arXiv:2512.14723v1 [cs.DB] 9 Dec 2025

MS-Index：基于欧几里得距离的多变量时间序列快速前 k 名子序列搜索

Jens d'Hondt
j.e.d.hondt@tue.nl 埃因霍温理工大学 埃因霍温，荷兰

Teun Kortekaas
teun@jelter.net 埃因霍温理工大学 埃因霍温，荷兰

Odysseas Papapetrou
o.papapetrou@tue.nl 埃因霍温理工大学 埃因霍温，荷兰

Themis Palpanas
themis@mi.parisdescartes.fr 巴黎西岱大学 & 法国大学研究院 巴黎，法国

摘要

现代应用经常以多变量时间序列（MTS）的形式收集和分析时间数据，即包含多个通道的时间序列。在此类场景中，一项常见任务是子序列搜索，即找出所有包含与查询时间序列高度相似子序列的 MTS。在实际场景中，MTS 的所有通道并非都与每次查询相关。例如，飞机传感器可能会收集大量部件和子系统的数据，但其中只有少数与特定查询相关，例如确定起落架故障的原因，或识别某个特定的飞行动作。因此，相关的查询通道通常会在查询时指定。本文介绍了多变量子序列索引（MS-Index），这是一种新颖的算法，用于在欧几里得距离下进行多变量时间序列子序列最近邻搜索，并支持按需选择查询通道。该算法是精确的，其查询性能随查询通道数量的增长呈次线性扩展。我们通过对 34 个数据集进行全面的实验评估，考察了 MS-Index 的特性，并表明无论对于原始子序列还是归一化子序列，其性能都比最先进的方法高出一到两个数量级。

1 引言

时间序列广泛存在于天体物理学、地震学、气象学、医疗保健、金融、视频和音频录制等多个领域 [5, 36, 38, 41, 57]。随着传感器技术的发展，近年来采集到的时间序列数量大幅增加，尤其是多变量时间序列（MTS） [62]。每个 MTS 都由多个时间序列构成，这些序列通常来自不同的传感器，用于测量同一现象或对象的不同方面。这些不同的时间序列称为 MTS 的通道 。MTS 的示例包括医院患者的健康监测数据（例如心率、血压和体温）、气候传感器阵列（例如在某一地点测量温度、湿度和气压的阵列） [44]，以及动作捕捉数据（例如不同身体部位的位置和加速度）。

时间序列的一项关键操作是相似性搜索，即查找与查询时间序列最相似的时间序列（也称为其最近邻） [25, 26]。相似性搜索也可以作为独立任务执行 [16, 21, 28, 40, 42, 45, 59, 60, 66],

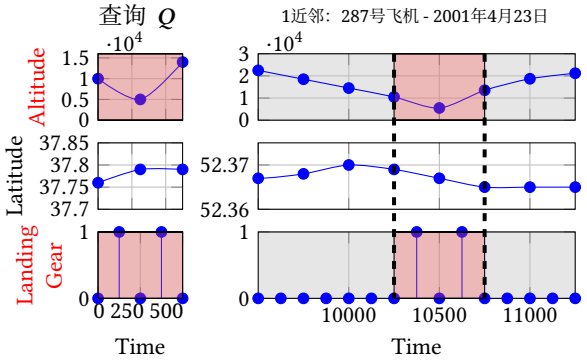

图1：合成飞机数据的多变量时间序列查询示例及其1近邻。高度和起落架是查询通道。高亮框（红色）表示所考虑的子序列。

但它也常作为子程序用于离群点检测 [14, 21, 70], 、分类 [7, 34, 63, 67, 68, 76], 和聚类 [9, 13, 23, 37, 64]等任务中。

为了充分利用不断增长的数据集，现代相似性搜索算法应支持灵活性很高的查询。例如，在分析患者数据时，医生可能需要在患者的生命体征中查找短期模式的相似出现，以了解其病情的根本原因。此类查询要求算法支持：(a) 多变量时间序列（多个传感器），(b) 比较子序列而非整个时间序列，以及 (c) 在查询时确定查询通道（仅选择与患者相关生命体征有关的传感器）。另一个需要此类查询的例子是分析飞机的传感器数据 [12],：用户正在分析因起落架故障而导致的飞机降落失败，并希望在历史数据中查找相似情况，以了解根本原因。在这种情况下，用户可以选择降落失败的时间段（即图1左侧的时间序列）以及相关通道（图中以红色高亮显示），以便在历史数据中查找相似情况。

尽管人们已针对大型单变量时间序列（UTS）数据集上的相似性搜索所面临的挑战开展了大量研究，但多变量时间序列（MTS）搜索方面的研究仍处于起步阶段。现有解决方案仅支持搜索完整时间序列（全序列匹配），而非子序列（子序列搜索），且只能在一组固定的预设通道上进行。在这项工作中，我们表明

将最先进的全序列匹配解决方案扩展到子序列搜索时，其性能会下降。进一步将这些解决方案扩展为支持多变量数据，只会让问题更加严重，因为通道数量也带来了额外挑战。总而言之，现有算法至少在以下某一方面存在不足：（a）它们原生仅支持单变量时间序列（UTS），而将其扩展到多变量时间序列（MTS）并非易事；（b）它们专注于全序列匹配，在加入子序列搜索后性能会变得无法接受；或者（c）它们依赖对用户查询的严格限制性假设，例如要求用户为每个通道指定阈值。

在这项工作中，我们提出了 *MS-Index*，这是一种新颖的算法，用于在多变量时间序列上基于欧几里得距离进行 k 近邻（*k*-NN）子序列搜索。MS-Index 支持在查询时选择查询通道，并适用于归一化和未归一化子序列。[1] MS-Index 支持固定长度查询，即索引构建时必须已知查询的长度。该算法是精确的：始终返回正确结果。

MS-Index 与现有方法有三点不同。首先，它专为多变量时间序列设计，利用了多通道带来的额外剪枝潜力。其次，它将索引驱动的搜索空间剪枝与通过卷积定理（MASS）[2, 50] 对剩余候选项进行的高效距离计算相结合。为此，我们使用一棵 R 树来索引所有通道上经离散傅里叶变换（DFT）近似的子序列，并采用两遍搜索算法，根据查询的距离下界剪枝候选项。据我们所知，这是首次将索引用于子序列搜索并与 MASS 相结合；以往的研究仅分别使用这些技术 [28, 50], , 主张采用基于索引的方法或顺序扫描方法。第三，它采用两种新颖的方法来收紧查询的距离下界，进一步提升搜索的剪枝潜力。这些方法具有通用性，意味着可应用于任何使用 R 树和/或 DFT 近似的搜索方案。我们的主要贡献如下：

- 我们提出 MS-Index，这是一种用于在多变量时间序列上进行 *k*-NN 子序列搜索的新型算法，采用欧几里得距离（第 3 节）。
- 我们提出了一套通用优化方法，用于改进采用 R 树和/或 DFT 近似的任意搜索方案（第3.4节）。这些优化结合使用后，可使 MS-Index 的性能提升 4 倍。
- 我们提供了一种通用方法，可将当前针对单变量时间序列的解决方案扩展到多变量情形，并以此为评估提供基线方法（第 4 节）。
- 我们在 34 个数据集上对 MS-Index 进行了全面评估，并将其与多种基线方法进行比较。结果表明，对于原始和归一化子序列，MS-Index 的性能均比最先进的方法快两个数量级（第 5 节）。

[1]与归一化时间序列相比，支持归一化子序列更具挑战性，因为这无法通过预处理数据来实现。此外，尽管大多数应用都需要归一化子序列来进行形状匹配，但有些应用需要查询原始子序列，以保留尺度差异 [8, 53]。

2 预备知识

2.1 定义与符号

时间序列。长度为 m 的单变量时间序列（UTS）$\boldsymbol{t}$表示为 $\boldsymbol{t} = [t_1, \ldots, t_m]$，其中 t_i 是时间 i 处的数据点。具有 c 个通道的多变量时间序列（MTS）$\boldsymbol{T}$表示为矩阵 $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_c]^T$，其中 $\boldsymbol{t}_i$ 是通道 i 的单变量时间序列。起始时间点为 i、长度为 s 的子序列表示为 $\boldsymbol{T}_i^s = \boldsymbol{T}_{*,\ i:i+s-1}$。数据集中可用通道的数量表示为 c，查询通道的索引集合表示为 ^cQ，其中 $c_Q \subseteq \{1、\ldots\ldots、\ c\}$。

相似性搜索查询。相似性搜索查询有两种形式：k近邻（*k*-NN）查询和r范围查询（也称阈值查询）。 *k*-NN查询定义为：在给定距离度量 d [25]下，找出数据集D中与查询时间序列 Q距离最小的 k 时间序列。对于多变量时间序列，D中的所有时间序列都包含相同的 c 通道。然而，查询时间序列 Q 可能只包含这些通道中的一个子集^cQ。相反，r范围查询定义为：找出D中所有与查询时间序列Q 的距离小于给定阈值 r [25]的时间序列。这些查询类型还可分为全序列匹配查询和子序列搜索查询。在全序列匹配中，我们考虑整个查询时间序列与整个候选序列 [25]之间的距离。搜索涉及的所有序列长度都必须相同。在子序列搜索中，我们考虑整个查询序列与候选序列中所有与查询长度相同的子序列 [25]之间的距离。在这种情况下，提取子序列的候选序列长度不必相同。子序列搜索有两种变体：(a)定长搜索，其中查询长度 $|Q|$ 预先确定，并且在各次查询中保持固定；(b)可变长度搜索，其中查询长度不固定，可在不同查询之间变化 [25]。在本研究中，我们专注于定长子序列搜索问题，这是文献中常见的假设 [12, 16, 28, 71, 72]。可变长度搜索留待未来研究。

距离。与以往关于多变量时间序列相似性搜索的研究一致 [7,12, 71], , 我们使用欧几里得距离（ED）来衡量时间序列之间的距离，其在多变量时间序列上的定义如下：

$$d(\boldsymbol{X}, \boldsymbol{Y}) = ||\boldsymbol{X} - \boldsymbol{Y}||_2 = \sqrt{\sum_{i \in c_{XY}} \sum_{j=1}^{m} (X_{i,j} - Y_{i,j})^2} \qquad (1)$$

其中，$c_{XY} = c_X \cap c_Y$是 $\boldsymbol{X}$和 $\boldsymbol{Y}$的公共通道，m =是最短时间序列的长度 $\min(m_X, m_Y)$。选择欧几里得距离是因为它简单、高效，对单变量数据的性质有深入研究，并且可自然扩展到多变量情形，同时在文献中也很常用 [7, 12, 22, 71]。事实上，尽管动态时间规整（DTW）和基于形状的距离（SBD）等更复杂的度量，通过时间对齐已被证明在单变量和多变量情形下的全序列匹配中很有效 [22, 24, 55, 56, 64, 67],，但在子序列搜索中，使用这些度量相较于欧几里得距离所带来的边际收益微乎其微 [22, 61, 66, 74]。这是因为考察时间序列中的所有子序列，本质上就是一种时间对齐形式：将查询

表 1：术语表

t	单变量时间序列
T	多变量时间序列
$\|t\|$	时间序列长度 T
T_i	T的通道 i
$T_{i,j}$	T的通道 i的数据点 j
T_i^s	从索引 i开始的、长度为 s 的 T 子序列
$\tilde{T}$	T 的离散傅里叶变换近似
$\tilde{T}'$	T 的展平的离散傅里叶变换近似（即特征向量）
Q	查询时间序列
c	数据集中通道的总数
c_Q	Q 的通道 ID 集合
n	数据集 D 中时间序列的数量
m	D 中时间序列的长度

