entropy

MDPI

Article

Algorithmics, Possibilities and Limits of Ordinal Pattern Based Entropies

Albert B. Piek [1,2,*], Inga Stolz [3] and Karsten Keller [1]

[1] Institute of Mathematics, University of Lübeck, Lübeck D-23562, Germany; keller@math.uni-luebeck.de
[2] Graduate School for Computing in Medicine and Life Sciences, University of Lübeck, Lübeck D-23562, Germany
[3] Department of Mathematics, The University of Flensburg, Flensburg D-24943, Germany; inga.stolz@uni-flensburg.de
* Correspondence: piek@math.uni-luebeck.de; Tel.: +49-451-3101-6025; Fax: +49-451-3101-6004

Received: 15 May 2019; Accepted: 27 May 2019; Published: 29 May 2019

Abstract: The study of nonlinear and possibly chaotic time-dependent systems involves long-term data acquisition or high sample rates. The resulting big data is valuable in order to provide useful insights into long-term dynamics. However, efficient and robust algorithms are required that can analyze long time series without decomposing the data into smaller series. Here symbolic-based analysis techniques that regard the dependence of data points are of some special interest. Such techniques are often prone to capacity or, on the contrary, to undersampling problems if the chosen parameters are too large. In this paper we present and apply algorithms of the relatively new ordinal symbolic approach. These algorithms use overlapping information and binary number representation, whilst being fast in the sense of algorithmic complexity, and allow, to the best of our knowledge, larger parameters than comparable methods currently used. We exploit the achieved large parameter range to investigate the limits of entropy measures based on ordinal symbolics. Moreover, we discuss data simulations from this viewpoint.

Keywords: symbolic analysis; ordinal patterns; Permutation entropy; conditional entropy of ordinal patterns; Kolmogorov-Sinai entropy; algorithmic complexity

1. Introduction

Symbolic-based analysis techniques are efficient and robust research tools in the study of non-linear and possibly chaotic time-dependent systems. Initially, an experimental time series $x_0, x_1, \ldots, x_N$ with N in the natural numbers $\mathbb{N}$ is decoded into a sequence of symbols and, if needed, successive symbols are conflated into symbol words of length $m \in \mathbb{N}$. Subsequently, these symbol sequences or symbol word sequences are analyzed mainly by considering symbol (word) distributions or quantifiers based on the distributions, such as entropies. Anomalies in the symbol data or changes in the entropy can be used to detect temporal characteristics in the data. As a result of long-term data acquisition and high sample rates, the study of long-time series is becoming increasingly important in order to provide useful insights into long-term dynamics [1]. Efficient and robust algorithms are required that can analyze long time-series without decomposing the data into smaller series.

In this paper, we are especially interested in the relatively new ordinal symbolic approach which goes back to the innovative works of Bandt and Pompe [2] and Bandt et al. [3]. In the symbolization process, the ordinal approach regards the dependence of $d+1$ equidistant data points resulting in ordinal patterns of order $d \in \mathbb{N}$. The ordinal approach is applied in many research areas to identify interesting temporal patterns that are hidden in the data. Application examples, techniques and further details are listed in the review papers of Kurths et al. [4], Daw et al. [5] and Zanin et al. [6]. In addition,

see the contributions to the special topic "Recent Progress in Symbolic Dynamics and Permutation Complexity. Ten Years of Permutation Entropy" of The European Physical Journal [7] and to the special issue "Symbolic Entropy Analysis and Its Applications" of Entropy [1]. The significant demand for literature on the ordinal idea is due to the fact that the method is easy to interpret and efficient to apply. However, large values of N, d and word lengths m lead to capacity or, on the contrary if N is reduced, undersampling problems. Here m is the number of successive ordinal patterns forming the words considered in the analysis.

This paper covers two main objectives: efficient algorithms for determining ordinal pattern (word) distributions and applicability of ordinal methods. The algorithms use overlapping information and binary representations of ordinal patterns and therefore realize an efficient determination of different entropy measures including and generalizing permutation entropies [2]. To the best of our knowledge, our algorithm not only allows larger parameters N, d and m than methods presently used but also compute a whole series of entropies based on ordinal words of different length and pattern order. A discussion of algorithmic complexity and runtime of the given algorithms in comparison to other ones is provided. Our algorithms are particularly useful for getting deeper insights into the general applicability of ordinal methods, which is the second objective of the paper. We discuss limits of estimating the complexity of finite time series and underlying systems on the base of ordinal patterns and words. There are different restrictions. First of all, if a system provides a large variety of ordinal patterns or ordinal pattern words with sufficiently large d or m, then a time series must be extremely long to represent the variety. This naturally bounds useful orders d and word lengths m (see e.g., [8,9] for further information). Even in the case that extremely long series and large d and m would be accessible, complexity estimation can be limited. Let us shortly explain reasons for that.

Central complexity measures for dynamical systems are related to the Kolmogorov-Sinai entropy being mathematically well-founded but being not easy from the computational and estimation viewpoint. There are different ways of approximating Kolmogorov-Sinai entropy on the base of ordinal pattern (word) distributions for sufficiently large d or m (see Bandt et al. [3] and Gutjahr and Keller [10] for ordinal pattern distributions and Keller et al. [11] for ordinal word distributions). Since there are systems with arbitrarily large, even infinite, Kolmogorov-Sinai entropy, there is no d or m which is large enough for reaching the Kolmogorov-Sinai entropies for all systems. Moreover, in case that data considered are directly an orbit of a time-discrete dynamical system (representing the system), calculation precision bounds the number of accessible ordinal patterns and words. The reason is that, independent from d and m, there are not more patterns and words than possible values of the system in the precision decided. The aspect of calculation precision is also interesting for data simulation including the problem of getting periodicities by rounding.

Many researchers use entropies based on ordinal pattern distributions as complexity measures themselves independent from the Kolmogorov-Sinai entropy, which is particularly justified by the problems mentioned and by the good performance of permutation entropy and similar measures [12,13]. For purely explorative data analysis as it is often given related to automatic learning, classification or comparison of data there is no a priori reason for avoiding large d and m.

