2012). Artificial neural network based intrusion detection system: A survey. International Journal of Computer Applications, 39(6). Foundation of Computer Science (FCS)

[83] Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. Machine learning, 29(2-3), 131-163. Springer.

[84] Sebyala, A. A., Olukemi, T., Sacks, L., & Sacks, D. L. (2002, May). Active platform security through intrusion detection using naive bayesian network for anomaly detection. In London Communications Symposium. Citeseer.

[85] Wong, W. K., Moore, A., Cooper, G., & Wagner, M. (2003, August). Bayesian network anomaly pattern detection for disease outbreaks. In ICML (pp. 808-815). Morgan Kaufmann Publishers Inc.

[86] Mascaro, S., Nicholso, A. E., & Korb, K. B. (2014). Anomaly detection in vessel tracks

using Bayesian networks. International Journal of Approximate Reasoning, 55(1), 84-98. Elsevier.

[87] Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers. Neural processing letters, 9(3), 293-300. Springer.

[88] Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., & Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. Bioinformatics, 16(10), 906-914. Oxford Univ Press

[89] Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on (Vol. 2, pp. 1702-1707). IEEE.

[90] Martins, H., Palma, L., Cardoso, A., & Gil, P. (2015, May). A support vector machine based technique for online detection of outliers in transient time series. In Control Conference (ASCC), 2015 10th Asian (pp. 1-6). IEEE.

[91] Cohen, W. W. (1995, July). Fast effective rule induction. In Proceedings of the twelfth international conference on machine learning (pp. 115-123). Morgan Kaufmann Publishers Inc.

[92] Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. IEEE transactions on systems, man, and cybernetics, 21(3), 660-674. IEEE.

[93] Lee, W., Stolfo, S. J., & Mok, K. W. (1999). A data mining framework for building intrusion detection models. In Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on (pp. 120-132). IEEE.

[94] Lee, J. H., Lee, J. H., Sohn, S. G., Ryu, J. H., & Chung, T. M. (2008, February). Effective value of decision tree with KDD 99 intrusion detection datasets for intrusion detection system. In Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on (Vol. 2, pp. 1170-1175). IEEE.

[95] Brauckhoff, D., Dimitropoulos, X., Wagner, A., & Salamatian, K. (2012). Anomaly

extraction in backbone networks using association rules. IEEE/ACM Transactions on Networking (TON), 20(6), 1788-1799. IEEE.

[96] Anscombe, F. J. (1960). Rejection of outliers. Technometrics, 2(2), 123-146. Taylor & Francis Group

[97] Hazel, G. G. (2000). Multivariate Gaussian MRF for multispectral scene segmentation and anomaly detection. Geoscience and Remote Sensing, IEEE Transactions on, 38(3), 1199-1211. IEEE.

[98] Pena, E. H., Barbon, S., Rodrigues, J. J., & Lemes Proenca Junior, M. (2014, June). Anomaly detection using digital signature of network segment with adaptive ARIMA model and Paraconsistent Logic. In Computers and Communication (ISCC), 2014 IEEE Symposium on (pp. 1-6). IEEE.

[99] Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions.

[100] Xie, M., Hu, J., & Tian, B. (2012, June). Histogram-based online anomaly detection in hierarchical wireless sensor networks. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on (pp. 751-759). IEEE.

[101] Tan, P. N., Steinbach, M., & Kumar, V. (2005). Introduction to data mining. Addison-Wesley.

[102] Amer, M., & Goldstein, M. (2012). Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer. In Proc. of the 3rd RapidMiner Community Meeting and Conference (RCOMM 2012) (pp. 1-12). Shaker Verlag GmbH.

[103] Branch, J. W., Giannella, C., Szymanski, B., Wolff, R., & Kargupta, H. (2013). In-network outlier detection in wireless sensor networks. Knowledge and information systems, 34(1), 23-54. Springer

[104] Wu, M., & Jermaine, C. (2006, August). Outlier detection by sampling with accuracy guarantees. In Proceedings of the 12th ACM SIGKDD international conference on

Knowledge discovery and data mining (pp. 767-772). ACM.