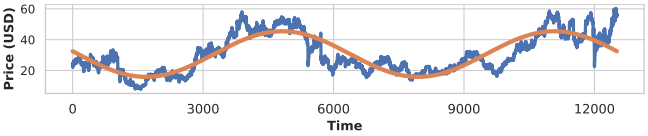

图 2：股票价格随时间的变化（蓝色），以及通过其前 3 个 DFT 系数重建的结果（橙色）。

与候选时间序列在不同位移下进行比较，从而自然地校正时间扭曲。表 1 总结了本文中使用的符号。

2.2 离散傅里叶变换近似

相似性搜索面临的一个关键挑战是数据的高维度，这源于时间序列的长度。解决这一挑战的一种常见方法是采用降维或“摘要”技术，将数据转换到较低维的空间中，从而以较低的成本近似计算距离。欧几里得距离（L_2）的一种常用摘要技术是离散傅里叶变换近似，即对时间序列进行 DFT，并仅保留所得向量 [28, 49, 59, 69] 中的一小部分。DFT 将时间序列分解为不同频率的正弦波之和（也称为系数），其中正弦波的振幅表示相应频率在时间序列中的重要性（也称为该系数的能量 ）。对于大多数现实世界中的时间序列，高频正弦波的振幅较小，因此丢弃这些正弦波也不会损失太多信息。这意味着，只使用 DFT 的前 f 个值，就能准确近似时间序列，得到大小为 f 的向量，而原始向量的大小为 m [28]。图 2 展示了一个来自股票 数据集（参见第 5 节）的示例，说明了这种近似的准确度。DFT 近似常用于估计欧几里得距离，因为两个经过 DFT 近似的时间序列之间的距离是其真实欧几里得距离的下界 [49]。也就是说，两个时间序列 $t, Q \in \mathbb{R}^m$ 之间的欧几里得距离受其 DFT 近似值$\tilde{t}, \tilde{Q} \in \mathbb{C}^f$ 的界定，如下所示 [49]:

$$d(t,Q) \geq \sqrt{\frac{\sum_{i=1}^{f} |\tilde{t}_i - \tilde{Q}_i|^2}{m}} = \frac{d(\tilde{t},\tilde{Q})}{\sqrt{m}} \tag{2}$$

随着 f 的值增大（即纳入更多系数），DFT 界会逐渐收敛到精确距离 [49]。这样，DFT

表 2：相关工作概览

		Name	Ref	Type	备注
UTS	Whole	VA+ 文件	[30]	索引	
		ISAX2+	[16]	索引	
		DSTree	[72]	索引	
	Sub.	ST-index	[28]	索引	
		KV-match	[73]	序列	自定义定义 归一化子序列
		MASS	[50]	序列	
MTS	Whole	Vlachos	[71]	索引	不支持 归一化子序列
	Sub.	Bhaduri	[12]	索引	仅支持 r-范围查询
		MS-Index（我们的方法）		索引	

近似可以在距离估计的准确度与数据维度之间进行权衡，这对于时间序列等高维数据尤其有用。对于多变量时间序列，可以采用相同的思路，分别概括每个通道。

2.3 R 树构建技术

R 树是一种用于空间数据的树形索引，它将相近的对象分组到最小边界矩形（MBR）中，再递归地将这些矩形分组为更大的 MBR [10, 33]。R 树可以自顶向下构建：将包含所有数据点的单个 MBR 拆分为更小的 MBR，所用策略包括二次分裂 [33]，、线性分裂 [33]，或 R*-树拓扑分裂 [10]。另一种方法是自底向上构建（或批量加载）：从将各个数据点视为 MBR 开始，根据叶节点大小 L 和分区策略将它们合并为更大的 MBR。这种方法通常能减少重叠，但要求预先掌握所有数据。一种常用的批量加载算法是排序-切片-递归 （STR）算法 [43]。STR 首先根据每个维度上的坐标对子节点排序（如果子节点是矩形而非点，则使用矩形中点）。然后，它递归地将条目划分为大小为 $\lceil \frac{N}{L} \rceil^{1/d}$ 的组，其中 N 是待索引的条目数， L 是期望的叶节点大小，而 d 是树的维度。例如，给定一个包含条目 $[1, 2, \ldots, 10]$ 和 $L = 2$ 的一维空间，该算法会将条目划分为 $[[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]$,，随后创建节点 $[[1, \ldots, 4], [5, \ldots 8], [9, 10]]$ 等。

2.4 相关工作

我们首先讨论单变量时间序列（UTS）方面的相关工作，随后介绍多变量时间序列（MTS）方面的近期研究。表 2 总结了所有相关工作。

单变量时间序列（*UTS*）全序列匹配。Echihabi 等人对 UTS 全序列匹配算法进行了大规模比较，结果显示，基于索引的算法通常优于相应的顺序扫描算法。此外，对于内存中的小型数据集，VA+ file [30] 和 iSAX2+ [16] 表现出色，而 DSTree [72] 在较大的数据集上占优。DSTree 使用扩展自适应分段常数近似（EAPCA），通过在分辨率逐渐提高时按均值和方差对时间序列分组，估算查询与各组时间序列之间的距离。全序列匹配索引可通过单独索引每个子序列来扩展，以支持子序列搜索。然而，当 $|Q| \ll |T|$时，这种方法会使索引不堪重负，导致性能下降；第 5 节的实验验证了这一点。

UTS 子序列搜索索引。已有多种用于 UTS 子序列搜索的索引。ST-index [28] 针对 r-范围查询，首先提取时间序列中每个长度为 $|Q|$ 的子序列的 DFT 近似。随后，这些 f维向量在 f维空间中形成一条轨迹，并使用最小边界矩形将其分割成若干子轨迹。然后，将这些最小边界矩形编入 R*-树 [10], , 以实现高效搜索。Dual Match 和 General Match [31, 47] 带来了更新的改进，它们使用不同的窗口类型来减小索引大小。然而，这些扩展本质上将索引限制为原始子序列，因为要支持归一化子序列就必须彻底重新设计。我们的工作采用了在 R 树中索引 DFT 近似的思路，但（a）聚焦于多变量时间序列（MTS），（b）不再显式进行时间序列分段，（c）利用卷积定理加快距离计算，并且（d）采用不同的搜索算法，以支持 k-近邻查询，并允许以临时方式选择查询通道。另一个值得关注的索引是 KV-match [73], ，它通过时间序列中不相交窗口的均值和方差，对单个时间序列中的子序列进行索引。该索引既支持归一化子序列，也支持原始子序列；但对于归一化子序列，它会在归一化之前将搜索空间限制为均值和方差与查询相近的子序列。KV-Match 可通过以下方式适配我们的设定：（a）为每个时间序列构建一个 KV-match，并逐一查询；（b）在查询归一化子序列之前，移除针对均值和方差的过滤条件。

其他单变量时间序列子序列索引还包括 L-match [29] 和 TS- index [18], ，我们不再进一步讨论它们，原因如下：L-match 缩短了 KV-match 的索引构建时间，但查询时间更长；TS-index 使用切比雪夫距离，而非我们采用的欧几里得距离。正如我们在第 5 节所示，我们的方法优于 KV-match；根据传递性，这也说明了我们的方法预期与 L-match 的关系。

单变量时间序列子序列搜索顺序扫描。Mueen相似性搜索算法（MASS） [50] 是一种精确的子序列搜索算法，可在 O（$|t|\log|t|$）时间内计算查询 Q 与时间序列 t的所有子序列之间的距离，而非穷举所需的 O（$|t||Q|$）时间。它通过卷积定理实现这一点。卷积定理指出，两个时间序列的互相关（即滑动点积）等价于其傅里叶变换的逐点乘法 [2]。具体而言，卷积定理指出，Q 与 t 中所有长度为 $|Q|$ 的子序列之间的点积 $\cdot,\ \cdot$可通过以下方式计算：

$$[\langle Q, t_1^{|Q|}\rangle, \ldots, \langle Q, t_{|t|-|Q|+1}^{|Q|}\rangle] = \mathcal{F}^{-1}(\mathcal{F}(Q) \otimes \mathcal{F}(t)) \qquad (3)$$

其中，F、F^{-1}和 $\otimes$分别表示傅里叶变换、逆傅里叶变换和逐点乘法。由于两个向量 x和 y之间的欧几里得距离定义为$d(x, y) = \sqrt{||x||^2 + ||y||^2 - 2x,\ y,}$ MASS 使用公式 3 以及 Q和 t的滑动平方和，在 $|t|\log|t|$的时间内计算 Q与 t的所有子序列之间的距离。通过分别对每个通道重复距离计算，并将结果相加以获得多变量欧几里得距离，MASS 可用作多变量时间序列的子序列搜索算法。在本研究中，我们将使用 MASS，既将其作为基线，也将其作为我们方法的一个组成部分。

多变量时间序列整体匹配。Vlachos 等人提出了首个用于多变量时间序列整体匹配的索引 [71], ，该索引除了支持欧几里得距离外，还针对动态时间规整和最长公共子序列（LCSS）距离下的 r-范围查询。其方法是将多变量时间序列拆分为跨越时间、通道和值轴的最小边界矩形，并将其存储在 R*-树中 [10]。随后，在查询时，会为查询构建一个最小边界包络（MBE），以覆盖查询在扭曲条件下所有可能的匹配区域。接着，将该 MBE 进一步分解为最小边界矩形，并在 R*-树中进行探测，以高效地找到候选项。然而，将该方法用于欧几里得距离下的子序列搜索，本质上会简化为 Dual Match 的一个变体 [31], ；正如前文所述，这会导致该索引无法支持归一化子序列，因此不适用于我们的问题设定。

多变量时间序列子序列搜索。 最近，有人提出了一种名为 MULISSE [58], 的新算法，用于可变长度 多变量时间序列子序列搜索。该算法是最先进的单变量时间序列子序列搜索算法 ULISSE [45] 的多变量扩展。该方法将 iSAX2+ [16] 表示扩展到多个通道，并引入通道感知节点分裂以提升剪枝能力。为适应可变长度子序列搜索的庞大搜索空间，ULISSE 索引（因此也包括 MULISSE）采用轻量级设计，以牺牲剪枝能力为代价来减小索引大小。正如我们在第 5 节中所示，只需将查询长度范围限制为单个值，即可将 MULISSE 用于固定长度搜索。

据我们所知，Bhaduri 等人 [12] 提出的方法是唯一一种用于固定长度 多变量时间序列子序列搜索的解决方案。该方法通过根据各通道子序列到单变量参考点的距离进行索引，在欧几里得距离下回答r 范围 查询。然而，该方法依赖通道级阈值，因此无法支持 k-近邻查询；这类查询需要同时查询所有通道。要将该方法改用于我们的问题设置，需要进行非平凡的修改，而且很可能会降低其性能，因此我们在本文中不再进一步考虑该方法。

总结。总而言之，文献中针对不同查询类型和解决方案设计，提出了大量适用于单变量时间序列和多变量时间序列数据的相似性搜索方案。然而，现有方案中只有少数能够直接应用于或扩展到我们的问题设置。具体而言，所有多变量时间序列方案要么（a）不支持 k-NN 查询 [12]或对归一化子序列进行查询 [71]，要么（b）优化目标是索引大小而非查询性能（MULISSE [58]）。有些单变量时间序列方案可以扩展到多变量时间序列，但预计这些扩展方案无法很好地扩展。这是因为它们要么（a）并非为子序列搜索而设计（*DSTree*），要么（b）属于顺序扫描算法（*MASS*），要么（c）仅针对单变量时间序列设计（*ST-index*）。

3 MS-INDEX

多元时间序列子序列搜索面临两大挑战。首先，即使数据集相当小，其中的子序列数量也会迅速增长。对于包含 n 个时间序列、每个长度为 m 的数据集，穷举搜索需要计算所有 $n*(m-|Q|+1)$ 个子序列与查询 Q 之间的欧几里得距离，每次计算都需要耗费 $|c_Q|*|Q|$ 时间。其次，要求

