Efficient Matrix Profile Computation Using Different Distance Functions
Reza Akbarinia, Bertrand Cloez

To cite this version:
Reza Akbarinia, Bertrand Cloez. Efficient Matrix Profile Computation Using Different Distance Functions. 2020. hal-02788459

HAL Id: hal-02788459
https://hal.inrae.fr/hal-02788459
Preprint submitted on 5 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives| 4.0 International License
Efficient Matrix Profile Computation Using Different Distance Functions

Reza Akbarinia
INRIA & LIRMM, Univ. Montpellier, France
reza.akbarinia@inria.fr

Bertrand Cloez
INRA, SupAgro, UMR MISTEA, Univ. Montpellier,
Montpellier, France
bertrand.cloez@inra.fr

ABSTRACT
Matrix profile has been recently proposed as a promising technique to the problem of all-pairs-similarity search on time series. Efficient algorithms have been proposed for computing it, e.g., STAMP [13], STOMP [15] and SCRIMP++ [10]. All these algorithms use the z-normalized Euclidean distance to measure the distance between subsequences. However, as we observed, for some datasets other Euclidean measurements are more useful for knowledge discovery from time series.

In this paper, we propose efficient algorithms for computing matrix profile for a general class of Euclidean distances. We first propose a simple but efficient algorithm called AAMP for computing matrix profile with the pure (non-normalized) Euclidean distance. Then, we extend our algorithm for the p-norm distance. We also propose an algorithm, called ACAMP, that uses the same principle as AAMP, but for the case of z-normalized Euclidean distance. We implemented our algorithms, and evaluated their performance through experimentation. The experiments show excellent performance results. For example, they show that AAMP is very efficient for computing matrix profile for non-normalized Euclidean distances. The results also show that the ACAMP algorithm is significantly faster than SCRIMP++ (the state of the art matrix profile) for the case of z-normalized Euclidean distance.

ACM Reference Format:

1 INTRODUCTION
Matrix profile has been recently proposed as an efficient technique to the problem of all-pairs-similarity search in time series [3, 6, 11, 12, 14]. Given a time series T and a subsequence length m, the matrix profile returns for each subsequence included in T its distance to the most similar subsequence in the time series. The matrix profile is itself a time series very useful for data analysis, e.g., detecting the motifs (represented by low values), discords (represented by high values), etc.

One naive solution for computing a matrix profile is the nested loop algorithm that for each subsequence computes its distance to any other subsequence using the distance function. However, this solution is not efficient and can take too much time for relatively big databases. Recently, efficient algorithms have been proposed for matrix profile computation.

e.g., STAMP [13], STOMP [15] and SCRIMP++ [10]. All these algorithms use the z-normalized Euclidean distance to measure the distance between subsequences.

However, we observed that for some datasets, other distances such as pure (non-normalized) Euclidean distance are more useful for knowledge discovery. For example, in the case of time series containing long subsequences of the same values (that show some type of stability in the activities), the z-normalized distance cannot be used because the standard deviation of these subsequences is zero, thus the z-normalized distance becomes infinite. In addition, in some cases the normalization can remove some rare information from the matrix profile. As an example, consider Figure 1.a that shows a time series T, and Figures 1.b and 1.c that depict the matrix profiles generated from T using z-normalized and pure Euclidean distances respectively. As seen, the matrix profile that uses z-normalized distance loses the information about the anomaly around 500 in the time series. But, the pure Euclidean distance highlights it.

We believe that for knowledge extraction from different datasets, we need to give to the users the possibility of computing matrix profiles with different similarity distances. In this paper, we propose matrix profile algorithms for different types of Euclidean distance. Our contributions are as follows.

• We first propose a simple but efficient algorithm called AAMP for computing matrix profile with the pure Euclidean distance. AAMP is executed in a set of iterations, such that in each iteration the distance of subsequences is computed incrementally. The time complexity of AAMP is $O(n \times (n - m))$ with small constants, where $n$ is the time series length and $m$ the subsequence length.

• We extend AAMP to compute matrix profile for the p-norm distance that is more general than Euclidean.