The paper is organized as follows. We discuss the idea of ordinal patterns in Section 2, in particular, different pattern representations. Please note that the right choice of representation is substantial in developing our fast algorithms. Moreover, Section 2 provides the main entropy concept used in the paper. Section 3 is devoted to the efficient computation of ordinal patterns and ordinal pattern words from a time series. Here the special binary ordinal pattern representation mentioned above is described and used. Section 4 discusses different methods for establishing ordinal pattern and ordinal pattern word distributions. Furthermore, algorithmic complexity and runtime of our algorithms are investigated. Finally, Section 5 discusses the above specified limits of ordinal pattern-based entropies for complexity quantification in more detail. In this context, also aspects of data simulation are touched. In a certain sense, this paper is complementary to the recent paper [8] of Cuesta-Frau et. al. considering limits from the more practical viewpoint. Please consider the pseudocode descriptions of our

algorithms given in the Appendix A and refer to MATLAB File Exchange [14] for realization in MATLAB 2018b.

2. Ordinal Pattern Representations and Empirical Permutation Entropy

Let $(\mathcal{X}, \preceq)$ be a totally ordered set. Mostly the set of real numbers $\mathbb{R}$ with the usual order $\leq$ is used in application. Nonetheless, our theory applies for the general case of ordinal data, for example on discrete-valued ratings or discrete rating scales as well. For our subsequent algorithmic analysis we demand that evaluation of the order relation $\preceq$ is not too complex and can be performed in constant time, i.e., independent from the compared elements.

We say that two vectors $\boldsymbol{x} = (x_k)_{k=0}^d \in \mathcal{X}^{d+1}$ and $\boldsymbol{y} = (y_k)_{k=0}^d \in \mathcal{X}^{d+1}$ share the same *ordinal pattern* of *order* d iff their components have the same order, i.e.,

$$\boldsymbol{x} \sim_{\mathrm{Ord}} \boldsymbol{y} \Leftrightarrow \forall\, 0 \leq k, l \leq d : x_k \preceq x_l \Leftrightarrow y_k \preceq y_l\,.$$

The equivalence relation $\sim_{\mathrm{Ord}}$ defines $(d+1)!$ equivalence classes on $\mathcal{X}^{d+1}$. Please note that the definition of $\sim_{\mathrm{Ord}}$ and the following ideas can easily be extended to the case that $\boldsymbol{x}$ and $\boldsymbol{y}$ are from different ordered sets. This expands the usage of ordinal analysis to a broader range of applications, by other means incomparable properties can be compared by these methods.

We call a single-pair-comparison within $\boldsymbol{x}$ an *atomic* ordinal information. A pair (x_k, x_l) with $k < l$ is said to be *discordant* or an *inversion* if $x_k \succ x_l$ and *concordant* if $x_k \preceq x_l$. There are $d(d-1)/2$ of these elementary comparisons. Due to the transitive property of order relations, not all variations are possible to occur.

Please note that the set $\mathrm{inv}(\boldsymbol{x}) := \{(k,l) \in \{0,\dots,d\} \mid k < l \wedge x_k \succ x_l\}$ of all inversions within $\boldsymbol{x}$ uniquely determines the regarding ordinal pattern.

2.1. Ordinal Representations

Each ordinal equivalence class is represented by an ordinal pattern. We describe the ordinal patterns formally as elements of a *pattern set* $\mathcal{P}_d := \mathcal{X}^{d+1}/\!\sim_{\mathrm{Ord}}$. Concrete representations of the ordinal patterns have to be bijections from $\mathcal{P}_d$ to a suitable set. The choice of representation of the ordinal information within a pattern has an influence on various factors. These include computability, comparability and behavior under transformation as the pattern order d increases. The most common representations include the *permutation representation* and the *inversion vector representation*. They were considered from the beginning of Ordinal Pattern Analysis including Bandt [15] and Keller et al. [16]. Formally, they are given by the transformations

$$\begin{aligned}
&\mathrm{P}\colon \mathcal{P}_d \to \mathcal{S}_{d+1} \qquad \boldsymbol{x} \mapsto \boldsymbol{\pi} = (\pi_k)_{k=0}^d, \qquad \pi_k = \#\{l \mid 0 \leq l \leq d \wedge (x_l \prec x_k \vee (x_l = x_k \wedge l < k))\}\,,\\
&\mathrm{I}\colon \mathcal{P}_d \to \mathcal{I}_d \qquad \boldsymbol{x} \mapsto \boldsymbol{i} = (i_k)_{k=1}^d, \qquad i_k = \#\{l \mid 0 \leq l < k \wedge x_l \succ x_k\}
\end{aligned}$$

where $\mathcal{S}_{d+1}$ is the symmetric group on $\{0,1,\dots,d\}$, i.e., it contains all permutations of the numbers $0,1,\dots,d$ and where $\mathcal{I}_d = \times_{k=1}^d \{0,\dots,k\}$. Please note that in the literature, the names of these representations vary; moreover some authors use the inverse permutation as the permutation representation instead.

Both transformations are based on order comparisons in different ways. The permutation representation assigns to each vector $\boldsymbol{x}$ the permutation which has the same order within its elements, i.e., for each element x_k of $\boldsymbol{x}$ and π_k of $\boldsymbol{\pi}$ the number of other elements in the respective vectors that are less or equal is the same. Therefore particular properties like the largest and smallest element can directly be determined from this representation. Moreover, the group structure on $\mathcal{S}_{d+1}$ with the permutation composition operation gives many possibilities to analyze the permutation vectors in algebraic terms (e.g., Amigó [17]). However, if a new element is appended to $\boldsymbol{x}$ the whole permutation

vector has to be updated. This is a main disadvantage of the permutation representation that makes computation rather inefficient.

The inversion vector representation is chosen to avoid this problem. It assigns a vector $\boldsymbol{i}$ to each $\boldsymbol{x}$ where the k-th entry is the number of *inversions* of x_k with all previous elements in $\boldsymbol{x}$. In other words, the underlying comparisons only refer to prior appended elements, therefore all preceding elements in $\boldsymbol{i}$ are not affected by the increase of d. This property allows a simple extension to patterns of infinite order d. Moreover, using the *factorial number system*, a special numeral system with mixed radix, the inversion vectors can easily be mapped to the numbers $0, \ldots, (d+1)! - 1$. The map is given by $\sum_{l=1}^{d} l! \cdot i_l$. The resulting number representation allows the most compact description of an ordinal pattern since it combines all ordinal information in a single integer.

For these representations as well as the representation presented in Section 3 one has to note the trade-off between accessibility of the atomic information and flexibility on the one hand and compactness of the representation on the other hand.