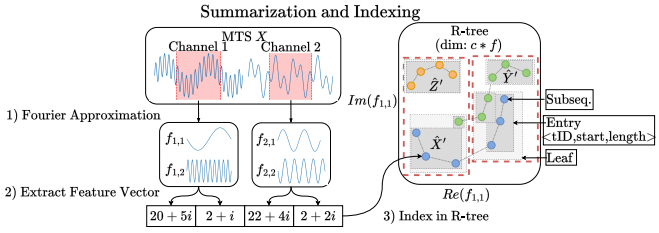

图 3：MTS子序列的汇总与索引。我们用蓝色、绿色和橙色节点表示特征空间中来自三个 MTS 的索引子序列。时间相邻的子序列以线相连。

支持按需选择查询通道，这排除了使用涉及通道合并的汇总方法，例如 PCA [76]。

我们提出的解决方案 MS-Index 结合索引的剪枝潜力与MASS算法在精确距离计算方面的高效性（参见第 2.4 节），以应对这些挑战。该方案包含三个关键步骤（参见图 3）：(a) 通过 DFT 近似汇总 MTS子序列；(b) 基于这些近似构建 R 树索引；[2] (c) 在查询时利用索引大幅剪枝候选子序列，随后使用 MASS 对剩余候选项进行精确距离计算。借助该索引，MS-Index 能以极低的代价，对多种数据集中的候选子序列剪枝超过 99%，相较于最先进的方法，最高可实现 100 倍的加速。我们还提出了多项优化，以进一步提高搜索过程的效率（第3.4节）。这些新颖的优化本身也颇具价值，因为它们可应用于任何使用 DFT 近似和/或空间索引的搜索算法，例如 [20, 28, 30, 49, 59, 71]。

现在我们介绍 MS-Index 的关键组成部分，首先是时间序列摘要（第 3.1 节）和索引（第 3.2 节），随后是查询执行算法（第 3.3 节）以及优化（第3.4节）。

3.1 使用 DFT 总结所有子序列

如第 2.2 节所述，处理时间序列时面临的一大挑战是其长度。使用部分离散傅里叶变换系数来近似时间序列的方法已在文献中得到广泛应用 [28, 30, 49]。这种方法之所以有效，是因为它提供了一种紧凑的表示形式，能够高效地对时间序列之间的距离进行下界估计。然而，我们对现实世界时间序列的分析揭示了两个重要观测结果，可以进一步提高此类近似的准确度；

观察 1： 就对时间序列总能量的贡献而言，前-f个系数并不总是贡献最大的-f个系数；某些高频贡献同样显著的区域也确实存在。这一点在图 4a 的两幅图中有所体现。图中展示了从 100 条包含不同地点真实温度读数的时间序列中提取的离散傅里叶变换系数所对应的总能量和距离的绝对贡献及累积贡献（数据集详情见第 5 节）。两条曲线均显示了涵盖大量测量值的 90% 置信区间（即能量曲线由全部 100 条时间序列得出，而总距离曲线由全部 100*99/2 个成对距离得出）。图中显示，第 62 个系数的能量贡献突然增大，表明该系数对时间序列的形状有显著贡献，因此应纳入近似。与其他自适应摘要技术的配置过程类似，最显著的前-f个系数可通过在索引构建前对数据集样本计算统计量来确定 [32, 54, 75]。

观察 2： DFT 系数对时间序列之间总距离 的贡献，比它们对能量的贡献更为显著。直观来说，这意味着排名靠前的系数不仅尺度更大，在不同时间序列之间的变化也大得多。这一点在图 4a 中显而易见：图中显示，距离的累积贡献增长速度远快于能量的累积贡献，这意味着仅用排名前 5 的系数就能覆盖总距离的 ~90%，尽管这些系数仅覆盖约 ~60% 的总能量。

MS-Index 利用这些观察结果，针对给定数据集生成准确的时间序列子序列近似。具体而言，我们按如下方式概括数据集中的多变量时间序列。首先，在索引构建之前，我们从数据集[3],中的不同时间序列中均匀随机抽取 S 个长度为 $|Q|$ 的子序列样本，并计算每个 DFT 系数和每个通道对这些子序列之间欧几里得距离的平均相对距离贡献（ARDC），即生成图 4a 所示曲线的过程。然后，对于数据集中的每个通道，选出 ARDC 最大的前 f 个系数，并选择 f，使得这前 f 个系数的 ARDC 总和超过某个阈值 d_{target} （例如 90%）。相较于以 f 为参数，将 d_{target} 作为算法参数能够使算法根据给定数据集调整概括的规模，从而确保近似达到预定的准确度。随后使用这些排名靠前的 f 个系数，计算数据集中所有 长度为 $|Q|$的子序列的 DFT 近似。请注意，公式 2 中的求和可以针对系数的任意排列或子集进行；只要两个子序列使用相同的系数，该界限仍然成立。

˜接下来，我们用 T表示大小为 $c \times f$的矩阵，该矩阵存储子序列 T的 DFT 近似。此外，$\tilde{T}$,将用于表示子序列 T的特征向量，其长度为 $c * f$，通过展平矩阵 T 构建（图 3 中的步骤 2）。汇总包含 n 条时间序列的数据集需要 $O(n*m*|Q|*f)$，并会生成总计 $O(n*m)$个特征向量——每个子序列对应一个。

[2]我们希望强调，尽管使用了 DFT（这是一种近似技术），MS-Index 仍然是一个精确的算法。近似只会影响 MS-Index 的界估计效率和剪枝能力（即速度），而不会影响其结果的准确度或完整性。具体而言，应用降维后，R 树中对象之间的距离是实际距离的下界。这可能导致假阳性，但绝不会导致假阴性，因为我们使用一个作为实际 k-NN 距离上界的阈值查询 R 树（即未进行降维时）。我们将在引理3.1中对此进行形式化证明。

[3]样本量是算法的一个参数，通常设为 100 个子序列。该数值是在不同数据集上进行实证测试后选定的，测试结果表明，增大样本量并不会显著提升性能。

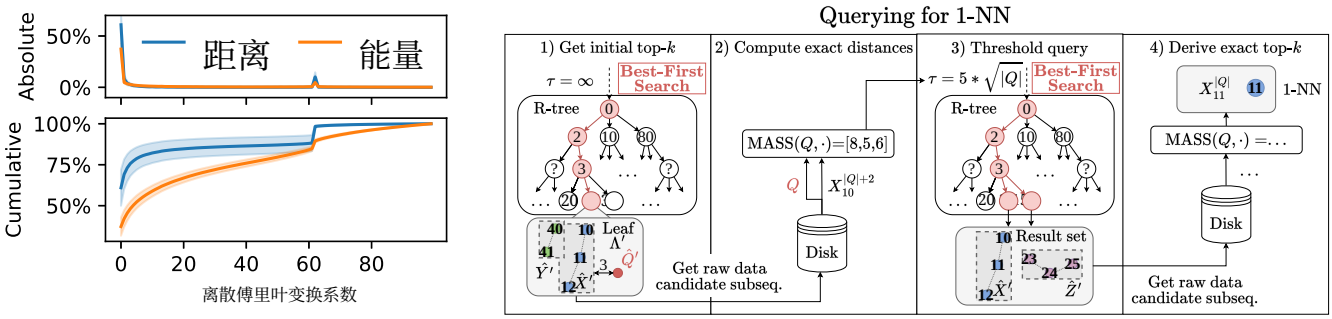

图4：（a）天气数据集的温度通道中，各离散傅里叶变换系数对应的总距离和能量的累积百分比与绝对百分比；（b）MS-Index中的查询执行。树节点中的数字表示相应最小边界矩形到查询的距离下界。

3.2 多变量时间序列的索引

下一步是将生成的特征向量索引到 R 树中（图 3 中的步骤 3）。[4] 然而请注意，特征向量总数为 $O(n * m)$，当时间序列数量 n 较大时，这可能导致 R 树规模庞大且效率低下，从而对查询性能产生负面影响。

我们通过以下方式缓解这个问题。首先，我们以自底向上方式构建 R 树，使叶节点包含多个条目（即叶节点大小为 $L > 1$），从而减少树中的节点数量。其次，在构建叶节点后，我们重新检查它们并修改其内部表示，以便通过一次距离计算剪枝多个子序列（即条目）。具体而言，对于每个包含两个或更多条目的叶节点，我们将所有源自同一多变量时间序列中时间相邻子序列 的条目分组，也就是说，这些特征向量对应一系列彼此仅相差一个时间偏移的子序列（例如，$T_i^{|Q|}$、$T_{i+1}^{|Q|}$、$T_{i+2}^{|Q|}$ ...）。这些分组对应于原始多变量时间序列中的连续子序列。例如，假设某个叶节点 Λ 包含以下子序列：$\Lambda = \{X_{10}^{|Q|}, X_{11}^{|Q|}, X_{12}^{|Q|}, Y_{40}^{|Q|}, Y_{41}^{|Q|}\}$，其中 X 和 Y 是两个不同的多变量时间序列。我们将这五个条目压缩为两个，分别表示相对较大的子序列 $\Lambda' = \{X_{10}^{|Q|+2}, Y_3^{|Q|+1}\}$。这些条目现在以最小边界矩形（MBR）而非特征空间中的单个点表示，连续子序列的起始位置和结束位置则与 MBR 一同存储在条目中。

这种分组背后的直觉是，由于时间上相邻的子序列存在大量重叠，它们通常非常相似 [28]。这意味着它们的特征向量也相似，往往会落在同一个 R 树叶节点中。因此，许多子序列可以用具有紧致 MBR 的紧凑条目表示，这既减少了 R 树中的条目数量，也能借助 MASS 高效地进行距离计算。在我们的实验中，这种压缩通常会将 8 到 50 个子序列合并为一个条目，具体取决于数据集；叶节点大小则针对查询性能进行了优化。

因此，MS-Index 的索引过程如下 (a) 我们基于数据集中所有子序列的特征向量，自底向上构建一棵 R 树。在包含 X 和 Y 的运行示例中，这一步涉及创建叶节点 Λ，以及创建包含数据集中其他子序列的所有其他叶节点。在 R 树的递归构建过程中，我们随后创建 Λ 的父节点，这些父节点包含叶节点的最小边界矩形（其自身也是最小边界矩形），如此继续，直到到达根节点。(b) 然后，我们重新访问叶节点，将所有时间相邻条目归为一组（即创建 Λ'），以减小索引大小。完成这些步骤后，我们得到一棵 R 树，其中每个条目存储：(a) 覆盖所索引特征向量的最小边界矩形；(b) 子序列在原始多变量时间序列中的起始位置和结束位置；以及 (c) 多变量时间序列标识符。该索引显示在图 3 的右侧，其中节点表示不同多变量时间序列的已索引子序列（以颜色区分），边则连接同一多变量时间序列中时间相邻的子序列。

3.3 查询执行

我们的算法会对索引进行两次探测。第一次探测用于粗略估计到第 k 个最近邻的距离（即一个上界）；第二次探测则检索所有距离不超过该上界的条目，并计算精确结果。为了确保答案正确，第二次探测不可或缺。

查询的执行过程如下。首先，我们提取查询 Q 的特征向量 $\tilde{Q}'$。然后，从距离阈值 $\tau_k = \infty$ 开始，在 R 树上执行最佳优先搜索，以找到与 $\tilde{Q}'$ 距离最小的 k 个条目（即长度为 $|Q|$ 的子序列组）。由于 R 树依赖特征向量来计算所有距离，因此计算得到的距离是到 Q 的真实距离的下界（LB）。接着，对于每个检索到的条目，我们使用 MASS 计算 Q 与该条目中长度为 $|Q|$ 的子序列之间的精确距离，并将距离阈值 τ_k 设为此步骤中找到的第 k 小距离。最后，我们以 $\tau_k * \sqrt{|Q|}$ 作为距离阈值，对索引执行阈值查询，并使用 MASS 计算这些条目的精确距离，从而得到最终的 k-NN。[5] 随后，我们将这些结果连同相应子序列的时间戳一起返回给用户；这些时间戳