• We propose an extension of AAMP, called ACAMP, that uses the same principle as AAMP but for z-normalized Euclidean distance. In ACAMP, we use an incremental formula for computing z-normalized distance that is based on some variables computed incrementally in a sliding window that moves over the subsequences of the time series.

We precise that these new algorithms are exact, anytime and incrementally maintainable. They take a deterministic execution time that only depends on the time series and subsequence length.

We implemented our algorithms and compared them with the state of the art algorithm on matrix profile, i.e., SCRIMP++
In this paper, we are interested in efficient computation of matrix profile using three different distance measures: 1) Euclidean distance; 2) p-norm distance that is a generalization of Euclidean distance; 3) z-normalized Euclidean distance. These distances are defined as follows.

**Definition 2.5.** The **Euclidean distance** between two subsequences $T_i$ and $T_j$ is defined as:

$$D_{i,j} = \sqrt{\sum_{l=0}^{m-1} (t_{i+l} - t_{j+l})^2}$$  \hspace{1cm} (1)

In this paper, sometimes we call the Euclidean distance as **pure Euclidean distance**.

**Definition 2.6.** Let $p > 1$ be a positive integer, then the **$p$-norm distance** between two subsequences $T_i$ and $T_j$ is defined as:

$$DP_{i,j} = \left(\sum_{l=0}^{m-1} (t_{i+l} - t_{j+l})^p\right)^{\frac{1}{p}}$$  \hspace{1cm} (2)

The z-normalized Euclidean distance is defined as follows.

**Definition 2.7.** Let $\mu_i$ and $\mu_j$ be the mean of the values in two subsequences $T_i$ and $T_j$, respectively. Also, let $\sigma_i$ and $\sigma_j$ be the standard deviation of the values in $T_i$ and $T_j$, respectively. Then, the **z-normalized Euclidean distance** between $T_i$ and $T_j$ is defined as:

$$DZ_{i,j} = \sqrt{\sum_{l=0}^{m-1} \left(\frac{t_{i+l} - \mu_i}{\sigma_i} - \frac{t_{j+l} - \mu_j}{\sigma_j}\right)^2}$$  \hspace{1cm} (3)

### 3 AAMP

In this section, we propose AAMP an efficient algorithm for computing matrix profile using the Euclidean distance. We first present a formula for incremental computation of the Euclidean distance in $O(1)$, and then we detail our AAMP algorithm that uses this formula for computing matrix profile.
3.1 Incremental Computation of Euclidean Distance

Here, we present a formula that allows us to compute the Euclidean distance between two subsequences $T_{i,m}$ and $T_{j,m}$ based on the Euclidean distance of subsequences $T_{i-1,m}$ and $T_{j-1,m}$. The formula is presented by the following lemma.

**Lemma 1.** Let $D_{i,j}$ be the Euclidean distance between two subsequences $T_{i,m}$ and $T_{j,m}$. Let $D_{i-1,j-1}$ be the Euclidean distance between two subsequences $T_{i-1,m}$ and $T_{j-1,m}$. Then $D_{i,j}$ can be computed as:

$$D_{i,j} = \sqrt{D_{i-1,j-1} - (t_{i-1} - t_{j-1})^2 + (t_{i+m-1} - t_{j+m-1})^2} \tag{4}$$

**Proof.** Let $T_{i,m} = (t_1, t_{i+1}, \ldots, t_{i+m-1})$ and $T_{j,m} = (t_1, t_{j+1}, \ldots, t_{j+m-1})$. Then the square of the Euclidean distance between $T_{i,m}$ and $T_{j,m}$ is computed as:

$$D_{i,j}^2 = \sum_{l=0}^{m-1} (t_{i+l} - t_{j+l})^2 \tag{5}$$

And the square of the Euclidean distance between $T_{i-1,m}$ and $T_{j-1,m}$ is:

$$D_{i-1,j-1}^2 = \sum_{l=0}^{m-1} (t_{i-1+l} - t_{j-1+l})^2 \tag{6}$$

By comparing Equations (5) and (6), we have:

$$D_{i,j}^2 = D_{i-1,j-1}^2 - (t_{i-1} - t_{j-1})^2 + (t_{i+m-1} - t_{j+m-1})^2.$$  

Thus, we have:

$$D_{i,j} = \sqrt{D_{i-1,j-1}^2 - (t_{i-1} - t_{j-1})^2 + (t_{i+m-1} - t_{j+m-1})^2}.$$  

By using the above equation, we can compute the Euclidean distance $D_{i,j}$ by using the distance $D_{i-1,j-1}$ in $O(1)$.