2.2. Ordinal Pattern Based Entropies

In the application of the ordinal approach, experimental data $\boldsymbol{x} = (x_n)_{n=0}^{N} \in \mathcal{X}^{N+1}$ of length $N+1$ is decoded into a sequence of ordinal patterns and, if needed, successive patterns are conflated into pattern words of length $m \in \mathbb{N}$.

An important reason ordinal pattern-based measures are interesting for data analysis is that the ordinal pattern (word) sequences usually store much information about the systems dynamics if the data originates from an ergodic dynamical system. This is especially notable, if one focuses on quantifying the complexity of the underlying system. Complexity measures considered are mainly built up on the Shannon entropy of ordinal pattern (word) distributions. In addition, as already mentioned in the introduction, these measures are related to the Kolmogorov-Sinai entropy which is a central complexity measure for dynamical systems. In the following, we give precise definitions of these basic entropy concepts.

Let us regard $d \in \mathbb{N}$ as fixed and let $\boldsymbol{p}$ be some ordinal pattern in $\mathcal{P}_d$. To determine the absolute frequency $h_{\boldsymbol{p}}$ of how often $\boldsymbol{p}$ occurs in $\boldsymbol{x}$, we identify the ordinal structures within each data segment $\boldsymbol{x}^{(n,d)} := (x_k)_{k=n}^{n+d}, n = 0, \ldots, N-d$. The patterns associated with the segments are denoted by $\boldsymbol{p}^{(n,d)}$. Then,

$$h_{\boldsymbol{p}} := \#\left\{ n \in \{0, \ldots, N-d\} \;\middle|\; \boldsymbol{p}^{(n,d)} = \boldsymbol{p} \right\}.$$

Let $\boldsymbol{w}$ be a word of $m \in \mathbb{N}$ successive patterns of order d and let $\mathcal{W}_{d,m}$ be the set of all possible ordinal words of length m and order d. We call the parameter pair (d, m) a *word configuration*. The absolute frequency $h_{\boldsymbol{w}}$ of some ordinal word $\boldsymbol{w} \in \mathcal{W}_{d,m}$, is defined by

$$h_{\boldsymbol{w}} = \#\left\{ n \in \{0, \ldots, N-d-m+1\} \;\middle|\; \boldsymbol{w}^{(n,d,m)} = \boldsymbol{w} \right\}$$

where

$$\boldsymbol{w}^{(n,d,m)} := \left(\boldsymbol{p}^{(n+k,d)} \right)_{k=0}^{m-1}, n = 0, \ldots, N-d-m+1.$$

Please note that we write $\boldsymbol{p}^{(n)}$ and $\boldsymbol{w}^{(n,m)}$ when d is known from the context.

All entropy measures we consider in this paper, are based on quantities $\boldsymbol{H}^{(d,m)}; d, m \in \mathbb{N}$. The $\boldsymbol{H}^{(d,m)}$ are called *Empirical Shannon entropy of order d and word length m*, and defined by

$$\begin{aligned} \boldsymbol{H}^{(d,m)} &:= -\sum_{\boldsymbol{w} \in \mathcal{W}_{d,m}} \frac{h_{\boldsymbol{w}}}{N-d-m+2} \ln \frac{h_{\boldsymbol{w}}}{N-d-m+2} \\ &= \ln(N-d-m+2) - \frac{1}{N-d-m+2} \sum_{\boldsymbol{w} \in \mathcal{W}_{d,m}} h_{\boldsymbol{w}} \ln h_{\boldsymbol{w}}. \end{aligned} \tag{1}$$

They quantify the complexity of ordinal patterns of order d conflated into pattern words of length m in a given time series. Please note that $\frac{h_w}{N-d-m+2}$ is the relative frequency of such words. To compute $\boldsymbol{H}^{(d,m)}; d, m \in \mathbb{N}$ one has to determinate the pattern (word) distributions, which is our main subject of the next section.

3. A Pattern Representation for Efficient Computation

In this section, we describe how all desired ordinal patterns in $\boldsymbol{x}$ can be computed in an efficient and fast way. We will do so by taking advantage of the redundancy in terms of atomic ordinal information between the patterns. We introduce a pattern representation that allows us to do the following tasks in a simple and fast manner when $\boldsymbol{p}^{(n,d)}$ is given:

- Computation of the successive pattern $\boldsymbol{p}^{(n+1,d)}$,
- Computation of patterns $\boldsymbol{p}^{(n,d-m)}, m = 1, \ldots, d-1$ of smaller order,
- Computation of the ordinal words $\boldsymbol{w}^{(n,d-m,m+1)}, m = 1, \ldots, d-1$.

Furthermore, our representation is chosen so that pattern and word frequencies can be determined fast.

3.1. Computation of Successive Patterns

Example: Consider the first six data points of a time series as shown in Figure 1 and the ordinal order $d = 4$. Then, we have two ordinal patterns: the red colored pattern $\boldsymbol{p}^{(0)}$ determined by the points $\boldsymbol{x}^{(0)} = (x_k)_{k=0}^{4}$ and the blue colored pattern $\boldsymbol{p}^{(1)}$ determined by $\boldsymbol{x}^{(1)} = (x_k)_{k=1}^{5}$. Please note that both patterns are maximally overlapping and share the data points $x_1, \ldots, x_4$.

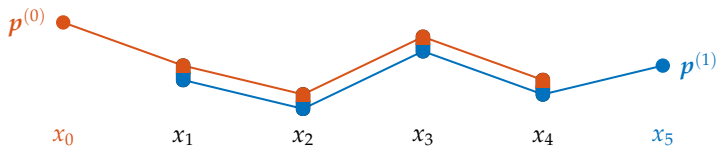


Figure 1. Successive ordinal patterns of order $d = 4$.

To take optimal advantage of this shared ordinal information, we subdivide the task of computing $\boldsymbol{p}^{(1)}$ – when $\boldsymbol{p}^{(0)}$ is given – into three parts:

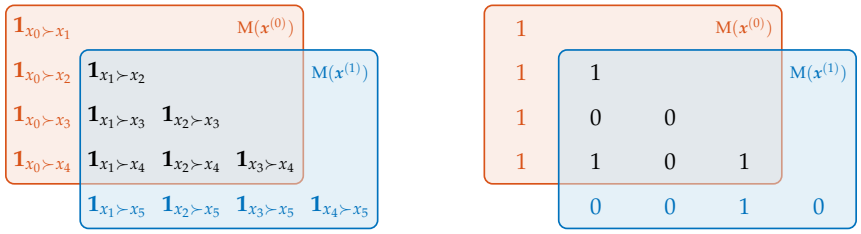
Tasklist 1.