[4] 我们还尝试了其他空间索引，例如 KD树， [11], 但发现，我们算法的召回阶段才是整个方案中最耗时的部分，而不是索引探查。因此，我们选择使用 R 树，以便与 ST-index 进行更清晰的比较 [28]。

[5] 请注意，MASS 并非在特征向量上执行，而是在原始子序列上执行，以确保距离精确。通过沿着 R 树条目中存储的指针，访问存放在磁盘上的原始多变量时间序列，即可检索这些数据。

由多变量时间序列的偏移量和时间戳得出。由于查询 Q不一定包含所有通道（即 $c_Q \subseteq \{1, \ldots\ldots, c\}$），因此只会在特征空间中与 Q所含通道对应的维度上查询 R 树。R树算法 [65] 原生支持此操作。

为说明这一过程，考虑查询 Q，其最近邻是上一节中叶节点 Λ' 里的子序列 $X_{11}^{|Q|}$。算法会在第一次探测时返回条目 $X_{10}^{|Q|+2}$，因为它到 Q的距离下界最小（图 4b 中的步骤 1）。随后，算法在原始子序列上计算 MASS$(Q, X_{10}^{|Q|+2})$，得到距离 $[8, 5, 6],$，并将 $X_{11}^{|Q|}$加入当前最近邻集合，因为它对应的距离 5 最小（步骤 2）。因此，阈值设为 $5 * \sqrt{|Q|}$，算法执行第二次探测，取回条目 $Z_{23}^{|Q|+2}$和 $X_{10}^{|Q|+2}$（步骤 3）。使用 MASS 计算这些条目的精确距离后，算法返回 $X_{11}^{|Q|}$作为 Q的最近邻（步骤 4）。

现在，我们证明上述查询过程能够保证返回查询 Q 的完整且正确的 k-NN。具体而言，我们旨在证明以下引理:

引理3.1。 任何属于 Q 的 k-*NN* 的长度为 $|Q|$ 的已索引子序列 T，都保证包含在 *MS-Index* 返回的子序列集合中。

证明。索引的首次探测会返回 k 个长度为 $s \geq |Q|$ 的子序列，并计算它们与查询之间的精确距离。由于这些 k 个子序列合起来至少包含 k 个长度为 $|Q|$ 的子序列，因此将 τ_k 设为这些距离中第 k' 小的值，即可得到真实 k-近邻距离的一个上界。在第二次探测中，我们在包含特征向量的 R 树索引上执行阈值为 $\tau_k * \sqrt{|Q|}$ 的范围查询。根据公式 2，特征空间中任意两点之间的距离，都是它们所表示的子序列之间真实距离的下界。具体而言，长度为 $|Q|$ 的子序列 T 与查询 Q 之间的距离，下界为其特征向量 T' 和 Q' 之间的距离 $\tilde{}$ $\tilde{}$:

$$d(T, Q) \geq \frac{1}{\sqrt{|Q|}} \sqrt{\sum_{i \in c_Q} d(\tilde{T}_i, \tilde{Q}_i)^2} = \frac{d(\tilde{T}'_{c_Q}, \tilde{Q}'_{c_Q})}{\sqrt{|Q|}} \tag{4}$$

$\tilde{}$ $\tilde{}$其中， T'_{c_Q}和 Q'_{c_Q}分别对应于 T和 Q在 Q通道范围内的特征向量。现在，假设索引子序列 T与 Q之间的距离小于 τ_k。那么，根据公式4，我们知道 $\frac{d(\tilde{T}_{c_Q}, \tilde{Q}'_{c_Q})}{\sqrt{|Q|}} \leq d(T, Q) \leq \tau_k$。因此，通过查询 R 树并设置阈值 $\tau_k * \sqrt{|Q|}$，可保证 T包含在返回结果集中。[6] □

该引理通过保证找到 k-近邻中的所有子序列，证明了 MS-Index 的完备性。由于使用 MASS（其结果是精确的）为所有候选项计算精确距离，并根据这些距离得出 k-近邻，因此该算法也是正确的，不会返回假阳性。

[6]请注意， $\sqrt{|Q|}$ 只是一个缩放因子，在将距离从特征空间转换到时域时需要考虑它。它不会削弱界。

3.4 优化

现在，我们介绍 MS-Index 的三种优化，旨在增强索引的剪枝能力并提高搜索过程的效率。这些优化不会影响算法的正确性或最终结果——该算法仍然保持精确。

收紧 *DFT* 界。如第 3.1 节所述，在大多数真实世界数据集中，时间序列之间约 60%–80% 的距离由排名前 2–5 的 DFT 系数覆盖。形式化地说，这一概念可表示为:

$$d^2(T, Q) = (\underbrace{d^2(\tilde{T}_f, \tilde{Q}_f)}_{\sim 80\%} + \underbrace{d^2(\tilde{T}_{f+}, \tilde{Q}_{f+})}_{\sim 20\%})/|Q| \tag{5}$$

$\tilde{}$其中， T_f表示使用$\tilde{}$前f个系数对多变量时间序列 T进行离散傅里叶变换近似后得到的结果，而 T_{f+} 表示由剩余系数组成的 c 向量集合。在公式4的离散傅里叶变换界中，我们实际上舍弃了第二项，因此得到的是真实距离的一个下界。该下界的计算效率很高，复杂度为 $O(f)$，但也可能削弱 R 树的剪枝效果，导致对最终 k-近邻之外的子序列进行不必要的距离计算。

我们通过在距离计算中引入一个校正项来改进这个下界，使其更接近真实距离。该校正项对剩余的（$|Q| - f$）个系数上的距离进行近似（同样通过下界估计），而无需实际计算这些系数。对于每个被探测的 R 树节点，在查询时计算该校正项的开销为 $O(1)$，并能提升索引的剪枝能力。

也就是说，在索引构建过程中，为每个子序列 T 计算离散傅里叶变换近似后，我们还会推导出 T 中未被前 f 个系数覆盖的部分，称为其 余项 $T_{f+} = [T_1 - \text{IDFT}(\tilde{T}_{f,\ 1}) \ldots,\ T_c - \text{IDFT}(\tilde{T}_{f,\ c})]$。该余项仅基于前 f 个系数即可计算，无需计算所有其他的（$|Q| - f$）个系数。这里的关键观察是，两个时间序列的余项之间的距离反映了它们距离中剩余的 ~20%，如公式 5 所示。因此，公式 5 可以改写为利用余项的形式:

$$d^2(T, Q) = \frac{d^2(\tilde{T}_f, \tilde{Q}_f)}{|Q|} + d^2(T_{f+}, Q_{f+}) \tag{6}$$

余项的长度为 $|Q|$，这意味着其代价相当于在原始数据上计算完整距离（即 $O(|c_Q| * |Q|)$）。为避免在查询时产生这项开销，我们在索引阶段预先计算每个子序列的余项到一小组固定枢轴点 的距离，并将这些距离存储在映射中。然后，在查询执行期间，我们对 Q_{f+} 执行相同的操作，并利用反三角不等式，根据 Q_{f+} 和索引子序列余项到枢轴的距离，对它们之间的距离进行下界估计。我们将后一个下界称为校正项。我们将校正项应用于公式 6，如下所示:

$$\begin{aligned} 0 \quad &\leq (|d(T_{f+}, P) - d(Q_{f+}, P)|)^2 \leq d^2(T_{f+}, Q_{f+}) \Rightarrow \\ d^2(T, Q) \quad &\geq \underbrace{d^2(\tilde{T}_f, \tilde{Q}_f)/|Q|}_{\text{DFT distance}} + \underbrace{(|d(T_{f+}, P) - d(Q_{f+}, P)|)^2}_{\text{Correction term}} \end{aligned} \tag{7}$$

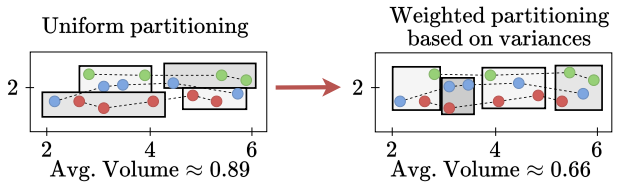

图 5：二维特征空间的不同分区策略；STR 算法（左）和所提出的加权分区（右），后者可生成更小的最小边界矩形，并得到更紧的界限。

其中 P 是一个枢轴。在索引构建阶段，我们通过对数据集中子序列余项的一个样本运行 k-均值聚类，得到 k 个枢轴。然后，在查询执行期间，算法会找出距离查询余项最近的枢轴，并根据公式 7 使用该枢轴 P 来计算下界。在第 5 节中，我们将展示这一优化可使查询执行速度提升 2 倍。

收紧最小边界矩形。通过减小 R 树节点中最小边界矩形的体积，可以进一步提升 R 树的剪枝效率。R 树采用自底向上方式构建，使用的是第 2.3 节中简要介绍的 STR 算法 [43], 。STR 将条目划分为每个维度上数量相等的分区；当数据在所有维度上均匀分布时，这种方法效果良好，但当数据分布偏斜或集中在空间的某些区域时，可能导致次优分区。为说明原因，考虑一个简单示例：使用 R 树为二维空间建立索引，其中第一维的取值范围为 2 至 6，第二维的取值范围为 1.5 至 2.5（见图 5）。第一维很可能是不同条目之间欧几里得距离的主要影响因素。因此，若在构建 R 树时为第一维分配更多切分/分区，生成的最小边界矩形在该维度上将更紧，从而得到更紧的距离下界，并实现更积极的剪枝。图 5 展示了这一思路。

DFT 近似尤其容易出现这种情况：方差高度集中在最初几个维度上，正如第 3.1 节所述。我们利用这一观察，根据各维度取值的方差对 R 树的维度（即各通道上的 DFT 系数）赋予权重，因为方差是衡量该维度对点间距离贡献的良好代理指标。具体而言，在构建 R 树之前，我们使用汇总步骤（第 3.1 节）中提取的子序列样本 S，估计特征空间中每个维度 i 上特征向量分布的方差 $\hat{\sigma}_i^2$。接下来，我们通过对方差 [15]进行 softmax 归一化，计算ω_i（维度 i的权重）。然后，给定期望的叶节点大小 L，维度 i上的分割数 p_i由 $p_i = \lceil (N/L)^{\omega_i} \rceil$确定，其中 N是整个数据集中待索引的条目总数。 p_i的这一定义可确保当 $\prod_{i=1}^{c*f*2} \lceil (N/L)^{\omega_i} \rceil \approx \frac{N}{L}$ 时达到期望的叶节点大小 L。这样，算法便能有效地在各维度间重新分配分区数量，使其与各维度对距离的贡献成正比。我们将在第 5 节展示，这项优化能够带来

算法 1： UTSBaseline($\mathcal{I}$ Q, k)
输入： 一组通道级索引 I、一个查询多变量时间序列 Q，以及结果集大小 k。
输出： T 中与 Q距离最近的前个子序列。

```
R̂ ←{}
对 i ∈ c_Q 执行                                  // 遍历索引
    R̂ ← R̂ ∪ 查询(I_i, Q_i, k)                    // 查询索引
R̂ ←穷举TopK(R̂ Q)                                // 计算估计的 top-k
对 i ∈ c_Q 执行                                  // 设置阈值
    τ_i ←最大值_{S∈R} d²(S_i, Q_i);
R ←{};
对 i ∈ c_Q 执行                                  // 重新查询
    R_i ← 查询(I_i, Q, τ_i);
    R ← R ∪ {T | T ∈ R_i ∧ d²(Q_i, T_i) ≤ τ_i}
返回 ExhTopK(R Q);
```