3.2 Algorithm

The main idea behind AAMP is that for computing the distance between subsequences it uses diagonal sliding windows, such that in each sliding window, the Euclidean distance is computed only between the subsequences that have a precise difference in their start position. These sliding windows allow us to use Equation (4) for efficient distance computation.

Algorithm 1 shows the pseudo-code of AAMP. Initially, the algorithm sets all values of the matrix profile to infinity (i.e., maximum distance). Then, it performs $n - m - 1$ iterations using a variable $k$ ($1 \leq k \leq n - m - 1$). In each iteration $k$, the algorithm compares each subsequence $T_{i,m}$ with the subsequence that is $k$ positions far from it, i.e., $T_{i,m+k}$. To do this, AAMP firstly computes the Euclidean distance of the first subsequence of the time series, i.e., $T_{1,m}$, with the one that stars at position $k$, i.e., $T_{k,m}$. This first distance computation is done using the normal formula of Euclidean distance, i.e., that of Equation (1). Then, in a sliding window, the algorithm incrementally computes the distance of other subsequences with the subsequences that are $k$ position far from them, and this is done by using Equation (4) in $O(1)$. If the computed distance is smaller than the previous minimum distance that is kept in the matrix profile $P$, then the smaller distance is saved in the matrix profile.

**Example 1.** Figure 2 shows an example of executing AAMP over a time series of length $n = 8$, and for subsequences of length $m = 3$. In this example, the algorithm proceeds in 4 iterations ($n - m - 1 = 4$). In iteration 1, firstly the Euclidean distance between $T_{1,m}$ and $T_{2,m}$ is calculated using the normal Euclidean distance formula. Then the sliding window $sw_1$ moves to the next subsequences, and computes the distance of $T_{2,m}$ and $T_{3,m}$ using Equation (4). Then, the sliding window moves to the next subsequences and computes their distances, i.e., $T_{3,m}$ and $T_{4,m}$. This continues until computing the distance of subsequences that have one point of difference in
their start position. In the second iteration, in the sliding window \(sw_2\), the Euclidean distance is computed between each subsequence and the one that is “two” positions far from it. This continues until Iteration 4. Note that in each iteration the first distance is computed using the normal formula of Euclidean distance, and the other distances are computed using the incremental formula, i.e., Equation (4).

As an optimization of AAMP, we can use the square of the Euclidean distance for comparing the distance of different subsequences, and at the end of the algorithm replace the square of the distance by the real distance in the matrix profile. This optimization reduces the number of sqrt operations done during the algorithm execution.

### Algorithm 1: AAMP algorithm: matrix profile with Euclidean distance

**Input:** \(T\): time series; \(n\): length of time series; \(m\): subsequence length  

**Output:** \(P\): Matrix profile;  

```
for i=1 to n do
  P[i] = \infty; // initialize the matrix profile
for k=1 to n-m do
  dist = Euc_Distance(T_i,m, T_k,m) // compute the distance between T_i,m, T_k,m
  if dist < P[1] then
    P[1] = dist;
  if dist < P[i] then
    P[i] = dist;
for i=2 to n-m+1-k do
  dist = \sqrt{\text{dist}^2 - (t_{i-1} - t_{i-1+k})^2 + (t_{i+m-1} - t_{i+m+k-1})^2}
  if dist < P[i] then
    P[i] = dist;
  if dist < P[i+k] then
    P[i+k] = dist;
```

### 3.3 Complexity Analysis

Here, we analyze the time and space complexity of AAMP. The algorithm contains two loops. In the first loop, in Line 5 the distance between \(T_i,m\) and \(T_k,m\) is computed using the normal Euclidean distance function in \(O(m)\), thus in total Line 5 is executed in \(O(m \times (n - m))\). In the nested loop (Lines 10-15), all operations are done in \(O(1)\), so in total these operations are done in \(O((n-m)^3)\). Thus, the time complexity of the algorithm is \(O((n-m)^3) + m \times (n - m)\) that is equivalent of \(O(n \times (n - m))\). If \(m\) is small compared to \(n\), i.e., \(n \gg m\), then the time complexity of AAMP can be written as \(O(n^2)\). But, if \(m\) is very close to \(n\), i.e., \(m = n - c\) for a small constant \(c\), then the time complexity is \(O(n)\).