(i) *remove atomic information regarding* x_0,
(ii) *adapt the atomic information regarding* x_1 *to* x_4,
(iii) *add atomic information regarding* x_5.

Generally, these successive steps can be realized as follows: Consider the set $\mathrm{inv}(\boldsymbol{x})$ of inversions within $\boldsymbol{x} \in \mathcal{X}^{d+1}$ and the lower triangular matrix $\mathrm{M}(\boldsymbol{x}) \in \{0,1\}^{d \times d}$

$$\mathrm{M}(\boldsymbol{x}) := \begin{pmatrix} \mathbf{1}_{x_0 \succ x_1} & & 0 \\ \vdots & \ddots & \\ \mathbf{1}_{x_0 \succ x_d} & \cdots & \mathbf{1}_{x_{d-1} \succ x_d}, \end{pmatrix},$$

where $\mathbf{1}$ is the indicator function. It takes the value of one if the statement in the subscript is true. Otherwise, the value is zero. For instance, the schematic illustrations in Figure 2 show the overlapping atomic information between successive matrix representations in general and for our previous example.



(a) **(b)**

Figure 2. Schematic representation of the overlap between successive matrix representations $\mathrm{M}(\boldsymbol{x}^{(0)})$ and $\mathrm{M}(\boldsymbol{x}^{(1)})$ in general (**a**) and for our example (**b**) (compare to Figure 1).

We see that $\mathrm{M}(\boldsymbol{x}^{(n+1,d)})$ is obtained from $\mathrm{M}(\boldsymbol{x}^{(n,d)})$ in terms of Tasklist 1 by executing the following steps:

Tasklist 2.

(i) *removing the first row and column of* $\mathrm{M}(\boldsymbol{x}^{(n,d)})$,

(ii) *keeping the rest of* $\mathrm{M}(\boldsymbol{x}^{(n,d)})$,

(iii) *appending a new row and column at the end with the atomic information* $\mathbf{1}_{x_{n+k}\succ x_{n+d+1}}$ *for all* $k = 1, \ldots, d$.

While the matrix representation M is suitable for computing successive patterns, it lacks of compactness and is difficult to compare. We define a more compact representation which nevertheless directly contains all atomic information – the *binary vector representation*:

$$\mathrm{B}(\boldsymbol{x}) := \mathrm{M}(\boldsymbol{x}) \cdot (2^0 \quad \cdots \quad 2^d)^{\mathrm{T}} = \left(\sum_{0 \le l < k} \mathbf{1}_{x_l \succ x_k} 2^l \right)_{k=1}^{d} \in \bigtimes_{k=1}^{d} \{0, 1, \ldots, 2^k - 1\}. \tag{2}$$

By this definition, we encode each row of M as a binary number where the least significant bit stands in the first column of M. Please note that the binary vector representation is closely related to inversion vectors by

$$\mathrm{I}(\boldsymbol{x}) = \mathrm{M}(\boldsymbol{x}) \cdot (1 \quad \cdots \quad 1)^{\mathrm{T}}.$$

By counting the ones in their binary representations, the elements of the binary vector can be transformed to inversion vector elements.

The key benefit of the binary system is that all operations on binary vectors can be performed as bit operations and can thereby be implemented in an efficient and fast way, especially in low-level programming languages. Here, we use *left* and *right bit shifts* in order to manipulate B and to use overlapping atomic information. By a right bit shift, the least significant bit gets dismissed. This corresponds to a division by 2 rounded down. Analogously, the left bit shift corresponds to a multiplication by 2 or, in other words, a concatenation of B with an additional 0 as the least significant bit. A bit shift on $n \in \mathbb{N}$ by i bits we denote by $n \gg i$ (resp. $n \ll i$) for a right (resp. left) bit shift. Denote that here $\ll$ is a mathematical operator and should not be confused with the "much smaller than" symbol. For example, consider the number 11 which is 1011 in binary representation. Then $1011_2 \gg 1 = 101_2$ which is 5 in decimal. Please note that bit shifts are monotonous functions.

The introduced representation simplifies Tasklist 2 to

Tasklist 3.

(i) *perform a right bit shift on* $\mathrm{B}(\boldsymbol{x}^{(n,d)})$ *and remove the first entry,*

(ii) *keep the rest of the vector,*

(iii) *compute* $\sum_{k=1}^{d} \mathbf{1}_{x_{n+k}\succ x_{n+d+1}} 2^{k-1}$ *and append it to the end of the vector.*

The underlying relationship between consecutive binary vectors used in Tasklist 3(i) reads as follows: Let $\boldsymbol{b}^{(n,d)} = (b_k^{(n,d)})_{k=1}^{d} = \mathrm{B}(\boldsymbol{x}^{(n,d)})$. Then

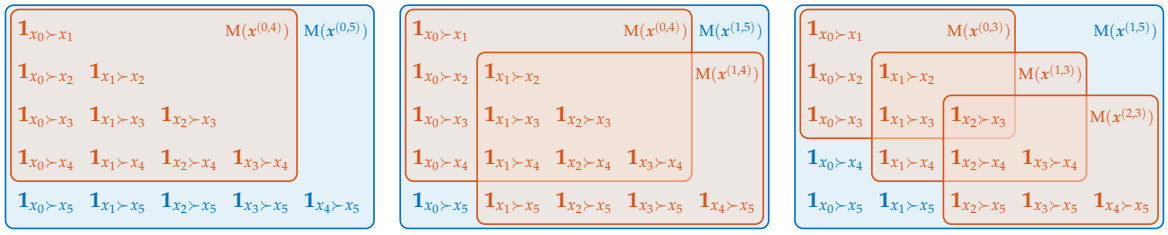
$$b_k^{(n,d)} = b_{k+1}^{(n-1,d)} \gg 1. \tag{3}$$

For the patterns of the example introduced in this section (see Figure 1), we have

$$\begin{aligned}\mathrm{B}(\boldsymbol{x}^{(0)}) &= (1 \quad 3 \quad 1 \quad 11)^{\mathrm{T}} \text{ and}\\ \mathrm{B}(\boldsymbol{x}^{(1)}) &= (3 \gg 1 \quad 1 \gg 1 \quad 11 \gg 1 \quad 4)^{\mathrm{T}} = (1 \quad 0 \quad 5 \quad 4)^{\mathrm{T}}.\end{aligned}$$

3.2. Computation of Patterns of Smaller Order

As with inversion vectors, both, the matrix and binary vector representation, share the possibility to append new elements without changing the other elements. In particular, when the pattern $\boldsymbol{p}^{(n,d)}$ is already computed in terms of M, the patterns of lower order $(\boldsymbol{p}^{(n,d-k)})_{n=0}^{N-d}$ can be obtained by taking the $(d-k)$th leading principle submatrices of M as depicted in Figure 3a. In terms of B, the vector's last k elements have to be removed to obtain the pattern of order $d-k$.