[105] Chawla, S., & Sun, P. (2006). SLOM: a new measure for local spatial outliers. Knowledge and Information Systems, 9(4), 412-429. Springer.

[106] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In Kdd (Vol. 96, No. 34, pp. 226-231). AAAI.

[107] Ertöz, L., Steinbach, M., & Kumar, V. (2004). Finding topics in collections of documents: A shared nearest neighbor approach (pp. 83-103). Springer.

[108] Smith, R., Bivens, A., Embrechts, M., Palagiri, C., & Szymanski, B. (2002). Clustering approaches for anomaly based intrusion detection. Proceedings of intelligent engineering systems through artificial neural networks, 579-584. ASME.

[109] Budalakoti, S., Srivastava, A., Akella, R., & Turkov, E. (2006). Anomaly detection in large sets of high-dimensional symbol sequences. NASA Ames Research Center, Tech. Rep. NASA TM-2006-214553. NASA.

[110] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In Applications of data mining in computer security (pp. 77-101). Springer.

[111] Pires, A., & Santos-Pereira, C. (2005). Using clustering and robust estimators to detect outliers in multivariate data. In Proceedings of the International Conference on Robust Statistics. Citeseer.

[112] Li, M., & Vitányi, P. (2013). An introduction to Kolmogorov complexity and its applications. Springer Science & Business Media.

[113] Rrnyi, A. L. F. R. P. E. D. (1961). On measures of entropy and information. In Fourth Berkeley symposium on mathematical statistics and probability (Vol. 1, pp. 547-561).

[114] Vedral, V. (2002). The role of relative entropy in quantum information theory. Reviews of Modern Physics, 74(1), 197. APS.

[115] Nychis, G., Sekar, V., Andersen, D. G., Kim, H., & Zhang, H. (2008, October). An empirical evaluation of entropy-based traffic anomaly detection. In Proceedings of the 8th ACM SIGCOMM conference on Internet measurement (pp. 151-156). ACM.

[116] Ding, Q., & Kolaczyk, E. D. (2013). A compressed PCA subspace method for anomaly detection in high-dimensional data. Information Theory, IEEE Transactions on, 59(11), 7419-7433. IEEE.

[117] Xiang, Y., Li, K., & Zhou, W. (2011). Low-rate DDoS attacks detection and traceback by using new information metrics. Information Forensics and Security, IEEE Transactions on, 6(2), 426-437. IEEE.

[118] Sun, J., Xie, Y., Zhang, H., & Faloutsos, C. (2007). Less is more: Compact matrix representation of large sparse graphs. In Proceedings of 7th SIAM International Conference on Data Mining. SIAM.

[119] Dutta, H., Giannella, C., Borne, K. D., & Kargupta, H. (2007, April). Distributed Top-K Outlier Detection from Astronomy Catalogs using the DEMAC System. In SDM (pp. 473-478). SIAM.

[120] Agovic, A., Banerjee, A., Ganguly, A. R., & Protopopescu, V. (2008). 6 Anomaly Detection in Transportation Corridors Using Manifold Embedding. Knowledge Discovery from Sensor Data, 81-105. CRC Press

[121] Huang, T., Zhu, Y., Wu, Y., Shi, W., (2015) J-distance Discord: An Improved Time Series Discord Definition and Discovery Method. 2015 IEEE 15th International Conference on Data Mining Workshops. IEEE.

[122] Huang, T., Zhu, Y., Zhang, Q., Zhu, Y., Wang, D., Qiu, M., & Liu, L. (2013, July). An LOF-based Adaptive Anomaly Detection Scheme for Cloud Computing. In Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual (pp. 206-211). IEEE.

[123] Wu, Y., Zhu, Y., & Huang T., (2015, August). Distributed Discord Discovery: Spark

Based Anomaly Detection in Time Series, IEEE 17th International Conference on High Performance Computing and Communications (HPCC 2015), 155-159, New York, U.S.A. IEEE.