The space needed for executing our algorithm is only the array of matrix profile and some simple variables. Thus, the space complexity of AAMP is \(O(n)\).

### 3.4 Extension of AAMP to \(p\)-Norm Distance

In this section, we extend the AAMP algorithm to the \(p\)-norm distance that is a more general distance than Euclidean. The \(p\)-norm functions are used in Lebesgue spaces \((L^p)\), which are useful in data analysis in physics, statistics, finance, engineering, etc.

Let \(T_i,m\) and \(T_j,m\) be two time series subsequences, then their \(p\)-norm distance (for \(p \geq 1\)) is defined as:

\[
DP_{i,j} = \sqrt[p]{\sum_{l=0}^{m-1} (t_{i+l} - t_{j+l})^p}
\]

Notice that the Euclidean distance is a special case of \(p\)-norm with \(p = 2\).

The following lemma gives an incremental formula for computing \(PNORM_{i,j}\).

**Lemma 2.** Let \(DP_{i,j}\) be the \(p\)-norm distance of subsequences \(T_i,m\) and \(T_j,m\). Then, \(DP_{i,j}\) can be computed by using the \(p\)-norm distance of subsequences \(T_i-1,m\) and \(T_j-1,m\), denoted by \(DP_{i-1,j-1}\), as following:

\[
DP_{i,j} = \sqrt[p]{DP_{i-1,j-1}^p - (t_{i-1} - t_{j-1})^p + (t_{i+m-1} - t_{j+m-1})^p}
\]

**Proof.** The proof can be easily done in a similar way as that of Lemma 1. \(\Box\)

Using Lemma 2, we can modify the AAMP algorithm to compute the matrix profile with the \(p\)-norm distance. This can be done just by modifying two lines in Algorithm 1: 1) Line 5: by replacing the Euclidean distance with the \(p\)-norm distance of subsequences \(T_i,m\) and \(T_k,m\); 2) Line 11: incrementally computing the \(p\)-norm distance using the equation of Lemma 2.

The pseudo-code of AAMP algorithm for the \(p\)-norm distance is shown in Appendix. The time and space complexity of the AAMP algorithm for \(p\)-norm is the same as that of AAMP with the Euclidean distance.

### 4 ACAMP: MATRIX PROFILE FOR Z-NORMALIZED EUCLIDEAN DISTANCE

In this section, we propose an algorithm, called ACAMP, that computes matrix profile based on the z-normalized euclidean distance and using the same principle as AAMP, i.e., incremental distance computation in diagonal sliding windows. However, the incremental computation of the distance in ACAMP is different than that of AAMP.

#### 4.1 Incremental Computation of Z-Normalized Euclidean Distance

Let us now explain how ACAMP computes the z-normalized Euclidean distance incrementally. Let \(T_i,m = \{t_i, \ldots, t_{i+m-1}\}\) and \(T_j,m = \{t_j, \ldots, t_{j+m-1}\}\) be two subsequences of a time series \(T\). In ACAMP, we compute the z-normalized Euclidean distance between \(T_i,m\) and \(T_j,m\) using the following five variables:

- \(A_t = \sum_{l=0}^{m-1} t_{l+1}: the sum of the values in \(T_i,m\);\)
The algorithm proceeds in two loops. In the first loop the variables needed for computing the distance (Lines 5 to 9) are computed using their normal formula in $O(m)$ (see Lines 5 to 9). For the other subsequences of the iterations, these variables are computed incrementally, i.e., in $O(1)$.

Note that in the algorithm, for performance reasons we compare the square of the $z$-normalized Euclidean distance of subsequences (Line 10 and 21). By this, we avoid performing $O(n^2)$ sqrt operations in our nested loop. At the end of the algorithm (Lines 26 to 27), in a loop we convert the square distances to the real distances, using $O(n)$ sqrt operations.