(**a**) (**b**) (**c**)

Figure 3. (**a**) Relationship between patterns of different order. (**b**) Difference between a word of length 2 and order 4 and the pattern of order 5. (**c**) Difference between a word of length 3 and order 3 and the pattern of order 5.

3.3. Computation of Ordinal Words

Ordinal words of length m combine the ordinal information of m successive patterns. Again, the scheme of atomic information shows the difference between one pattern of order d and two patterns of order $d-1$ which form a word of length 2. In the example seen in Figure 3b, the order between x_0 and x_5 is taken into account in $\boldsymbol{w}^{(0,5,1)} = \boldsymbol{p}^{(0,5)}$ but not in $\boldsymbol{w}^{(0,4,2)} = (\boldsymbol{p}^{(0,4)}, \boldsymbol{p}^{(1,4)})$. Therefore, the corresponding bit $\mathbf{1}_{x_0\succ x_5}$ in $\boldsymbol{w}^{(0,5,1)}$ can be set to zero in pursuance of uniting all patterns that are the same except for $\mathbf{1}_{x_0\succ x_5}$. The result corresponds with $\boldsymbol{w}^{(0,4,2)}$. In the next step, $\mathbf{1}_{x_0\succ x_4}$ and $\mathbf{1}_{x_1\succ x_5}$ are set to zero to deduce $\boldsymbol{w}^{(0,3,3)}$ (see Figure 3c). For general $k \in \{1,\ldots,d\}$ we have

$$w_k^{(n,d-m,1+m)} = \begin{cases} w_k^{(n,d,1)} & \text{for } k \le d-m,\\ (w_k^{(n,d,1)} \gg k-(d-m)) \ll k-(d-m) & \text{for } k > d-m. \end{cases} \tag{4}$$

In direct consequence of the results in Section 3.2, for ordinal words we have

$$w_k^{(n,d,m+1)} = \begin{cases} w_k^{(n,d,1)} & \text{for} \quad 1 \le k \le d,\\ \sum_{l=1}^{d} \mathbf{1}_{x_{k-l}\succ x_k} 2^{k-l} & \text{for } d+1 \le k \le d+m+1. \end{cases} \tag{5}$$

Since ordinal words can be treated as special ordinal patterns in our representation, we will often refer to both as patterns.

4. Implementation

In this part, we describe our algorithm that computes a whole series of Shannon entropies based on ordinal words of different word length and pattern order. Specifically, for a given *maximal order* $D \in \mathbb{N}$, the algorithm computes all Shannon entropies $\boldsymbol{H}^{(d,m)}$ as defined in (1), where $d + m \leq D + 1$ with $1 \leq d, m \leq D$. When arranged in an array by pattern order d and word length m, this corresponds to all entries above, left, and on the main antidiagonal. These are the entropies regarding all ordinal words which contain information on at most D consecutive data points from $\boldsymbol{x}$.

In Section 4.1, we will discuss different approaches and our choice. The following Section 4.2 is concerned with the algorithm's structure. Section 4.3 is dedicated to an analysis of the complexity of our algorithm. In the end we apply our algorithm on randomly generated artificial data and analyze its speed in Section 4.4.

4.1. Discussion of Methods

There are several possible approaches for computing the entropy of a given data vector, which differ in terms of time and memory consumption dependent on the maximal order D, data length N. For these parameters, we have usually N significantly larger than D but smaller than $D!$.

To compute the empirical entropy, we need the frequencies of the occurring patterns. Here, we will discuss different ways to compute them. We demand the methods to be adaptable to the transformations (4) and (5) in a sense that the data structure should not be recomputed completely after transforming the patterns.

A first method is to iterate through all patterns and increment a frequency counter when the respective pattern occurs. It has the advantage that there is no need to store all patterns at once; instead, each pattern can be computed separately. A naïve approach is to prepare an array, which length is the number of all possible patterns, and increment its corresponding entries when a certain pattern occurs. Since the number of possible patterns increases factorially in the pattern order D, this method is only practical for small orders. Overall, the array would be very sparse: on the one hand, the data length N is far smaller than $D!$ and thus the number of nonzero entries is relatively small. On the other hand, it was shown by Amigó [18] that the amount of forbidden patterns (i.e., patterns that cannot occur due to the inner structure of the underlying dynamics) also increases superexponentially, leading to further zero entries.

To avoid these zero entries, only the occurring patterns could be counted. This generally requires hashing for the key-value pairing that connects each pattern to its frequency. As an again simple and collision-free example, the hash can be chosen such that the patterns are indexed by their first occurrence. The disadvantage of this method is clearly that the indexing is not gained from information about the pattern itself; to achieve the hash value, it is necessary to search through all hashed patterns to find out whether the current pattern is new or did already occur. Other hash functions can be more effective, but also more complicated and not collision-free. However, it is not immediately clear how to construct them. In addition, the pattern transformations (4) and (5) would make it necessary to recompute all hashes in each step.

Since the problem is basically a searching problem, where the current pattern has to be found in the data structure for the frequencies, more advanced data structures such as search trees are promising approaches. Possible structures are B trees, AVL trees, where searching an element happens in $\mathcal{O}(D \cdot \log N)$ time. An extensive discussion can be found e.g., in standard text books such as the book by Cormen et al. [19]. However, search trees need an underlying total preorder on the set of ordinal patterns. Though all considered representations can be equipped with a total preorder, comparisons will typically have $\mathcal{O}(d)$ time complexity. This would lead to $\mathcal{O}(D \cdot N \cdot \log N)$ time complexity for each entropy, or $\mathcal{O}(D^3 \cdot N \cdot \log N)$ for all desired entropies.

For our computations, we have chosen a different approach where all patterns are given (or computed) beforehand with a total time complexity of $\mathcal{O}(D^2 \cdot N \cdot \log N)$. The choice is based on our idea to compute entropies for different pattern orders and word lengths by manipulating the

representation of the patterns. We explain this idea in the following subsections in detail. Specifically by considering all patterns at once our method allows computing the entropies $(\boldsymbol{H}^{(d,m)})_{m=1}^{D+1-d}$ in one run, which reduces the complexity magnitude of D by one. Though our methods result in the same asymptotic complexity for a single entropy, our experiments show that our approach is faster due to the possibility to vectorize the operations.