附录一 符号与标记

APDD	Approximated Parallel Discord Discovery	近似的并行化时间序列异常检测方法
ASC	Adaptive Stop Criteria	自适应停止条件
SAX	Symbolic Aggregate approXimation	符号化方法
DDE	Distributed Discord Estimation	分布式异常估算方法
DDS	Direct Discord Search	直接异常搜索
DR-LOF	Dimension Reasoning LOF	溯源的异常检测方法
GDS	General Direct Search	通用的异常直接搜索方法
JDD	J-distance Discord	J 距离异常
LOF	Local Outlier Factor	局部离群系数
MDD	Multi-dimensional Discord Discovery	多维时间序列异常检测方案
PAA	Piecewise Aggregate Approximation	逐段聚集平均
PDD	Parallel Discord Discovery	并行化的时间序列异常检测方法
WAT	Wavelet Augmented Trie	小波增强前缀树

附录二 推导与证明

1. 子序列的 J 距离与邻居 J 距离之间的大小关系证明：

给定一个子序列C_p和它的不重叠的第 J 个最近邻C_q，在使用对称距离的情况下，对于任何正整数 J，有$Jdist(C_q) \leq 2 \times Jdist(C_p)$，具体证明如下：

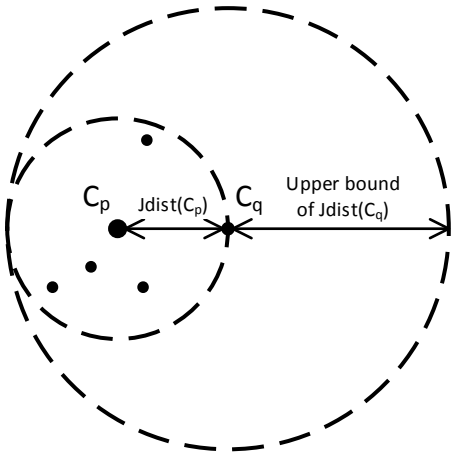

图 1 子序列的 J 距离与邻居 J 距离之间的大小关系

Fig. 1 The J distance of a subsequence C_p and the J distance of it J^{th} nearest neighbor C_q

证明：将子序列看作 n 维空间的点，n 是子序列的长度，如图 1 所示。考虑一个以C_p为球心，$dist(C_p, C_q)$为半径的球面，C_q是C_p的第 J 个不重叠最近邻，即$dist(C_p, C_q) = Jdist(C_p)$。所以$C_p$的前J个不重叠最近邻都分布在球面的表面或内部。显然这些最近邻之间的距离小于或等于$2 \times Jdist(C_p)$，以C_q为球心，以$2 \times Jdist(C_p)$为半径的球体至少包

含C_p及其 J-1 个最近邻在内的 J 个邻居，C_q的J距离$Jdist(C_q)$一定小于等于该球体的半径，所以我们可以断定C_p的第 J 个最近邻的 J 距离的上界为$2 \times dist(C_p, C_q)$。□

2. DR-LOF 性质的证明

性质 1. 如果数据点 A 是正常的，那么移除数据点 A 的任何一个维度都将得到与原 LOF 值相近的新 LOF 值。

性质 1 可以通过如下第一个假设进行解释：A 的 k 个最近邻都处在以 A 为圆心，kdist(A)为半径的超球体的表面，那么 A 与其任一前 k 个最近邻 B 的可达距离可以表示为：

$$rd_k(A,B) = \sqrt{\big(A(1) - B(1)\big)^2 + \cdots + (A_d - B_d)^2} \tag{1}$$

除去第 i 维情况下的可达距离可以表示为：

$$\Big(rd_k^i(A,B)\Big)^2 = \big(rd_k(A,B)\big)^2 - \big(A(i) - B(i)\big)^2 \tag{2}$$

因此$rd_k^i(A,B)$与$rd_k(A,B)$的比值可以表示为：

$$\frac{rd_k^i(A,B)}{rd_k(A,B)} = \sqrt{1 - \left(\frac{A(i) - B(i)}{rd_k(A,B)}\right)^2} \tag{3}$$