索引剪枝更加积极，因此查询执行速度可提升 2–4 倍。

距离浏览。 请注意，由于第二次探测所使用的阈值 τ_k 设为第一次探测返回的子序列中第 k小的精确 距离，因此第二次探测保证会返回第一次探测所返回条目的超集。为避免冗余的距离下界计算，我们在两次探测之间保留最佳优先搜索所用的条目优先队列，因此第二次探测可以从第一次探测停止的位置继续进行。这种优化常用于查询空间索引，通常称为距离浏览 [35]。

4 将现有算法扩展至多变量情形

如第 2.4 节所述，已有多种用于 UTS 相似性搜索的高效算法。因此，一个自然的问题是，这些算法能否扩展到查询多变量时间序列，以及它们在这种情况下的表现如何。现在，我们将介绍一种统一的扩展方案——包装算法，可用于将任何 UTS 搜索算法扩展为适用于多变量时间序列。我们还注意到，与 MS-Index 相比，这种替代设计方案存在若干缺点，例如当通道之间不相关时，剪枝能力较弱，如第 5 节所示。尽管如此，这仍是一种有用的方法，可用于比较 MS-Index 与直接扩展到多变量时间序列数据上的 UTS 算法，例如 DSTree [72], ST-index [28], 和 KV-match [73] 。

该通用方法基于广为人知的阈值算法 [27], 。该算法可用于从同一对象的多个属性值各异的有序列表中得出全局前-k 名，前提是提供一个单调聚合函数来计算用于确定前-k 名的目标值。该方法的步骤如下（参见算法1）：(a) 为每个通道初始化一个索引（例如，一个 DSTree 或一个 ST-index）。(b) 在查询时，首先通过查询相应索引，为每个通道 $i \in c_Q$ 获取一个初始前-k 名估计值 R_i（第 2–3 行）。(c) 对局部估计值并集中的每个子序列，计算其与查询的完整距离（即使用所有查询通道），构造一个中间

全局前-k 名R（第 4 行）。(d) 对每个通道 $i \in c_Q$，将该通道上 R 中最大的单变量 欧几里得距离设为距离阈值 τ_i（第 5–6 行）。(e) 最后，使用各自的阈值 τ_i重新查询各通道级索引。我们计算结果与查询之间的完整距离，相应更新全局前 k 名，并将其作为最终结果返回（第 7–11 行）。由于属于全局前-k 名的任何子序列，在至少一个通道 $i \in c_Q$上的距离都必须不大于 τ_i，因此可以保证该算法的结果正确。最后，使用 MASS 加速精确距离计算（第 4 行和第 10 行）。

5 评估

我们的实验有三个目的：(a) 评估 MS-Index 在各种查询配置下的可扩展性和效率；(b) 将 MS-Index 与其他方法进行比较；(c) 测试第3.4节提出的优化措施是否有效。由于我们的方法和所有基线方法都是精确的，因此评估仅关注效率；准确度实验的结果始终会是 100%。

比较方法。 首先，我们将 MS-Index 与 **MULIS-SE** [58], 进行比较：通过将支持的查询长度范围限制为 $|Q|$，将其用于固定长度搜索。这样，MULISSE 也能利用预先已知的查询长度，从而略微提升性能，同时不影响结果质量。由于没有其他方法原生支持 k-NN 多元时间序列子序列搜索，我们采用的其他基线方法是通过算法 1 扩展的单变量时间序列搜索领域的 SOTA 方法：[7] (a) **ST-Index*** [28], 一种广为人知的子序列搜索索引； [48, 77], (b) **KV-Match*** [73],一种用于单条长序列子序列搜索的 SOTA 索引，并通过为每个多变量时间序列建立索引进行了扩展；以及 (c) **DSTree*** [72], 一种用于全序列匹配的索引， [25], 通过为所有子序列建立索引进行了调整。我们还纳入了两种顺序扫描基线方法：(a) **MASS** [50] 应用于所有多变量时间序列；(b) **暴力搜索** 穷举比较。关于这些比较方法的详细说明，请参见第 2.4 节。

硬件和实现。 所有实验均在一台服务器上运行，该服务器配备 64 核、主频 2.4 GHz 的 AMD Genoa 9654 处理器和 128 GB 内存。MULISSE (C++) 的代码由 [58], 的作者提供，并按照论文中的说明，直接使用原始参数运行。DSTree* (Java-11) 的代码由 [72], 的作者提供，并作为子程序在算法 1中调用。由于没有公开可用的代码，其他所有算法均使用 Java-11 从头实现。[8]为确保比较公平，所有方法均实现为完全在主存 中运行，并以单线程方式执行。

数据集。 我们使用了来自不同领域的32个真实世界公开数据集，以及一组合成数据集：(a) **股票**。1987年1月2日至2021年2月26日期间28678只股票的每日成交量、开盘价、收盘价、最高价和最低价。(b) **天气**。ISD 天气数据集的一个片段 [52] ，包含在2021年1月至2020年12月期间从13545个传感器采集的每小时风速、海平面气压、大气温度和露点温度读数31。(c) **风**。 一台运行中的风力涡轮机每2秒采样一次的传感器输出，持续10天 [51], , 形成一个包含432,000个观测值和10个通道的单一MTS，涵盖功率输出、转子转速、风速及其他风相关变量。(d)**合成数据**。

[7]我们在它们的名称后加上星号，以区别于原始版本。[8]我们的代码可在 https://github.com/JdHondt/MS-Index 获取

表3：默认查询配置。

	股票	天气	合成数据	Wind	UEA（30 倍）
n	2000	1300	1600	1	全部，参见 [7]
平均值 m	5590	8692	4096	432,000	参见 [7]
$\|Q\|$	730（2 年）	1488（2个月）	1024	1800（1小时）	20%
#通道	5	4	64	10	参见 [7]

包含1600个MTS的随机游走数据集，每个MTS具有64个通道和4096个观测值。随后 [25, 26], , 这些数据集通过从均值为 0、标准差从区间 $[0, \ldots, 10]$中均匀随机抽取的正态分布中生成随机数，并将起始点从区间 $[0, \ldots, 100]$中均匀随机抽取来生成。(e) **UEA档案库**。 用于MTS分类的热门基准档案库，由来自不同领域的30个带标签真实世界数据集组成 [7], , 其中MTS、通道和观测值的数量各不相同。每个数据集的训练集和测试集划分均进行了合并，以形成单个数据集。

我们评估了原始和归一化子序列，重点关注原始结果，仅在归一化结果能提供额外见解时才予以强调。由于规模较大，Stocks数据集是我们的主要基准；其他数据集则在指定的实验中讨论，或在它们揭示显著模式时加以讨论。表3总结了各数据集的默认查询确认信息。我们在代码仓库中提供了有关这些数据集的更多详细信息。[8]

查询与评估指标。 按照标准做法 [25], , 我们通过从数据集中随机选取长度为 $|Q|$ 的子序列，并在所有通道上添加标准差为 $0.1 * \sigma_{Q_i}$ 的高斯噪声，为每个数据集生成查询工作负载。作为敏感性分析实验的一部分，我们还在第5.2.6节研究了不同噪声级别以及从数据集之外的子序列生成查询的情况。除非另有说明，我们在数据集的所有通道上进行查询。查询通道子集时（如第 5.2.7 节所示），从所有可用通道中均匀随机选取查询通道。对于每种算法，我们测量：（a）初始化时间，例如索引构建时间；（b）查询执行时间；以及（c）索引和/或辅助数据的大小。我们将总执行时间的超时设为 12 小时，并报告 10 次运行（每次运行包含 100 个查询）的中位数结果。

5.1 算法调优

5.1.1 MS-Index 和 *ST-Index** 的 *DFT* 系数数量回顾第 3.1 节，DFT 系数的数量 f 是根据前 f 个系数覆盖的距离 d_{target} 推导得出的，而不是直接将系数数量作为参数。对于这两种算法，我们通过网格搜索在所有数据集上进行调优，搜索值为 [20%、40%、.....、100%], , 并对原始和归一化子序列进行查询。结果表明，对于这两种算法，在各数据集上，覆盖率设为 60% 是最稳健的选择，其查询时间至多比每个数据集的最优选择多 20%。因此，在后续实验中， d_{target} 被设为 60%。

5.1.2 叶节点大小 DSTree*、ST-Index* 和 MS-Index 支持调整叶节点大小 L，即每个叶节点所包含条目的最大数量。与 DFT 系数数量类似，我们通过网格搜索来调整叶节点大小，以优化查询时间；搜索范围从索引条目总数的 0.0001% 到 100%（即不限制叶节点大小）。

表 4：MS-Index 在不同叶节点大小下的平均查询时间（毫秒）。叶节点大小表示为各数据集（原始数据集）中子序列总数的百分比。

叶节点大小	Insectwingbeat	股票	合成数据	天气
0.0001 %	**8.61**	8.50	2.98	4.67
0.01 %	**8.61**	8.37	**2.97**	4.64
0.05* %	9.01	**8.16**	**2.98**	**4.61**
0.1 %	10.79	8.36	3.01	4.69
1 %	22.70	11.83	3.62	5.06
100 %	38.12	158.10	4.59	100.87

表5：不同Stocks数据集大小下不同算法的索引大小（MB）及其占数据集大小（MB）的相应百分比。

	Raw				归一化			
n	1000	2000年	3000	%	1000	2000年	3000	%
数据集大小（MB）	210	430	641	100%	210	430	641	100%
MULISSE	14	15	16	5%	16	18	20	5%
KV-Match*	259	536	793	124%	10	20	30	5%
MASS	315	646	964	150%	315	646	964	150%
MS-Index	968	1923	2861	452%	1094	2248	3348	522%
DSTree*	1491	3230	4773	731%	620	1276	1900	296%
ST-Index*	2904	5771	8584	1355%	3282	6746	10045	1567%

查看表4中的结果可知，对于 MS-Index，0.05% 的叶节点大小在所有数据集上都能稳定地提供良好性能。因此，所有实验均采用这一叶节点大小。对于 ST-Index*，最优叶节点大小同样为 0.05%。对于 DSTree*，最优叶节点大小为 10%。这与 Echihabi 等人 [25], 的结果相矛盾：他们指出，对于长度为 256 的时间序列，查询时间的最优叶节点大小为 0.9%。在我们的实验中，0.9% 和 10% 的查询时间相近，但选择 10% 可显著降低初始化成本。

*5.1.3 KV-Match** 分段大小 KV-Match* 中的分段大小参数控制用于为每个子序列建立索引的分段均值和方差的数量。我们发现，对于所有数据集，最优分段大小均为 1。对于原始子序列，这会将索引简化为每个通道和时间序列各一个查找表。对于归一化子序列，由于均值为零、方差为单位方差，所有条目最终都落入同一个桶，从而使 KV-Match* 实际上退化为带有额外开销的 MASS。

5.2 评估结果

5.2.1 初始化时间 图6a-b展示了随着MTS数量（n）增加，各种基于索引的算法在Stocks数据集上的初始化时间。所有方法的耗时都随 n线性增长，这符合预期，因为这些方法在初始化时都需要遍历所有时间序列子序列。MS-Index和ST-Index* 的初始化时间相近，主要耗时都用于计算DFT近似。其余初始化成本来自索引构建；对于ST-Index*，这包括构建通道级树（耗时随通道数量线性增长）。相比之下，MS-Index为所有通道构建一棵树，因此在这一阶段快2至3倍。对于归一化子序列，DSTree*的初始化时间与其他方法相近，但对于原始子序列，其耗时则慢了两个数量级。这一差异是因为尺度和均值各异的原始子序列会使DSTree*构建出深度较大且不平衡的树，导致节点分裂操作成本高昂；而归一化子序列则会根据模式差异形成更平衡的树。KV-Match*和MULISSE的初始化时间最短。这是因为KV-Match*只需为每条时间序列创建一个小型查找表，而MULISSE最初就是为可变长度子序列搜索而构建和设置参数的，这为问题增加了一个自由度，也使索引性能成为关键优先事项。