4.2. Structure of the Algorithm

We present the design of our algorithm in different ways. First we give a short overview of its structure with schematic illustrations. In the following four parts of the section, we explain key ideas of the algorithm in detail. A pseudocode implementation is provided in Algorithm A1 in the Appendix A. An implementation for MATLAB 2018b is provided by the authors on MATLAB File Exchange [14].

In Figure 4, the algorithm is depicted as a workflow diagram. The basic idea is to precompute all the words $\boldsymbol{w}^{(\cdot,D,1)}$ and use them to compute all other word configurations by (4) and (5), i.e., $\boldsymbol{w}^{(\cdot,d,m)}$ with $d \leq D$ and $m \leq D-d+1$. For each word configuration we then compute the respective entropy. We start with $\boldsymbol{w}^{(\cdot,D,1)} = \boldsymbol{p}^{(\cdot,D)}$ since this is the word which contains all atomic information about the $D+1$ considered points. As shown in the preceding section, other word configurations can be obtained by successively removing the atomic information through bit manipulations.

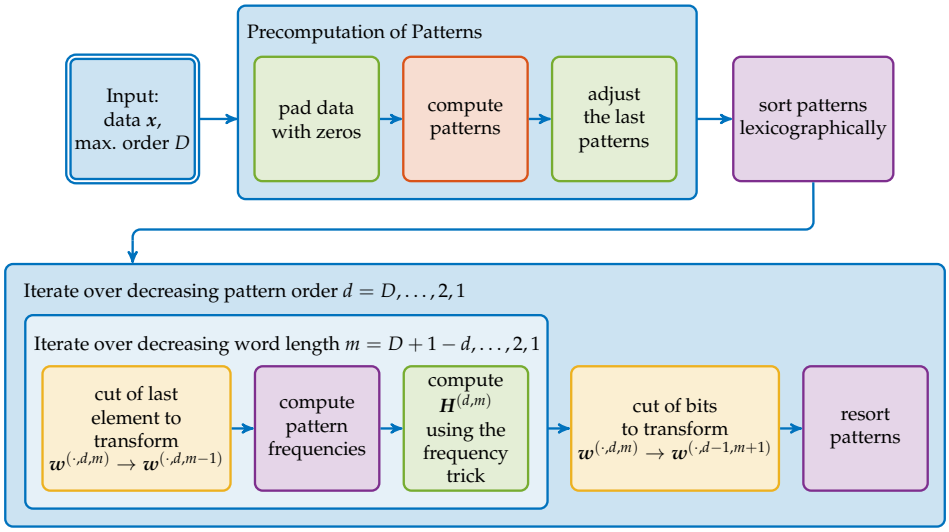


Figure 4. Schematic overview ofr our algorithm. The colors of the steps refer to different aspects of the algorithm discussed in the following subsections: The initial pattern computation (**red**), the processing of these patterns to other word configurations (**yellow**), the calculation of the pattern frequencies (**purple**) and the modifications to include all lower order patterns (**green**).

4.2.1. Initial Pattern Computation

The initial computation of the $\boldsymbol{p}^{(\cdot,D)}$ is shown schematically in Figure 5. The corresponding pseudocode implementation can be found in Algorithm A2 in the Appendix A. It follows the idea of Tasklist 3, but operates in a different sequence. Here, at first the last entry $p_D^{(\cdot,D)}$ is computed for all patterns. The penultimate elements $p_{D-1}^{(\cdot,D)}$ are computed by performing a right bit shift on these last entries (using $k = D-1$ in (3)). Sequentially all prior elements $p_{D-2}^{(\cdot,D)}, \ldots, p_2^{(\cdot,D)}, p_1^{(\cdot,D)}$ get computed in this way. In comparison to computing all patterns consecutively, the computation steps of this methods are highly parallelizable and vectorizable and thereby faster to perform.

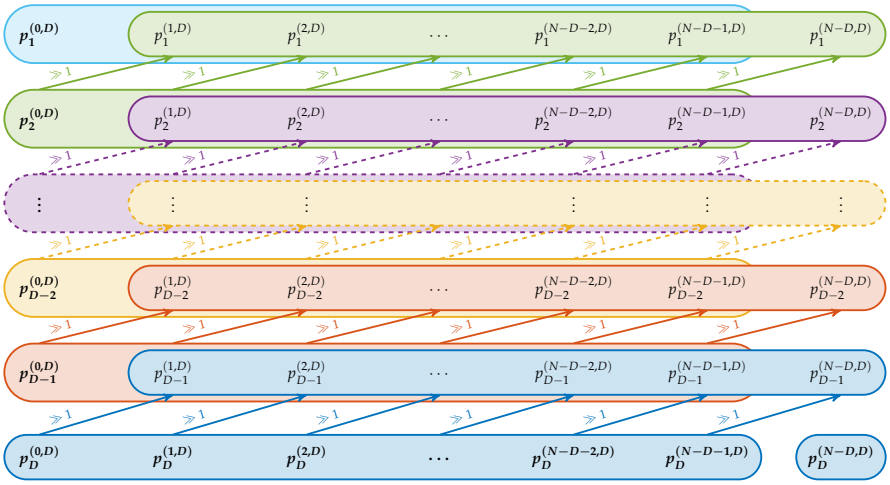


Figure 5. Computation scheme for the ordinal patterns, where each column corresponds to a binary vector. Each color shows one iteration step. The **bold** elements are computed directly while all other elements are derived from them by bit shifts. These are indicated by the arrows.

4.2.2. Obtaining other Word Configurations

The way our main algorithm processes through the different word configurations after computing the $\boldsymbol{p}^{(\cdot,D)} = \boldsymbol{w}^{(\cdot,D,1)}$ is illustrated in Figure 6. In the main iteration using (4) the pattern order gets successively decreased while the word length gets increased. For a fixed pattern order d and word length m, all words with length smaller than m can be computed by simply cutting of last elements of the binary vectors as (5) indicates.