### 4.3 Complexity Analysis

Let us now analyze the time and space complexity of ACAMP. The algorithm proceeds in two loops. In the first loop the variables needed for computing the distance (Lines 5 to 9) are computed in $O(m)$, thus in total this part of the algorithm is executed in $O(m \times (n - m))$. In the nested loop, the variables are computed in $O(1)$, thus in total the Lines 16 to 25 are done in $O((n - m)^2)$. Therefore, the time complexity of the algorithm is $O(n \times (n - m))$. If $m$ is small compared to $n$, then the time complexity of ACAMP is $O(n^2)$. But, if $m$ is very close to $n$, i.e., $m = n - c$ for a small constant $c$, then the time complexity of ACAMP is linear, i.e., $O(n)$.

The algorithm needs to keep only some variables and an array for the output matrix profile, thus its space complexity is $O(m)$, i.e., the size of the output.

### 4.4 More Optimization of ACAMP

We can further optimize ACAMP by not comparing the square of z-normalized distance in Lines 11, 13, 22 and 24 in Algorithm 2, but by comparing $F_{i,j}$ defined as follows:

$$F_{i,j} = \frac{(A_i B_j - m C_{i,j})}{(A_i - \frac{1}{m} A_i^2)(B_j - \frac{1}{m} B_j)}.$$
In this section, we compare the execution time of our algorithms AAMP and ACAMP with the state-of-the-art exact motif discovery algorithm SCRIMP++ [10]. We first describe the experimental setup, and then present the results of our experimental evaluation.

5 PERFORMANCE EVALUATION

We implemented our algorithms in MATLAB. For Scrimp++, we use the algorithm available in [1] with step input the usual step size of PreSCRIMP which is 0.25.

The execution times of the three algorithms AAMP, ACAMP and SCRIMP++ are only affected by the length of the time series (i.e., n) and the length of the subsequences (i.e., m). The values inside the time series have no impact on the execution time of the tested algorithms, thus we generated them randomly using a uniform distribution. In our tests, we varied the parameters n and m, and measured their impact on the algorithms execution time. Unless otherwise specified, the default values for m and n are \( m = 2^8 \) and \( n = 2^{18} \) respectively.

The execution tests of the three algorithms were carried out on a machine with Intel®Core™i7-4770 CPU 3.40GHz AÚ x8 processor, on Ubuntu 14.04 LTS and 7.7 Gio memory with the R2015B version of Matlab.

For each test, we perform two experiments and report their average execution times.

5.2 Results

We studied the effect of the time series length (i.e., n) on the execution time of our algorithms. Figure 3 shows the time required by AAMP, ACAMP and SCRIMP++ to compute matrix profile for a fixed subsequence length \( m = 256 \), and with varying time series length values. As seen the execution time of the three algorithms increases with increasing n. But, AAMP and ACAMP perform much better than Scrimp++. The results show that the difference between the performance of AAMP/ACAMP and Scrimp++ increases significantly when n gets higher.

We also studied the effect of subsequence length on the performance of our algorithms. Figure 4 shows the execution time of the three algorithms for computing the matrix profile for time series with a fixed length of \( n = 2^{18} \) and varying the subsequence length from \( m = 256 \) to \( m = 2^{16} \). The response of the our algorithms and that of Scrimp++ decreases when m increases. This is in accordance with our complexity analysis presented in Sections 3.3 and 4.3 showing that for the cases where m is close to n, the time complexity of our algorithms gets linear to n.

6 RELATED WORK

Motif discovery from time series is important for many application domains such as bioinformatics [8], speech processing [2], Seismology [9] and entomology [7]. Matrix profile has been recently proposed as an efficient technique to the problem of all-pairs-similarity search on time series [3, 6, 11, 12, 14].

In [13], Yeh et al. introduced the theoretical foundations of matrix profile, and proposed a first algorithm, called STAMP, for computing the matrix profile over a time series. The algorithm uses a similarity search algorithm, called MASS, that under z-normalized Euclidean computes the distance of each subsequence to other subsequences by using the Fast Fourier Transform (FFT). Other exact algorithms such as Quick-Motif [5], IMD [4], or MK [7] can be fast for cooperative data (those that are relatively smooth data, short motif lengths etc.). But
in less-cooperative data (e.g., seismology data) these algorithms are not efficient [15].