5.2.2 数据结构大小表5展示了各算法在 Stocks 数据集上的内存需求，既以 MB 为单位，也以占数据集大小的百分比表示。所有方法的内存占用都随 n线性增长，其中 MULISSE 的空间效率最高（仅在浅层 iSAX 树中存储n 包络），其次依次为 KV-Match*、MASS、MS-Index、DSTree* 和 ST-Index*。MS-Index 和 ST-Index* 的最坏情况空间复杂度为 $O(n*(m_{\max}-|Q|)\ *f*c)$；由于采用了通道级索引，ST-Index* 所需空间约为 MS-Index 的三倍。比较各方法的相对内存需求可见，许多方法存储索引所需的空间超过数据集大小的 100%。这看似有悖常理，但请记住，这些结构索引的是子序列空间；由于子序列彼此重叠，该空间远大于数据集。举例来说，虽然 Stocks 的 $n = 1000$ 子集大小为 210 MB，但存储所有长度为 $|Q| = 730$的子序列需要 133 GB 内存，这意味着 MS-Index 已将该空间压缩了 138 倍。尽管如此，子序列索引的内存占用仍可能令某些应用有所顾虑，这也是这些方法公认的一项局限 [45, 60]。

5.2.3 查询时间 图6c-d展示了在 Stocks 数据集的时间序列子集上执行查询时，各种方法的查询执行时间。可以看到，MS-Index 的性能显著优于所有其他方法。事实上，它比最接近的竞争方法 MASS 快两个数量级以上，比其他方法快三个数量级以上。这些结果可以解释为：MS-Index 增强了MASS算法的剪枝能力，而 MASS算法本身相较于暴力算法已能大幅提升性能。进一步调查发现，在所有数据集上，MS-Index 的剪枝有效性中位数达到 99%，也就是说，99% 的子序列已从索引中剪除，无需与 MASS 进行比较。尽管 ST-Index*、KV-Match* 和 DSTree* 也会在 MASS 之上执行预过滤，但它们的剪枝能力明显弱于 MS-Index：在 Stocks 数据集的原始子序列上，这些方法的平均剪枝有效性分别为 52%、65% 和 46%。这是因为这些方法的剪枝能力取决于子序列相对于部分（逐通道）结果的相对排序。当不同查询通道之间的相对排序差异显著时，逐通道距离的阈值会相对较高（在算法1的第5–7行中设定），从而导致大量基于MASS的比较。MS-Index 通过同时查询所有查询通道，避免了这一问题。最后，MULISSE 的速度倒数第二（仅快于暴力算法），剪枝率仅为 9%。这表明，尽管其极其节省内存的设计使其索引大小在所有方法中最小，但这种设计

[9]在调整离散傅里叶变换系数的数量后，我们调整了 ST-Index* 的叶节点大小，第一步采用 0.1% 的叶节点大小。

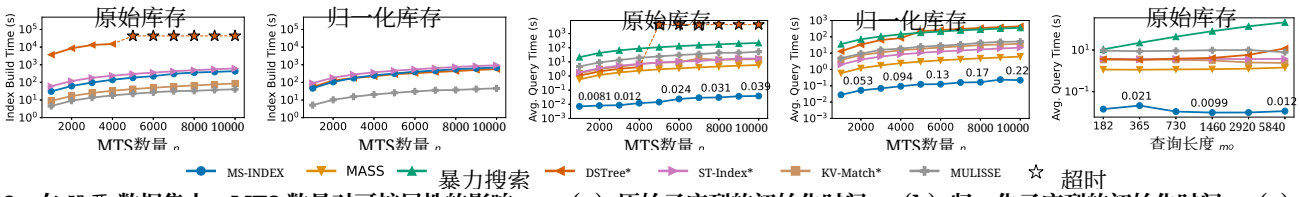

图6：在 股票 数据集上，MTS 数量对可扩展性的影响 n。（a）原始子序列的初始化时间；（b）归一化子序列的初始化时间；（c）原始子序列的查询时间；（d）归一化子序列的查询时间；（e）不同查询长度下原始子序列的查询时间。所有 y 轴均采用对数刻度。

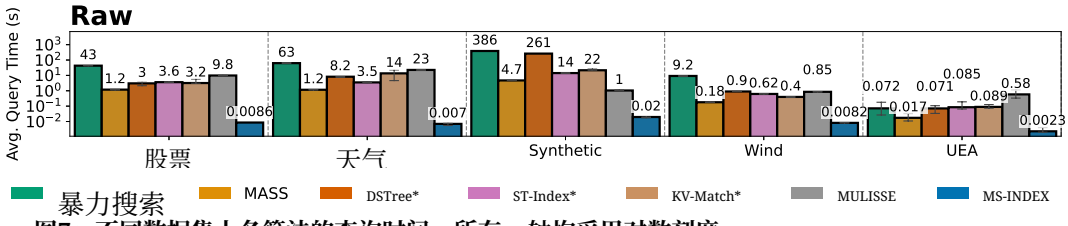

图7：不同数据集上各算法的查询时间。所有 y 轴均采用对数刻度。

是以距离界限较宽松为代价的，因此会导致大量穷举比较。

5.2.4 查询长度由于必须在构建索引之前指定查询长度，我们评估了它对性能的影响。图6e显示，除暴力算法外，所有算法的性能均不受查询长度影响，因为它们使用的 MASS 的计算复杂度随子序列长度（$O(m \log m)$，其中 $m \geq |Q|$）变化，而非随查询长度变化。这表明 MS-Index 的性能与 $|Q|$ 无关，用户可以选择最适合其使用场景的长度。

5.2.5 不同数据集图7展示了使用表3中的默认查询参数时，各数据集上的查询执行性能。[10] 对于30个UEA数据集，由于各数据集的结果呈现相似模式，我们展示其平均结果。除UEA数据集外，MS-Index在所有数据集上的表现始终比竞争方法快2至3个数量级；在UEA数据集上，MS-Index相较MASS的优势则缩小至5至7倍。这是因为UEA数据集捕捉的是短暂的单个事件（例如，一个人抬起手臂或一只鸭子发出声音），而非包含多个事件的连续长序列。此类数据集中的子序列通常比股票或天气数据集中的子序列彼此更相似。这使得基于索引的方法面临更具挑战性的查询，第5.2.6节将进一步讨论这一点。另一个值得注意的现象是，即使面对极长的时间序列，MS-Index相较竞争方法仍保持显著优势：在风数据集上，其速度比MASS快22倍，比其他方法快 ~100倍。这一点意义重大，因为物联网和传感器数据等领域中，此类数据集正变得越来越常见。 [51] 总而言之，我们得出结论：面对不同特征、来自不同领域的各类数据集，MS-Index的速度都显著快于所有竞争方法，这证实了其鲁棒性和普遍适用性。

5.2.6 查询难度的影响 基于剪枝的算法依赖于“左尾”假设 [3]: 只有数据集中一小部分与查询相似。它们的性能取决于查询与索引数据之间的相对对比度，即最近和最远

[10]归一化子序列的结果未列出，因为它们呈现出相似的模式。

表6：在 DuckDuckGeese 数据集（UEA）上，MS-Index 对不同查询通道数的查询的相对对比度及剪枝节点百分比。

查询通道 $\lvert c_Q \rvert$		16	64	256	1024
相对对比度	Raw	62	107	110	120
	归一化	14.5	14.4	13.9	13.9
剪枝百分比	Raw	70.54%	79.36%	92.20%	97.94%
	归一化	89.58%	84.66%	83.68%	83.61%

索引实体的距离之比 [1, 17]。遵循 [4, 78],，我们通过改变噪声级别并使用来自留出数据的分布外（OOD）查询，测试算法对不同情况的敏感性。图 8a 展示了股票和天气数据集的结果，表明查询时间与相对对比度成反比（次坐标轴上的灰色条），这与先前的研究结果一致 [4, 17, 78]。归一化子序列查询的对比度低于原始查询，因此更具挑战性。对于高噪声或分布外查询，MS-Index 的剪枝能力会减弱，性能降至与 MASS 相当（另加树遍历开销）；这表明顺序扫描更为合适，因为它不依赖剪枝。这促使未来研究探索一种混合方法，根据估计的查询难度在 MS-Index 和 MASS 之间进行选择。

5.2.7 查询通道数量我们使用 UEA 的*DuckDuckGeese* 数据集研究算法对查询通道数量的敏感性，该数据集包含 1024 个通道。图8b 显示，随着通道数量增加，MS-Index 的查询时间呈次线性增长，而其他方法则呈线性增长。这一性能提升源于剪枝能力的增强，其效果超过了额外距离计算带来的开销。例如，查询时间在通道数从 8 增至 16 时由 8.4 毫秒降至 4.4 毫秒，之后随着边际选择性减弱而缓慢增长。表6证实，在查询原始子序列时，随着通道数从 16 增至 1024，MS-Index 能剪去 70.5%–97.9% 的 R 树节点，表明其能有效利用多变量

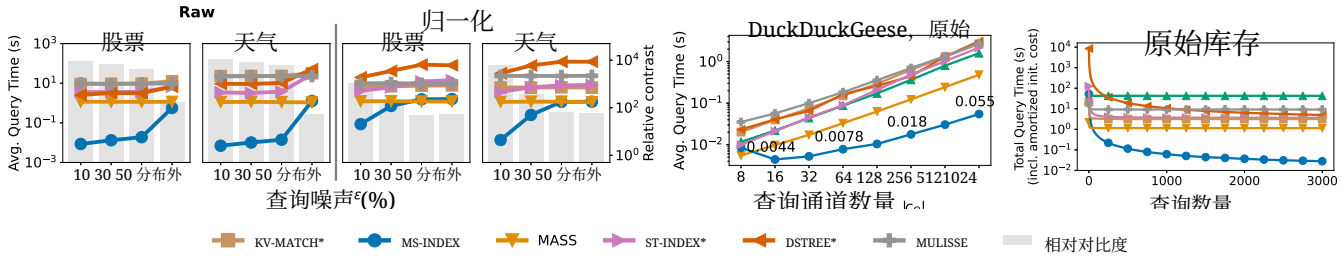

图8：查询工作负载对算法的影响。(a) 不同查询噪声水平下算法的查询时间；(b) 不同查询通道数下的查询时间；(c) 查询的摊销初始化时间。所有 y 轴均采用对数刻度。

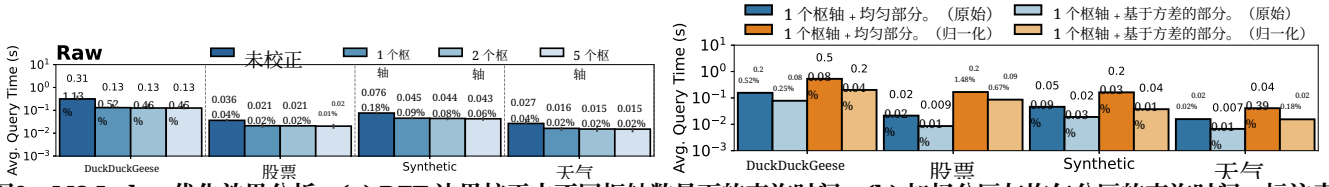

图9：MS-Index 优化效果分析：(a) DFT 边界校正中不同枢轴数量下的查询时间；(b) 加权分区与均匀分区的查询时间。标注表示平均查询时间以及被穷尽检查的子序列所占百分比（其余子序列均被剪枝）。

信息。[11] 相比之下，DSTree*、ST-Index* 和 KV-Match* 的表现会随着查询通道增多而变差，因为它们逐通道结果的并集几乎呈线性增长。对于归一化子序列，MS-Index 的性能仍然更优，但由于维数灾难，其剪枝能力有所下降——随着通道数量增加，点之间的距离变得更加均匀，判别能力随之降低 [10, 46]。对于原始子序列，这种影响较不明显，因为通道内数值的尺度仍可提供额外的判别信息，有助于剪枝。表6中的相对对比度值证实了这一点：对于原始子序列，相对对比度值随着 $|c_Q|$ 增加；而对于归一化子序列，相对对比度值则略有下降。