由于每个维度对可达距离的贡献相近，我们做出如下第二个假设：第 i 个维度对可达距离的贡献度为$\frac{\mathrm{rd_k}(A,B)}{d}$，根据该假设，我们简化(3)可得：

$$\frac{rd_k^i(A,B)}{rd_k(A,B)} = \sqrt{1 - \frac{1}{\mathrm{d}}} \tag{4}$$

我们将(5-6)展开为与$rd_k^i(A,B)$和$rd_k(A,B)$相关的表达式，可得：

$$\frac{LOF_k^i(A)}{LOF_k(A)} = \frac{\sum_{B \in N_k(A)} rd_k^i(A,B)}{\sum_{B \in N_k(A)} rd_k(A,B)} \times \frac{\sum_{B \in N_k(A)} \frac{1}{\sum_{C \in N_k(B)} rd_k^i(B,C)}}{\sum_{B \in N_k(A)} \frac{1}{\sum_{C \in N_k(B)} rd_k(B,C)}} \tag{5}$$

然后，根据先前的假设以及将(4)代入(5)可得：

$$DR\text{-}LOF_k^n(A) = \sqrt{1-\frac{1}{d}} \times \frac{1}{\sqrt{1-\frac{1}{d}}} = 1 \tag{6}$$

除此之外我们还可以得到如下性质：

性质 2. 如果数据点 A 是异常的，那么 DR-LOF 值按照如下规律变化：
当 A 的第 i 维是异常的主要原因，那么除去第 i 维的新 LOF 值将变小
当 A 的第 i 维不是异常的主要原因，那么除去第 i 维的新 LOF 值将变大

性质 2 可以通过如下推算进行解释：假设$\mathbb{C}$是一个 d 维空间中由正常数据点组成的集群，A 是一个异常点，所以 A 是远离集群 C 的。假设 A 与$\mathbb{C}$在第 i 维上的映射相距较远，而 A 在其他维度的映射落入集群$\mathbb{C}$中。为了更准确的表达以上假设，我们给出如下定义:

1. $a = Avg_{j\in U}\{|A(j) - B(j)|\}$，其中 U 是 A 的正常维度，B 是$\mathbb{C}$中的点；
2. $b = Avg_{i\in V}\{|A(i) - B(i)|\}$，其中 V 是 A 的异常维度，B 是$\mathbb{C}$中的点。

B 代表$\mathbb{C}$中的正常数据点。由于 V 是 A 中的异常维度，那么我们可以假设$a < b$。如果在计算 DR-LOF 值时被移去的第 i 个维度度是异常维度，即i ∈ V，那么(3)可以被继续简化为

$$\sqrt{1-\left(\frac{A(i)-B(i)}{rd_k(A,B)}\right)^2} = \sqrt{1-\frac{b^2}{|U|a^2+|V|b^2}} < \sqrt{1-\frac{1}{d}} \tag{7}$$

其中$|U|$和$|V|$分别表示 U 和 V 中维度的数量。根据(7)，可得

$$\begin{aligned} DR\text{-}LOF_k^i(A) &= \sqrt{1-\left(\frac{A(i)-B(i)}{rd_k(A,B)}\right)^2} \times \sqrt{\frac{d}{d-1}} \\ &< \sqrt{1-\frac{1}{d}} \times \sqrt{\frac{d}{d-1}} = 1 \end{aligned} \tag{8}$$

因此性质 2 的第一部分成立。如果被移除的第 i 个维度是正常的维度，那么(3)可以被简化成：

$$\sqrt{1-\left(\frac{A(i)-B(i)}{rd_k(A,B)}\right)^2} = \sqrt{1-\frac{a^2}{|U|a^2+|V|b^2}} > \sqrt{1-\frac{1}{d}} \tag{9}$$

并且有：

$$DR\text{-}LOF_k^i(A) = \sqrt{1-\left(\frac{A(i)-B(i)}{rd_k(A,B)}\right)^2} \times \sqrt{\frac{d}{d-1}} > \sqrt{1-\frac{1}{d}} \times \sqrt{\frac{d}{d-1}} = 1 \tag{10}$$