In [15], Zhu et al. proposed an algorithm, called STOMP, that is faster than STAMP. The STOMP algorithm is similar to STAMP in that it can be seen as highly optimized nested loop searches, with the repeated calculation of distance profiles as the inner loop. However, while STAMP must evaluate the distance profiles in random order (to allow its anytime behavior), STOMP performs an ordered search. STOMP exploits the locality of these searches, and reduces the time complexity by a factor of $O(\log n)$. In [10], the authors proposed an extension of STOMP, called SCRIMP++, that converges much faster than STOMP.

To the best of our knowledge, all most all matrix profile algorithms have been developed for z-normalized Euclidean distance. In this paper, we proposed efficient algorithms for a larger class of Euclidean functions. We also proposed an algorithm for the z-normalized case, i.e., ACAMP, that uses the same principle as AAMP, and compared their performance with the state of the art algorithms. The ACAMP algorithm is designed based on an efficient incremental technique that does not need to calculate FFT (in contrast to SCRIMP++).

7 CONCLUSION
In this paper, we addressed the problem of matrix profile computation for a general class of Euclidean distances. We first proposed an efficient algorithm called AAMP for computing matrix profile for the "non-normalized" Euclidean distance. Then, we extended our algorithm for the p-norm distance, which is a general form of Euclidean. Then, we proposed an algorithm, called ACAMP, that uses the same principle as AAMP, but for the case of z-normalized Euclidean distance. Our algorithms are exact, anytime, incrementally maintainable, and can be implemented easily using different languages. To evaluate the performance of our algorithms, we implemented them, and compared their performance with the state of the art algorithm SCRIMP++. The results show excellent performance gains. For example, they show that ACAMP is significantly faster than SCRIMP++. They also show that AAMP is very efficient for computing matrix profile for non-normalized Euclidean and p-norm distances.

REFERENCES
[1] [n. d.]. (n. d.). https://sites.google.com/site/scrimpplusplus

8 APPENDIX A: INCREMENTAL COMPUTATION OF Z-NORMALIZED EUCLIDEAN DISTANCE - PROOF
Here, we present the proof of Lemma 3 that gives an incremental formula for computing matrix profile with z-normalized Euclidean distance.
Proof. Let $\mu_i$ and $\mu_j$ be the mean of the values in the sequences $T_{i,m}$ and $T_{j,m}$ respectively. Also, let $\sigma_i$ and $\sigma_j$ be the standard deviation of the values in the subsequences $T_{i,m}$ and $T_{j,m}$ respectively. Then, the z-normalized Euclidean distance between the subsequences $T_{i,m}$ and $T_{j,m}$ is defined as:

$$DZ_{i,j} = \sqrt{\sum_{l=1}^{m-1} \frac{(t_{i,l} - \mu_i)^2 - (t_{j,l} - \mu_j)^2}{\sigma_i^2 - \sigma_j^2}},$$

where

$$\mu_i = \frac{1}{m} \sum_{l=0}^{m-1} t_{i,l}, \quad \mu_j = \frac{1}{m} \sum_{l=0}^{m-1} t_{j,l}$$

and

$$\sigma_i = \sqrt{\frac{1}{m} \sum_{l=0}^{m-1} (t_{i,l} - \mu_i)^2}, \quad \sigma_j = \sqrt{\frac{1}{m} \sum_{k=0}^{m-1} (t_{j,k} - \mu_j)^2}.$$
We can write the square of $DZ$ as follows:

$$DZ_{i,j}^2 = \sum_{l=0}^{m-1} \left( \frac{t_{i+l} - \mu i}{\sigma_i} - \frac{t_{j+l} - \mu j}{\sigma_j} \right)^2$$

$$= \sum_{l=0}^{m-1} \left( \frac{(t_{i+l} - \mu i)^2}{(\sigma_i)^2} - 2 \left( \frac{(t_{i+l} - \mu i)(t_{j+l} - \mu j)}{\sigma_i \sigma_j} \right) + \left( \frac{(t_{j+l} - \mu j)^2}{(\sigma_j)^2} \right) \right)$$

$$= \sum_{l=0}^{m-1} \left( t_{i+l}^2 - 2t_{i+l}\mu_i + (\mu_i)^2 \right) \left( \frac{1}{(\sigma_i)^2} \right) - 2 \left( \frac{t_{i+l}t_{j+l} - t_{i+l}\mu_j + \mu_j \mu_i}{\sigma_i \sigma_j} \right) + \left( \frac{t_{j+l}^2 - 2t_{j+l}\mu_j + (\mu_j)^2}{(\sigma_j)^2} \right)$$