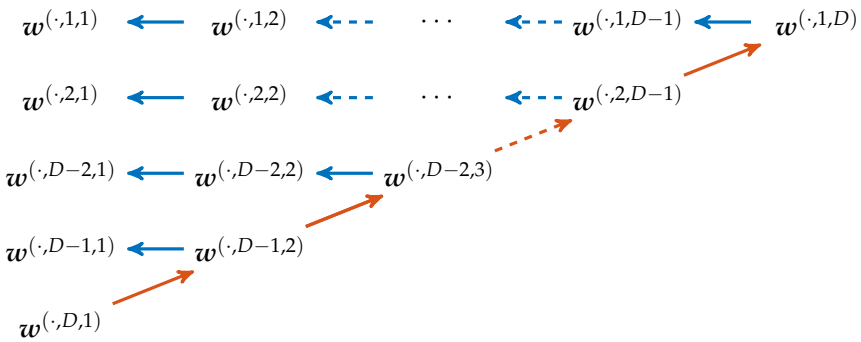


Figure 6. Computation of the different words in the algorithm. **Red** arrows denote applications of (4), while **blue** arrows mean usage of (5).

4.2.3. Computation of the Pattern Frequencies

With each list of words, we compute the word frequencies. In short, the frequencies are determined by sorting the patterns and by computing the length of blocks of the same pattern. Thus, as the first step, all words are sorted lexicographically. Next, we compute the elementwise differences

$$\boldsymbol{w}^{(n,d,m)} - \boldsymbol{w}^{(n-1,d,m)} = \left(w_1^{(n,d,m)} - w_1^{(n-1,d,m)} \quad w_2^{(n,d,m)} - w_2^{(n-1,d,m)} \quad \cdots \quad w_d^{(n,d,m)} - w_d^{(n-1,d,m)} \right)^{\mathrm{T}}$$

between successive patterns. Since the patterns are sorted, the difference vectors are zero vectors for identical patterns and have positive entries for different patterns. The signs of the cumulated differences within the patterns, given by

$$\left(\operatorname{sgn} \left(\sum_{l=1}^{k} \left(w_l^{(n,d,m)} - w_l^{(n-1,d,m)} \right) \right) \right)_{k=1}^{d}$$

indicate a pattern change for all word lengths $1, \dots, d$. The cumulation reflects the property that—due to the lexicographic order—changes in prior entries imply different patterns for all higher pattern lengths. As the positive signs indicate a change of patterns, the pattern frequencies for the words of length m can be determined by taking the difference between the indices of patterns that have positive signs in the m-th element. With the computed absolute frequencies h_p, the corresponding entropy can be computed with (1). Refer to Algorithm A3 in the Appendix A for a pseudocode implementation.

4.2.4. Inclusion of Missing Lower Order Patterns—The Frequency Trick

Not all ordinal words can be computed by transforming the $\boldsymbol{p}^{(\cdot,D)}$ into other word configurations. This is due to the fact that applying (4) and (5) does not change the total number of patterns although the number of patterns contained in the data increases for other configurations. With a total data length $N+1$, we get $N+1-D$ patterns $(\boldsymbol{p}^{(n,D)})_{n=0}^{N-D}$ in the beginning. For arbitrary pattern order d and word length m, we get $N-d-m+2$ words $(\boldsymbol{w}^{(n,d,m)})_{n=0}^{N-d-m+1}$. Since $d+m-1 \leq D$ holds by assumption, we have $N-d-m+1 \geq N-D$. Thus, the last $D-d-m+1$ patterns cannot be derived from the $\boldsymbol{p}^{(\cdot,D)}$. We illustrate the missing patterns in Figure 7.

The procedure would become complicated if all the missing different sized patterns are computed and stored separately. Instead, we desire to compute all patterns by our methods. We achieve this by expanding the data vector with artificial data such that now the missing patterns of the enlarged data vector contain only artificial data and can be neglected. The word $\boldsymbol{w}^{(N-1,1,1)}$ is the last occurring ordinal word in $\boldsymbol{x}$ of all configurations; it contains the ordinal information between the last points x_{N-1} and x_N. By going backwards through the computation process, it can be seen that $\boldsymbol{w}^{(N-1,1,1)}$ is obtained from $\boldsymbol{w}^{(N-1,D,1)}$. This pattern is determined by the points $\boldsymbol{x}^{(N-1)} = (x_k)_{k=N-1}^{N+D}$ which are yet undefined for $N+1 \leq k \leq N+D$. Therefore it suffices to pad these $N+D-(N+1) = D-1$ elements to $\boldsymbol{x}$. With the padded data, we get N patterns in total, which is sufficient for each word configuration.

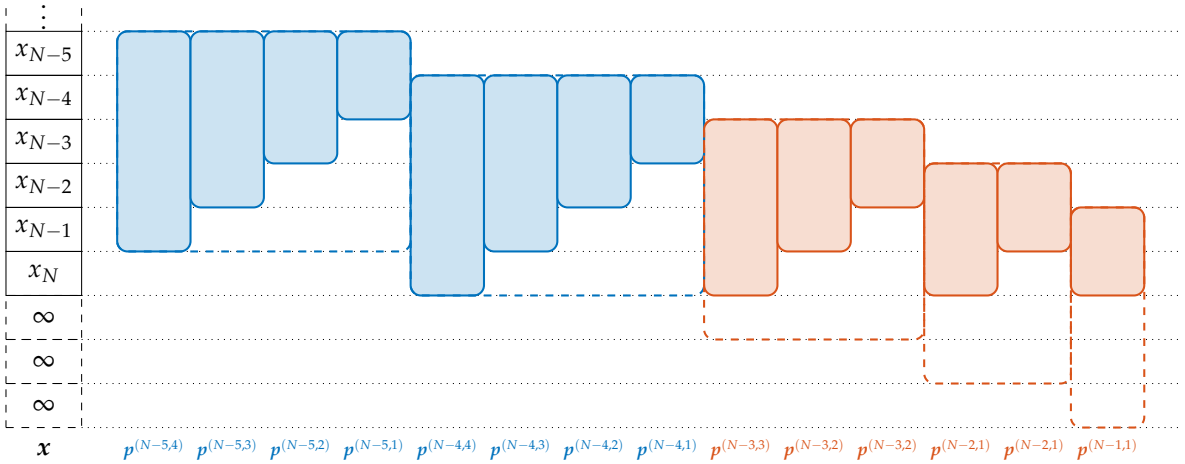


Figure 7. The patterns which are not covered by $\boldsymbol{p}^{(\cdot,D)}$ for $D=4$ and $m=1$. The **blue** patterns are the two last patterns of $\boldsymbol{p}^{(\cdot,D)}$ and the derived patterns of smaller order. The **red** patterns are the missing patterns. They can be derived from the dashed patterns constructed by padding $D-1=3$ elements with value ∞ at the end of $\boldsymbol{x}$.