因此性质 2 的第二部分成立。

攻读博士学位期间已发表或录用的论文

已发表或录用的论文：

[1] Huang, T., Zhu, Y., Wu, Y., Bressan, S., & Dobbie, G. (2015). Anomaly detection and identification scheme for VM live migration in cloud infrastructure. Future Generation Computer Systems. **Impact Factor: 2.786**

[2] Huang, T., Zhu, Y., Qiu, M., Yin, X., & Wang, X. (2013). Extending Amdahl's law and Gustafson's law by evaluating interconnections on multi-core processors. The Journal of Supercomputing, 66(1), 305-319. **Impact Factor: 0.858**

[3] Zhang, Q., Huang, T., Zhu, Y., & Qiu, M. (2013). A case study of sensor data collection and analysis in smart city: provenance in smart food supply chain. International Journal of Distributed Sensor Networks, 2013 **Impact Factor: 0.665**

[4] Huang, T., Zhu, Y., Mao Y., Li X., Liu M., Ha, Y., Dobbie, G., Parallel Discord Discovery. The 20th Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD) 2016 (Accepted)

[5] Huang, T., Zhu, Y., Wu, Y., Shi, W., (2015) J-distance Discord: An Improved Time Series Discord Definition and Discovery Method. 2015 IEEE 15th International Conference on Data Mining Workshops

[6] Huang, T., Zhu, Y., Zhang, Q., Zhu, Y., Wang, D., Qiu, M., & Liu, L. (2013, July). An LOF-based Adaptive Anomaly Detection Scheme for Cloud Computing. In Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual (pp. 206-211). IEEE.

[7] Zu, Y., Huang, T., & Zhu, Y. (2013, December). An efficient power-aware resource

scheduling strategy in virtualized datacenters. In Parallel and Distributed Systems (ICPADS), 2013 International Conference on (pp. 110-117). IEEE.

[8] Xie, Q., Huang, T., Zou, Z., Xia, L., Zhu, Y., & Jiang, J. (2012, December). An accurate power model for GPU processors. In Computing and Convergence Technology (ICCCT), 2012 7th International Conference on (pp. 1141-1146). IEEE.

[9] Wu, Y., Zhu, Y., & Huang, T., (2015, August). Distributed Discord Discovery: Spark Based Anomaly Detection in Time Series, IEEE 17th International Conference on High Performance Computing and Communications (HPCC 2015), 155-159, New York, U.S.A. IEEE.

审稿中的论文：

[1] Huang T., Zhu Y., Ha Y., Wang X., Qiu M. A Hardware Pipeline with High Energy and Resource Efficiency for FMM Acceleration. 2015. Transactions on Embedded Computing Systems. ACM.

攻读博士学位期间参与的科研项目

[1] 863 计划重点项目，新概念高效能计算机体系结构及系统研究开发（2009AA012201)一级子课题、2011-2012、大文件服务应用设备试制（CFA2011SHJD01），研究分布式数据存储与访问的效率问题

[2] 863 计划重点项目，新概念高效能计算机体系结构及系统研究开发（2009AA012201)一级子课题、2009-2011、上海交通大学 CFA 项目系统概要设计（CFA2009SHJD01），研究时间序列异常检测的并行化问题

[3] 上海交大与新加坡国立大学国际合作“卓越研究与技术企业学园（简称 CREATE 计划）”项目合作协议，“超大城市的能源环境可持续发展方案（简称 E2S2）”，2013-2017，研究时间序列异常检测在环境保护中的应用

[4] 国家自然科学基金（NSFC）面上项目：“支持动态可扩展 Cache 一致性分区的众核处理器关键技术研究”，项目批准号：61373032，研究时间序列检测方法的并行化问题

[5] 863 计划主题项目：“大数据分析技术在输变电设备状态评估中的研究及应用”，项目编号：2015AA050204，2015 年至 2018 年，研究超大规模时间序列异常检测方法在变电设备异常检测领域的应用