5.2.8 摊销初始化成本由于各算法的初始化成本不同，其相对效率取决于执行的查询次数。图8c展示了最多执行 3000 次查询时的摊销总成本（初始化时间 + 查询时间）。虽然 MASS 凭借较低的初始化成本，在查询次数较少时速度最快，但仅执行 45 次查询后，MS-Index 就变得更加高效；随着查询次数增加，由于其查询性能更优，这一优势也会进一步扩大。

*5.2.9 DFT*界限校正优化的影响 回顾第 3.4 节，我们使用基于三角不等式的校正项收紧距离界限，从而以增加初始化开销为代价实现更多剪枝。我们评估了这一校正对不同枢轴数量下初始化时间和查询时间的影响。图9a显示，仅添加一个枢轴就能使查询性能提升 2 倍，而增加更多枢轴带来的改进则微乎其微。这种现象称为覆盖饱和效应：随着枢轴增多、空间得到越来越充分的覆盖，基于枢轴的界定效果会趋于稳定 [6, 19]。在我们的案例中，第一个枢轴之后的收益递减表明余项空间（即高频 DFT 系数）复杂度较低：数据集中在一个点附近。这一观察并不令人意外，因为根据定义，这些余项由能量较低的频率组成，而这些频率大多代表符合正态分布的噪声 [39]。尽管如此，与不进行校正相比，性能的提升表明余项仍包含有价值的信息。我们的分析显示，进行枢轴比较会使初始化时间分别增加 15%、30% 和 75%（对应 1、2 和 5 个枢轴）。考虑到查询时间的改善，单个枢轴提供了最佳的成本效益权衡。

5.2.10 收紧最小边界矩形的优化效果 我们将通过加权 R 树分区（第3.4节）收紧最小边界矩形的优化效果与均匀分区进行了比较。图9b中的结果表明，加权分区使 R 树能够更好地适应数据集的特征，从而将查询时间缩短至原来的 1/1.5 至 1/3；在 DuckDuckGeese 和合成数据等高维数据集上，效果尤为显著。由此可见，加权分区策略是有益的。

6 结论

我们研究了多变量时间序列上的定长子序列搜索问题，并提出了 MS-Index，这是一种可自适应地对基于离散傅里叶变换的特征空间进行分区和索引的精确算法，能够以较低开销剪枝 99% 以上的子序列。基于 34 个数据集的评估表明，对于原始和归一化子序列，MS-Index 的性能均比最先进的方法高出两个数量级；此外，其性能随查询通道数量的增加而呈次线性扩展。

致谢

本研究获得了地平线欧洲研究与创新计划STELAR（101070122）、AI4Europe（10107- 0000）、TwinODIS（101160009）、ARMADA（101168951）、DataGEMS（101188416）、RECITALS（101168490）的资助，并获得 ΥΠΑΙΘΑ & NextGen- erationEU项目 HARSH（ΥΠ3ΤΑ – 0560901）的资助。

[11]节点的剪枝百分比并不等同于剪枝能力（即所有子序列中被考虑的子序列所占的比例；对于 MS-Index，该比例约为 99.9%），因为树中的许多节点都包含成组的子序列。

参考文献

[1] Charu C. Aggarwal、Alexander Hinneburg和Daniel A. Keim。2001年。高维空间中距离度量的惊人行为。载于《数据库理论——ICDT 2001》，Jan Van den Bussche和Victor Vianu（编）。420–434。[2] George Arfken。1985年。《物理学家的数学方法》（第三版）。Academic Press, Inc.，圣迭戈。[3] Martin Aumüller和Matteo Ceccarello。2021年。局部维度度量在最近邻搜索基准测试中的作用。《信息系统》101（ 2021 101807 10 1016 2021 101807 ），。https://doi.org/ 。/j.is. 。[4] Ilias Azizi、Karima Echihabi和Themis Palpanas。2023年。ELPIS：面向可扩展数据科学的基于图的相似性搜索。《VLDB基金会会刊》16，6（2023年2月），1548 1559 10 14778 3583140 3583166 – 。https://doi.org/ 。/ 。Martin [5] Bach-Andersen、Bo Rømer-Odgaard和Ole Winther。2017年。面向大规模风力涡轮机诊断的灵活非线性预测模型。《风能》20（2017），753–764。[6] Ricardo Baeza-Yates、Walter Cunto、Udi Manber和Sun Wu。1994年。使用固定查询树进行邻近匹配。载于《组合模式匹配》，Maxime Crochemore和Dan Gusfield（编）。198–212。[7] Anthony Bagnall、Hoang Anh Dau、Jason Lines、Michael Flynn、James Large、Aaron Bostrom、Paul Southam和Eamonn Keogh。2018年。UEA多变量时间序列分类档案，2018年。arXiv:1811.00075 https://arxiv.org/abs/ 1811.00075[8] Anthony J. Bagnall、Richard L. Cole、Themis Palpanas和Konstantinos Zoumpatianos。2019年。数据序列管理（Dagstuhl研讨会19282）。《Dagstuhl报告》9，7（2019），24–39。[9] A. J. Bagnall和G. J. Janacek。2004年。利用截断数据对ARMA模型中的时间序列进行聚类。载于《第十届ACM SIGKDD知识发现与数据挖掘国际会议论文集》（KDD '04）。49–58。https://doi.org/ 10 1145 1014052 1014061 。/ 。Norbert Beckmann、[10] Hans-Peter Kriegel、Ralf Schneider和Bernhard Seeger。1990年。R*-树：一种用于点和矩形的高效且稳健的访问方法。《SIGMOD记录》19, 2 1990 322 331 10 1145 93605 98741 （5月），– 。[11] 1975 https://doi.org/ 。/ 。Jon Louis Bentley。 。用于关联搜索的多维二叉搜索树。《ACM通讯》18，9（1975年9月），509–517。[12] Kanishka Bhaduri、Qiang Zhu、Nikunj C. Oza和Ashok N. Srivastava。2010年。快速且灵活的多变量时间序列子序列搜索。载于《2010年IEEE国际数据挖掘会议》。48–57。https://doi.org/10.1109/ICDM。2010.36[13] Angela Bonifati、Francesco Del Buono、Francesco Guerra、Miki Lombardi和Donato Tiano。2023年。利用Time2Feat进行多变量时间序列的可解释聚类。《VLDB基金会会刊》16，12（2023），3994–3997。[14] Markus M. Breunig、Hans-Peter Kriegel、Raymond T. Ng和Jörg Sander。2000年。LOF：识别基于密度的局部离群点。载于《2000年ACM SIGMOD国际数据管理会议论文集》（SIGMOD '00）。93–104。https://doi.org/ 10 1145 342009 335388 。/ 。John S Bridle。 。前馈分类网络输出的概率解[15] 1990 释及其与统计模式识别的关系。载于《神经计算：算法、架构与应用》。Springer，227–236。[16] Alessandro Camerra、Themis Palpanas、Jin Shieh和Eamonn Keogh。2010年。iSAX 2.0：十亿条时间序列索引与挖掘。载于《2010年IEEE国际数据挖掘会议》。58 67 10 1109 2010 124 – 。[17] https://doi.org/ 。/ICDM。 。Matteo Ceccarello、Alexandra Levchenko、Ioana Ileana和Themis Palpanas。2025年。评估和生成高维向量相似性搜索的查询工作负载。载于《第31届ACM SIGKDD知识发现与数据挖掘会议论文集》第2卷（KDD '25）。5299–5310。https: //doi.org/10.1145/3711896.3737383[18] Georgios Chatzigeorgakidis、Dimitrios Skoutas、Kostas Patroumpas、Themis Palpanas、Spiros Athanasiou和Spiros Skiadopoulos。2023年。时间序列中的高效范围与kNN双子序列搜索。《IEEE知识与数据工程汇刊》35，6（2023），5794–5807。[19] Edgar Chávez、Gonzalo Navarro、Ricardo Baeza-Yates和José Luis Marroquín。2001年。在度量空间中进行搜索。《ACM计算综述》33，3（2001年9月），273–321。[20] Paolo Ciaccia、Marco Patella和Pavel Zezula。1997年。M树：度量空间中用于相似性搜索的高效访问方法。载于《第23届国际超大型数据库会议论文集》（VLDB '97）。426–435。[21] Michele Dallachiesa、Themis Palpanas和Ihab F. Ilyas。2014年。不确定数据序列中的Top-k最近邻搜索。《VLDB基金会会刊》8，1（2014年9月），13 24 10 14778 2735461 2735463 – 。[22] https://doi.org/ 。/ 。Jens E. d'Hondt、Haojun Li、Fan Yang、Odysseas Papapetrou和John Paparrizos。2025年。多变量时间序列距离度量的结构化研究。《ACM数据管理会刊》3，3，文章121（2025年6月），29页。https://doi.org/10.1145/ 3725258[23] Rui Ding、Qiang Wang、Yingnong Dang、Qiang Fu、Haidong Zhang和Dongmei Zhang。2015年。YADING：大规模时间序列数据的快速聚类。《VLDB基金会会刊》8, 5 2015 473 484 10 14778 2735479 2735481 （1月），– 。[24] 2024 https://doi.org/ 。/ 。Jens E. d'Hondt、Odysseas Papapetrou和John Paparrizos。 。超越维度：多变量时间序列距离度量的结构化评估。载于《2024年IEEE第40届国际数据工程会议研讨会》（ICDEW）。107 112 10 1109 61823 2024 00020 – 。[25] https://doi.org/ 。/ICDEW 。 。Karima Echihabi、Kostas Zoumpatianos、Themis Palpanas 和 Houda Benbrahim。2018。数据序列相似性搜索中的勒耳那九头蛇：对最先进技术的实验评估。《VLDB基金会会刊》12，2（2018年10月），112–127。https://doi.org/ 10 14778 3282495 3282498 。/ 。Karima [26] Echihabi、Kostas Zoumpatianos、Themis Palpanas 和 Houda Benbrahim。2019。勒耳那九头蛇归来：数据序列近似相似性搜索的实验评估。《VLDB基金会会刊》13，3（2019年11月），403–420。https://doi.org/ 10 14778 3368289 3368303 。/ 。Ronald Fagin、Amnon Lotem 和 [27] Moni Naor。2001。中间件的最优聚合算法。载于《第二十届ACM SIGMOD-SIGACT-SIGART数据库系统原理研讨会论文集》（PODS '01）。102–113。https://doi.org/ 10 1145 375551 375567 。/ 。Christos Faloutsos、[28] M. Ranganathan 和 Yannis Manolopoulos。1994。时间序列数据库中的快速子序列匹配。载于《1994年ACM SIGMOD数据管理国际会议论文集》（SIGMOD '94）。419–429。https://doi.org/ 10 1145 191839 191925 。/ 。Kefeng Feng、[29] Peng Wang、Jiaye Wu 和 Wei Wang。2020。L-Match：一种轻量且有效的子序列匹配方法。《IEEE Access》8（2020），71572–71583。[30] Hakan Ferhatosmanoglu、Ertem Tuncel、Divyakant Agrawal 和 Amr El Abbadi。2000。面向非均匀高维数据集的基于向量近似的索引。载于《第九届信息与知识管理国际会议论文集》（CIKM '00）。202–209。https://doi.org/10.1145/354756.354820[31] Ada Wai-chee Fu、Eamonn Keogh、Leo Yung Hang Lau 和 Chotirat Ann Ratanamahatana。2005。时间序列查询中的尺度变换与时间规整。载于《第31届超大型数据库国际会议论文集》（VLDB '05）。649–660。[32] Roger Grosse、Rajat Raina、Helen Kwong 和 Andrew Y. Ng。2007。用于音频分类的平移不变稀疏编码。载于《第二十三届不确定性人工智能会议论文集》（UAI'07）。149–158。[33] Antonin Guttman。1984。R树：用于空间搜索的动态索引结构。载于《1984年AC M SIGMOD数据管理国际会议论文集》（SIGMOD ' *84* 47 57 10 1145 602259 602266 ）。–。[34] https://doi.org/ 。/ 。Jon Hills、Jason Lines、Edgaras Baranauskas、James Mapp 和 Anthony Bagnall。2014。通过形状片段变换进行时间序列分类。《数据挖掘与知识发现》28，4（2014年7月），851–881。[35] Gísli R. Hjaltason 和 Hanan Samet。1999。空间数据库中的距离浏览。《ACM数据库系统汇刊》24，2（1999年6月），265–318。[36] Pablo Huijse、Pablo A. Estevez、Pavlos Protopapas、Jose C. Principe 和 Pablo Zegers。2014。大规模天文时间序列数据库中的计算智能挑战与应用。《IEEE计算智能杂志》9，3（2014年8月），27–39。[37] Konstantinos Kalpakis、Dhiral Gada 和 Vasundhara Puttagunta。2001。用于有效聚类ARIMA时间序列的距离度量。载于《2001年IEEE国际数据挖掘会议论文集》（ICDM '01）。273–280。[38] K. Kashino、G. Smith 和 H. Murase。1999。用于快速检索音频和视频的时间序列主动搜索。载于1999年IEEE声学、语音与信号处理国际会议。论文集，第6卷。2993–2996，第6卷。https://doi.org/ 10 1109 1999 757470 。/ [39] 1993 ICASSP。 。Steven M Kay。 。统计信号处理基础：估计理论。Prentice-Hall, Inc. [40] Eamonn Keogh、Themistoklis Palpanas、Victor B. Zordan、Dimitrios Gunopulos 和 Marc Cardle。2004。人体运动大型数据库的索引。载于《第三十届超大型数据库国际会议论文集——第30卷》（VLDB '04）。780–791。[41] S. Knieling、J. Niediek、E. Kutter、J. Bostroem、C.E. Elger 和 F. Mormann。2017。用于选择性神经元反应的在线自适应筛查程序。《神经科学方法杂志》291（2017），36–42。[42] Haridimos Kondylakis、Niv Dayan、Kostas Zoumpatianos 和 Themis Palpanas。2018。Coconut：一种用于构建数据序列索引的可扩展自底向上方法。《VLDB基金会会刊》11，6（2018年2月），677–690。https://doi.org/10.14778/3199517.3199519 [43] S.T. Leutenegger、M.A. Lopez 和 J. Edgington。1997。STR：一种简单高效的R树打包算法。载于《第13届数据工程国际会议论文集》。497 506 10 1109 1997 582015 – 。[44] https://doi.org/ 。/ICDE。 。Stefan Liess、Saurabh Agrawal、Snigdhansu Chatterjee 和 Vipin Kumar。2017。西西伯利亚平原与ENSO区域之间的遥相关。《气候杂志》30，1（2017），301–315。[45] Michele Linardi 和 Themis Palpanas。2018。数据序列中的可扩展可变长度相似性搜索：ULISSE方法。《VLDB基金会会刊》11，13（2018年9月），2236 2248 10 14778 3275366 3284968 – 。[46] 1979 https://doi.org/ 。/ 。R. B. Marimont 和 M. B. Shapiro。 。最近邻搜索与维数灾难。《IMA应用数学杂志》24，1（1979年8月），59–70。

