Supplementary Material: PISD: A Linear Complexity Distance beats Dynamic Time Warping on Time Series Classification and Clustering

Minh-Tuan Tran[a,1,*], Xuan-May Le[a,1], Van-Nam Huynh[b], Sung-Eui Yoon[c]

[a] *Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam*
[b] *School of Knowledge Science, Japan Advanced Institute of Science & Technology (JAIST), Ishikawa, Japan*
[c] *School of Computing, Korea Advanced Institute of Science & Technology (KAIST), Daejeon, South Korea*

1. Pseudocode

1.1. Pseudocode of PIP

Given time series $Q = q_1, q_2, ..., q_n$ and k is the number of important points to extract. The pseudocode of PIPs extractor is shown in **Algorithm 1**.

We use the Perpendicular Distance (PD) version for computing the distance between the line and the point. Perpendicular Distance of one position pos and PIP is calculated by Eq. 1.

$$PD(pos, PIPs) = \frac{a * P[pos] - Q[pos] + c}{\sqrt{a^2 + 1}} \tag{1}$$

Given g where $1 \leq g \leq k$ and $PIPs[g] < pos < PIPs[g+1]$, assign $s = PIPs[g]$ and $e = PIPs[g+1]$. a and c is calculated by Eq. 2 and P is list of position with z normalization, $P = z\_norm([1, .., n])$.

[*] Corresponding author.
Email address: tranminhtuan@tdtu.edu.vn (Minh-Tuan Tran)
[1] These authors contributed equally to this work.

$$a = \frac{Q[e] - Q[s]}{P[e] - P[s]}; \quad c = Q[e] - a * P[e] \tag{2}$$

Algorithm 1 PIPExtractor

Input: time series: Q, number of important point: k
Output: PIP set of Q: $QPIPs$
$QPIPs = [1, n]$
for i = 1 to k - 2 **do**
    Find z from 1 to n with max $PD(Q[z], QPIPs)$ where pos not in QPIPs
    $QPIPs$.append(pos)
    $QPIPs$.sort()

1.2. Pseudocode for PISD

Algorithm 2 details the pseudocode of PISD. This method consists of the following steps. Firstly, it detects k important points for Q and C (line 1-2) (**Algorithm 1**). After that, extracting PIS (**Algorrithm 3**) and PCS (**Algorrithm 4**) for Q and C respectively (line 3-4 and line 5-6). Then, calculate SubDist of each PIS of Q and its PCS (line 7) and do the same thing for each PIS of C and its PCS (line 8) by **Algorithm 5**. Finally, the PISD is calculated as in line 9. Note that, in our source code, the PIPExtractor and PISExtractor were done before calculating the distance.

Algorithm 2 PISD

Input: time series: Q and C, window size: w, and number of PIPs: k
Output: PISD of Q and C: PISD
1: QPIPs = PIPExtractor(Q, k)
2: CPIPs = PIPExtractor(C, k)
3: QPISs = PISExtractor(Q, QPIPs)
4: CPISs = PISExtractor(C, CPIPs)
5: CPCSs = PCSExtractor(C, w, QPIPs)
6: QPCSs = PCSExtractor(Q, w, CPIPs)
7: QsumD, QsumL = FindDist(QPISs, CPCSs)
8: CsumD, CsumL = FindDist(CPISs, QPCSs)
9: PISD = (QsumD/QsumL) + (CsumD/CsumL)

Algorithm 3 PISExtractor

Input: time series: Q, PIP set of Q: QPIPs
Output: PIS set of Q: QPISs

QPISs = []
for i = 1 to len(QPIPs) - 2 **do**
    PIS = Q[QPIP_i : QPIP_{i+2}]
    QPISs.append(PIS)

Algorithm 4 PCSExtractor

Input: time series: C, window size: w, PIP set of Q: QPIPs
Output: PCS set of Q in C: QPCSs

QPCSs = []
for i = 1 to len(QPIPs) - 2 **do**
    startpos = (QPIP_i - w + 1 < 1) ? 1 : QPIP_i - w + 1
    endpos = (QPIP_{i+2} + w > n) ? n : QPIP_{i+2} + w
    PCS = C[startpos : endpos]
    QPCSs.append(PCS)

Algorithm 5 FindDist

Input: list of Important subsequences: PISs, list of corresponding subsequences: PCSs
Output: Sum of distance: sumD and sum of length: sumL

sumD = sumL = 0
for i = 1 to len(PISs) **do**
    sumD += CID-ED-SubDist(PISs_i, PCSs_i) * len(PISs_i)
    sumL += len(PISs_i)

1.3. Pseudocode for speed up PISD

Algorithm 6 details the pseudocode of Speed Up PISD. Firstly, we detect k important points for each Q and C (line 1-2) (**Algorithm 1**). After that, calculate the list of complexity invariants for Q and C (line 3-4). Then, generate the Position-aware Distance Matrix and Position Lookup Table for each Q and C by **Algorithm 7** and **Algorithm 8** respectively (line 5-6). Line 7-8 indicate the process to calculate the distance for each Q and C by **Algorithm 9**. Finally, the PISD is calculated as in line 9. Note that, in our source code, the PIPExtractor and CIList calculation was done before measuring the distance.