致 谢

非常感谢我的指导老师祝永新副教授。祝老师充分给予我课题选择与研究方面的自由；不遗余力地为我创造宽松的环境和良好的氛围，从学术、生活等方面支持着我的课题研究；在我犯错误时给予宽容与体谅；在我遇到失败与挫折时给予关怀与勉励。感谢祝老师给予我出席国际会议和赴新加坡学习交流的机会，帮助我开拓了视野。

感谢新加坡 A*STAR 研究院的哈亚军教授以及 PACE 大学的邱美康教授给予我学术研究方面的指导。感谢新加坡国立大学计算机学院 Stephane Bressan 教授为我开展课题研究提供的帮助。感谢上海交通大学机械与动力学院的刘晓教授在新加坡以及我回国后都给予我的生活与学习上的关怀。感谢开题时提供建设性建议的施国勇，毛志刚，蒋江，连勇等老师，他们的指导加快了课题开展进度，感谢高峰老师在日常教务给予我的各种支持。

我想感谢的人太多了，感谢陪我度过冗长博士生活的实验室学长学姐学弟学妹焦佳佳，阙志强，王绪，王冬阳，张倩楠，辜晓琪，石巍巍，王畅，吴亚飞，陶翔，毛亦姝，刘梦云，李欣阳。感谢邓瑶瑶同学在我博士生活最忙碌心烦的时侯及时给我“添乱”。感谢你们为我的博士生活变得绚丽多彩。

感谢爸爸妈妈给予我生命，在任何时间无条件接纳我的任性，在我生病时给予我无微不至的关怀，在我面临重大选择时无论我作了怎样的决定都会给予我鼓励与支持，没有你们就没有我今天的成就。

我想对所有关怀过我帮助过我的人说声：谢谢！

黄田

2015 年 9 月于上海交通大学

上海交通大学博士学位论文答辩决议书

0102109005

姓　名	黄田	学号	0102109005	所在学科	电子科学与技术(芯片设计与系统)
指导教师	祝永新	答辩日期	2016-01-13	答辩地点	微电子学院401室
论文题目	面向超大规模时间序列的异常检测				

投票表决结果：7/7/7（同意票数/实到委员数/应到委员数）答辩结论：☑通过　□未通过

评语和决议：

超大规模时间序列分析是数据挖掘研究领域的热点。论文研究了超大规模时间序列异常检测问题，提出了四种时间序列异常检测方法。其主要创新点如下：

1. 针对现有时间序列异常检测方法的“双胞胎怪胎”问题(Twin Freak Problem)，改进了时间序列异常的定义，并且提出了基于该新定义的异常检测方法；
2. 提出了多维时间序列异常检测方法，通过降维降低了异常检测的时间复杂度，并通过异常溯源方法给出了潜在的引起异常的因素；
3. 通过近似的划分方法将时间序列异常检测问题分解为不相关的子问题，降低了检测算法的计算复杂度，并给出了一种并行化的实现方法；
4. 基于Spark并行处理平台，提出了一种并行化的异常检测方法，提高异常检测的效率。

该论文的选题具有较高的理论价值和应用前景，内容翔实丰富，数据可信，条理清楚，表明该作者具有扎实的理论基础和宽广的专业知识，具备了独立从事科研的能力。答辩过程中表述清楚，回答问题正确。

经答辩委员会无记名投票表决，一致通过黄田博士学位论文答辩，建议授予其工学博士学位。

2016 年 1 月 13 日

	职务	姓名	职称	单位	签名
答辩委员会成员签名	主席	王伶俐	教授	复旦大学	
	委员	祝永新	副教授	上海交通大学	
	委员	王国兴	副教授	上海交通大学	
	委员	汪辉	研究员	中国科学院上海高等研究院	
	委员	毛志刚	教授	上海交通大学	
	委员	朱燕民	教授	上海交通大学	
	委员	戚正伟	教授	上海交通大学	
	秘书	刘婷	讲师	上海交通大学	