Let

$$A_i = \sum_{l=0}^{m-1} t_{i+l}, \quad B_j = \sum_{l=0}^{m-1} t_{j+l}, \quad A_i = \sum_{l=0}^{m-1} t_{i+l}^2, \quad B_j = \sum_{l=0}^{m-1} t_{j+l}^2, \quad C_{i,j} = \sum_{l=0}^{m-1} t_{i+l}t_{j+l}.$$

Then, we have:

$$\mu_i = \frac{1}{m}A_i, \quad \mu_j = \frac{1}{m}B_j$$

$$(\sigma_i)^2 = \frac{1}{m^2}A_i^2, \quad (\sigma_j)^2 = \frac{1}{m^2}B_j^2.$$

Then, the z-normalized Euclidean distance can be written as:

$$DZ_{i,j}^2 = \sum_{l=0}^{m-1} \left( \frac{t_{i+l}^2 - 2t_{i+l}\mu_i + (\mu_i)^2}{(\sigma_i)^2} - 2 \left( \frac{t_{i+l}t_{j+l} - t_{i+l}\mu_j + \mu_j \mu_i}{\sigma_i \sigma_j} \right) + \left( \frac{t_{j+l}^2 - 2t_{j+l}\mu_j + (\mu_j)^2}{(\sigma_j)^2} \right) \right)$$

$$= A_i - 2A_i^2 \frac{1}{m} + \frac{A_i^2}{m^2} - 2 \times \frac{C_{i,j}}{m^2 A_i A_j} - \frac{A_i B_j}{m^2} + \frac{A_i B_j}{m^2} + B_j - 2B_j^2 \frac{1}{m} + \frac{B_j^2}{m^2}$$

$$= 2m - 2 \times \frac{m^2 C_{i,j} - m A_i B_j}{\sqrt{(m^2 A_i - A_i^2)(m B_j - B_j^2)}} = 2m - \frac{C_{i,j}}{m^2 A_i A_j} - \frac{A_i B_j}{m^2}$$

As mentioned in Subsection 4.4, by taking

$$F_{i,j} = \frac{(A_i B_j - m C_{i,j})}{(A_i - \frac{A_i^2}{m})(B_j - \frac{B_j^2}{m})},$$

we have $DZ_{i,j} = 2m + 2\text{sign}(F_{i,j}) \sqrt{|F_{i,j}|}$ and we can use the following equivalence in our algorithm:

$$DZ_{i,j} > DZ_{i,k} \iff F_{i,j} > F_{i,k}.$$
Algorithm 3: AAMP algorithm for p-norm distance

**Input:** T: time series; n: length of time series; m: subsequence length

**Output:** P: Matrix profile;

```
1 begin
2 for i=1 to n do
3   P[i] = ∞; // initialize the matrix profile
4 for k=1 to n-m+1 do
5   dist = PNORM_Distance(T_i, m, T_k, m) // compute the distance between T_i, m, T_k, m
6   if dist < P[1] then
7     P[1] = dist;
8   if dist < P[k] then
9     P[k] = dist;
10 for i=2 to n - m + 1 - k do
11   dist = \sqrt[p]{(t_i-1 - t_i-1+k)^p + (t_i+m-1 - t_i+m+k-1)^p}
12   if dist < P[i] then
13     P[i] = dist;
14   if dist < P[i+k] then
15     P[i+k] = dist;
```
This figure "coutmfixe3.jpg" is available in "jpg" format from:

http://arxiv.org/ps/1901.05708v1
This figure "coutnfixe.jpg" is available in "jpg" format from:

http://arxiv.org/ps/1901.05708v1