[47] Yang-Sae Moon、Kyu-Young Whang 和 Wook-Shin Han。2002。General Match：一种基于广义窗口的时间序列数据库子序列匹配方法。载于*2002*年*ACM SIGMOD*数据管理国际会议论文集（*SIGMOD '02*）。382–393。https://doi.org/10.1145/564691.564735[48] Yang-Sae Moon、Kyu-Young Whang 和 Wook-Shin Han。2002。General Match：一种基于广义窗口的时间序列数据库子序列匹配方法。载于*SIGMOD'02*论文集。382–393。[49] Abdullah Mueen、Suman Nath 和 Jie Liu。2010。大规模时间序列数据的快速近似相关性计算。载于*2010*年*ACM SIGMOD*数据管理国际会议论文集（*SIGMOD '10*）。171–182。https://doi.org/10.1145/1807167.1807188[50] Abdullah Mueen、Sheng Zhing、Yan Zhu、Michael Yeh、Kaveh Kamgar、Krishnamurthy Viswanathan、Chetan Gupta 和 Eamonn Keogh。2022。基于欧几里得距离的时间序列子序列最快相似性搜索算法。http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html。[51] Carlos Enrique Muniz-Cuza、Søren Kejser Jensen、Jonas Brusokas、Nguyen Ho 和 Torben Bach Pedersen。2024。评估误差有界有损压缩对时间序列预测的影响。载于数据库技术进展 - *EDBT*（数据库技术进展 - *EDBT*）。650–663。https://doi.org/10.48786/edbt.2024.56[52] 美国国家海洋和大气管理局。 [无日期]。NOAA综合地表数据集。https://www.ncei.noaa.gov/access/search/dataset-search。[53] Themis Palpanas 和 Volker Beckmann。2019。第一届和第二届跨学科时间序列分析研讨会（ITISA）报告。*SIGMOD*记录48，3（2019），36–40。[54] John Paparrizos 和 Michael J. Franklin。2019。GRAIL：高效的时间序列表示学习。*VLDB'19*论文集（2019），1762–1777。[55] John Paparrizos 和 Luis Gravano。2016。k-Shape：高效且准确的时间序列聚类。*SIGMOD*记录45，1（2016年6月），69–76。https://doi.org/ 10 1145 2949741 2949758 。/ 。John Paparrizos、[56] Chunwei Liu、Aaron Elmore 和 Michael J. Franklin。2023。时间序列数据查询：距离度量的全面比较。*IEEE*数据工程公报（*DEB 2023*）47（2023），69–88。[57] Pavlos Paraskevopoulos、Thanh-Cong Dinh、Zolzaya Dashdorj、Themis Palpanas、Luciano Serafini 等。2013。从移动电话数据中识别和刻画人类行为模式。*D4D*挑战赛专题环节，*NetMob*（2013）。[58] Balázs Pelok 和 Jens E. d'Hondt。2025。MULISSE：多变量时间序列的可变长度相似性搜索。载于*2025*年*IEEE*第*41*届数据工程国际会议研讨会（*ICDEW*）。154–163。https://doi.org/10.1109/ICDEW67478.2025.00026[59] Davood Rafiei 和 Alberto O. Mendelzon。2000。相似时间序列的高效检索。75–89。https://doi.org/10.1007/978-1-4615-1379-7_6[60] Thanawin Rakthanmanon、Bilson Campana、Abdullah Mueen、Gustavo Batista、Brandon Westover、Qiang Zhu、Jesin Zakaria 和 Eamonn Keogh。2012。基于动态时间规整搜索和挖掘数万亿个时间序列子序列。载于第*18*届*ACM SIGKDD*知识发现与数据挖掘国际会议论文集（*KDD '12*）。262–270。https://doi.org/10.1145/2339530.2339576[61] Chotirat Ann Ratanamahatana 和 Eamonn Keogh。2005。关于动态时间规整数据挖掘的三大误区。506–510。https://doi.org/10.1137/1.9781611972757.50[62] John F. Roddick 和 Kathleen Stewart Hornsby。2001。时间、空间及时空数据挖掘。载于计算机科学讲义。[63] Alejandro Pasos Ruiz、Michael Flynn、James Large、Matthew Middlehurst 和 Anthony Bagnall。2021。多变量时间序列分类大赛关：近期算法进展的综述与实验评估。数据挖掘与知识发现 35, 2（2021年3月1日），401–449。[64] A Salarpour 和 H Khotanlou。2018年。多变量时间序列聚类距离度量的实证比较。国际工程学杂志 31, 2（2018年），250–262。[65] Hanan Samet。2005年。多维与度量数据结构基础（*Morgan Kaufmann*计算机图形学与几何建模系列）。Morgan Kaufmann出版社，旧金山，加利福尼亚州，美国。[66] Jin Shieh 和 Eamonn Keogh。2008年。iSAX：索引和挖掘太字节级时间序列。载于第*14*届*ACM SIGKDD*知识发现与数据挖掘国际会议论文集（*KDD '08*）。623–631。https://doi.org/10. 1145/1401890.1401966[67] Ahmed Shifaz、Charlotte Pelletier、François Petitjean 和 Geoffrey I. Webb。2023年。多变量时间序列的弹性相似性与距离度量。知识与信息系统 65, 6（2023年6月1日），2665–2698。[68] Mohammad Shokoohi-Yekta、Bing Hu、Hongxia Jin、Jun Wang 和 Eamonn Keogh。2017年。将动态时间规整推广到多维情形需要一种自适应方法。数据挖掘与知识发现 31, 1（2017年1月1日），1 31 10 1007 10618 016 0455 0 – 。https://doi.org/ 。[69] /s - - - Gilbert Strang。1994。小波。《美国科学家》82, 3（1994年），250–255。[70] Shreshth Tuli、Giuliano Casale 和 Nicholas R. Jennings。2022年。TranAD：用于多变量时间序列数据异常检测的深度Transformer网络。《*VLDB*基金会会刊》15, 6（2022年2月），1201–1214。https://doi.org/10.14778/3514061. 3514067[71] Michail Vlachos、Marios Hadjieleftheriou、Dimitrios Gunopulos 和 Eamonn Keogh。2003年。支持多种距离度量的多维时间序列索引。载于第九届*ACM SIGKDD*知识发现与数据挖掘国际会议论文集（*KDD '03*）。216–225。https://doi.org/ 10 1145 956750 956777 。/ 。Yang Wang、Peng Wang、Jian Pei、[72] Wei Wang 和 Sheng Huang。2013年。一种用于时间序列全序列匹配的数据自适应动态分段索引。《*VLDB*基金会会刊》6, 10（2013年8月），793–804。https://doi.org/10.14778/2536206. 2536208[73] Jiaye Wu、Peng Wang、Ningting Pan、Chen Wang、Wei Wang 和 Jianmin Wang。2019年。KV-Match：支持归一化与时间规整的子序列匹配方法。载于*2019*年*IEEE*第*35*届数据工程国际会议（*ICDE*）。866–877。https://doi.org/10.1109/ICDE.2019.00082[74] Xiaopeng Xi、Eamonn Keogh、Christian Shelton、Li Wei 和 Chotirat Ann Ratanamahatana。2006年。使用数值数量缩减进行快速时间序列分类。载于第*23*届国际机器学习会议论文集（*ICML '06* 1033 1040 10 1145 1143844 1143974 ）。– 。https://doi.org/ 。[75] / 。Fan Yang 和 John Paparrizos。2025。SPARTAN：数据自适应符号时间序列近似。《*ACM*数据管理会刊》3, 3，文章220（2025年6月），30 10 1145 3725357 页。https://doi.org/ 。/ Kiyoung Yang 和 Cyrus Shahabi。[76] 2004。一种用于多变量时间序列的基于主成分分析的相似性度量。载于第*2*届*ACM*多媒体数据库国际研讨会论文集（*MMDB '04*）。65–74。https://doi.org/10.1145/1032604. 1032616[77] Haohan Zhu、George Kollios 和 Vassilis Athitsos。2012年。一种高效且有效的子序列检索通用框架。《*VLDB*基金会会刊》5, 11（2012年7月），1579–1590。https://doi.org/10.14778/2350229.2350271 [78] Kostas Zoumpatianos、Yin Lou、Ioana Ileana、Themis Palpanas 和 Johannes Gehrke。2018年。生成数据序列查询工作负载。《*VLDB*期刊》27, 6（2018年12月1日），823–846。https://doi.org/10.1007/s00778-018-0513-x