Algorithm 6 SpeedUpPISD

Input: Two time series: Q and C, window size: w, and number of PIPs: k
Output: PISD of Q and C: PISD

```
QPIPs = PIPExtractor(Q, k)
CPIPs = PIPExtractor(C, k)
QCIList = (Q[2:] - Q[:-2])**2
CCIList = (C[2:] - C[:-2])**2
QPDM, CPDM = PDMGenerator(Q,C,w)
PLStart, PLEnd = PLGenerator(len(Q),w)
QsumD, QsumL = SpeedUpFindDist(QPIPs, PLStart, PLEnd, QPDM, QCIList,CCIList)
CsumD, CsumL = SpeedUpFindDist(CPIPs, PLStart, PLEnd, CPDM, CCIList,QCIList)
PISD = (QsumD/QsumL) + (CsumD/CsumL)
```

Algorithm 7 PDMGenerator

Input: two time series: Q and C, window size: w
Output: PDM of Q and C: QPDM and CPDM

```
QPDM = [[inf for i = 1 to (2w+1)] for j = 1 to n]
CPDM = [[inf for i = 1 to (2w+1)] for j = 1 to n]
QPDM[:,w] = CPDM[:,w] = (Q - C)**2
for i = 1 to w do
    QPDM[i:,w-i] = CPDM[:-i,w+i] = (Q[i:] - C[:-i])**2
    QPDM[:-i,w+i] = CPDM[i:,w-i] = (Q[:-i] - C[i:])**2
```

Algorithm 8 PLGenerator

Input: Length of time series: n and window size: w
Output: Position lookup table PLStart and PLEnd

```
PLStart = [2 for i = 1 to n]
PLEnd = [(2w+1) for i = 1 to n]
for i = 1 to w - 1 do
    PLStart[i] += w - i
for i = 1 to w do
    PLEnd[i] -= w - i + 1
```

Algorithm 9 SpeedUpFindDist

Input: PIPs, PLStart, PLEnd, PDM, CI list of target time series TCIList and corresponding time series CCIList
Output: Sum of distance: sumD and sum of length sumL

```
sumD = [], sumL = []
for i = 1 to k - 2 do
    s = PIPs[i], e = PIPs[i+2]
    pcs-s = (s - w + 1) < 1 ? 1 : (s - w + 1)
    pcs-e = (e + w) > n ? n : (e + w)
    Sub-PDM = PDM[s:e,PLStart[s]:PLEnd[e]]
    PIS-CI = CIList[s:e]
    PCS-CI = CCIList[pcs-s:pcs-e]
    QsumD += Sub-PDM-Distance(Sub-PDM, PIS-CI, PCS-CI)
    QsumL += len(QPIS_i)
```

Algorithm 10 Sub-PDM-Distance

Input: a target matrix: Sub-PDM, CI of PIS: PISCI, and CIList of PCS: PCSCIList
Output: Distance of PIS and PCS: SPDMD

```
DistList = list of sum of each column in Sub-PDM.
DistList = list of square root of each column in DistList.
CIList = []
PISCIV = sqrt(sum(PISCI))
for i = 1 to (len(PCSCI) - len(PISCI) + 1) do
    PCSCI = PCSCIList[i:len(PISCI)+i-1]
    PCSCIV = sqrt(sum(PCSCI))
    CIList.append(max(PISCIV,PCSCIV)/min(PISCIV,PCSCIV))
DistList = DistList * CIList
SPDMD = min(DistList)
```

1.4. Pseudocode for PISA

Given a set of time series D = $Q^1, Q^2, ..Q^m$, the centroid time series Z = $z_1, z_2, ..., z_n$, a window size w, and the number of perceptually important points k. The averaging time series of D is calculated by **Algorithm 11** which consists following steps:

Algorithm 11 PISA

Input: A set of time series: D, a centroid: Z, window size: w, number of pips: k

Output: Averaging time series: $avgTS$

```
01: ZPIPs = PIPExtractor(Z, k)
02: ZPISs = PISExtractor(Z, ZPIPs)
03: SPISList = [[] for i = 1 to k - 1]
04: SumW = [0 for i = 1 to k - 1]
05: for Q in D:
06:    ZPCSs = PCSExtractor(Q, w, ZPIPs)
07:    for i = 1 to k - 1 do
08:       MinDist, BestFit = CID-ED-SubDist(ZPISs[i], ZPCSs[i])
09:       Weight = 1/MinDist
10:       SPIS1 = BestFit[:ZPIPs[i+1]-ZPIPs[i]]*Weight
11:       SPIS2 = BestFit[ZPIPs[i+1]-ZPIPs[i]:]*Weight
12:       SPISList[i].append(SPIS1)
13:       SPISList[i+1].append(SPIS2)
14:       sumW[i] += Weight
15:       sumW[i+1] += Weight
16: avgTS = []
17: for i = 1 to k - 1 do
18:    avgSPIS = sum(SPISList[i])/sumW[i]
19:    avgTS.concatenate(avgSPIS)
```

- Firstly, find k PIPs and k - 2 PISs for time series Z (line 1-2). For each Q in D, extract PCSs of Z in Q (line 6). Then, calculate the min distance and best fit time series of each PCS and PIS (line 8). Note that, the SubDist function is calculated by Eq. **??**, but also returns the best fit time series $C_{i,i+n}$.
- Secondly, divide the BestFit into two SPIS (subsequence constructed by two continuous PIPs) and multiply them with a weight (line 10-11)

which is computed by $1/MinDist$ (line 9). The SPIS and Weight are stored on SPISList and sumW respectively (line 12-15).

- Finally, the averaging time series is generated by concatenating an averaging of each SPIS in SPISList (line 16-19).