For convenience, we define an element $\infty \notin \mathcal{X}$ to be the unique value such that $\infty \succ y$ for all $y \in \mathcal{X}$. By padding $\boldsymbol{x}$ with ∞, each atomic information about artificial elements as well as the regarding bit in the binary vector representation is always set to zero.

When the patterns are computed based on the padded data, each required word can be deduced from the N initially computed patterns. As stated above, we have to consider only the first $N-d-m+2$ words $(\boldsymbol{w}^{(n,d,m)})_{n=0}^{N-d-m+1}$ for the entropies. The last $d-m+2$ words $(\boldsymbol{w}^{(n,d,m)})_{n=N-d-m+2}^{N-1}$ partly contain ordinal information regarding the artificial data and have to

be excluded from the entropy computations. Nonetheless, the latter are still needed for deducing words with other configurations and, thus, they cannot simply be deleted.

To prevent these words from distorting the word frequencies without removing them, we use a property of the entropy formula that we call the *frequency trick*. When the entropy is written in terms of absolute frequencies as in (1), patterns of absolute frequency 1 have no effect to the summation due to $\ln 1 = 0$. We take advantage of this by adjusting the binary vectors such that

$$h_{\boldsymbol{w}^{(n,d,m)}} = 1 \text{ for } N - d - m + 2 \leq n \leq N - 1. \tag{6}$$

Thus, we can keep these patterns in our computing procedure. We can use the entropy Formula (1), although the number of pattern $\sum_{\boldsymbol{w} \in \mathcal{W}_{d,m}} h_{\boldsymbol{w}} = N - 1$ is higher than the expected number of patterns $N - d - m + 2$.

We can make all words deduced from the initial patterns satisfy (6) only by adjusting the additional patterns $(\boldsymbol{p}^{(n,D)})_{n=N-D+1}^{N-1}$. This can be done by replacing the artificial zero entries as follows:

$$\begin{aligned}
\boldsymbol{p}^{(N-D+1)} &= \left(p_1^{(N-D+1)} \quad p_2^{(N-D+1)} \quad p_3^{(N-D+1)} \quad \cdots \quad p_{d-2}^{(N-D+1)} \quad p_{d-1}^{(N-D+1)} \quad 2^D\right), \\
\boldsymbol{p}^{(N-D+2)} &= \left(p_1^{(N-D+1)} \quad p_2^{(N-D+2)} \quad p_3^{(N-D+2)} \quad \cdots \quad p_{d-2}^{(N-D+2)} \quad 2^{D-1} \quad 2^D\right), \\
&\vdots \\
\boldsymbol{p}^{(N-2)} &= \left(p_1^{(N-2)} \quad p_2^{(N-2)} \quad 2^3 \quad \cdots \quad 2^{D-2} \quad 2^{D-1} \quad 2^D\right), \\
\boldsymbol{p}^{(N-1)} &= \left(p_1^{(N-1)} \quad 2^2 \quad 2^3 \quad \cdots \quad 2^{D-2} \quad 2^{D-1} \quad 2^D\right).
\end{aligned}$$

The values are chosen in a way that they are larger than the greatest possible value for each entry. Indeed, in the binary vector representation the k-th entry is an integer based on k bits, which reaches its maximum with all bits being ones, which is $\sum_{0 \leq l < k} 2^l = 2^k - 1$. This choice makes it possible to identify any of the additional patterns independent from the data. Thus, each of the additional patterns is fully unique.

On top, each word that is deduced from one of the additional patterns is unique, as long as it contains ordinal information about artificial data points. Recall that the entries with artificial information has values larger than all entries with real information at the same index. Since bit shifts are monotonously increasing functions, applying (4) to each entry keeps the values of the artificial entries to be larger. Furthermore, by (5), only the last elements are removed. After applying both transformations, $\boldsymbol{p}^{(N-D+1)}$ contains no more artificial ordinal information. This is consistent with the increase of the number of patterns when m decreases. The other patterns $\boldsymbol{p}^{(N-D+2)}, \ldots, \boldsymbol{p}^{(N-1)}$ stay unique. In summary, both transformations keep the uniqueness of the patterns as long as artificial information remains and (6) is satisfied.

4.3. Complexity Analysis

In the following, we analyze our algorithms both theoretically and practically. First we give an analysis in terms of Landau Big-$\mathcal{O}$ notation dependent on the parameters N and D. It is summarized in Figure 8. Take into account that in general it is not trivial to extend Big-$\mathcal{O}$ notation to the multidimensional case (c. f. [20]). Nonetheless, since $D \ll N$ holds, the asymptotics of our algorithm is mainly determined by N. Therefore, the one dimensional definitions of Big-$\mathcal{O}$ notation apply. Further note that in terms of complexity, the base of considered logarithms has no influence. Therefore we denote the logarithm with the generic log in this section.

We analyze the algorithm on the individual parts as divided in Figure 4. In terms of time complexity, the initial computation of the patterns discussed in Section 4.2.1 needs $N - D + 1$ comparisons for the last pattern row. For each of the $D - 2$ following rows, one additional comparison and $N - D - 1$ bit shifts are performed, which leads to $DN - D^2 + 2D - 2N + 2$ bit shifts and $N - 1$

comparisons. Assuming both operations can be performed in constant time, we get a time complexity of $\mathcal{O}(D^2 + D \cdot N)$. Since usually N is significantly larger that D, it can be simplified to $\mathcal{O}(D \cdot N)$. In comparison, the naïve approach of finding all ordinal patterns in the binary vector representation by performing all $\binom{D}{2}$ comparisons for each pattern has a worse total time complexity of $\mathcal{O}(D^2 \cdot N)$.

For computing the other representations presented in Section 2 patternwise, there exist $\mathcal{O}(D \log D \cdot N)$ algorithms; the permutation representation can be obtained by sorting the data, for the inversion vector representation Knuth [21] gave an $\mathcal{O}(D \log D)$ conversion from permutations. It is known that $D \log D$ is a lower boundary for sorting algorithms based on comparisons [19]. Thus in terms of complexity, these algorithms are optimal for these representations when computing each pattern by itself. Adapting information from previous patterns allowed us to achieve the improvement in terms of linear time complexity in D. In similar ways, there are ways to obtain linear time complexity for permutations and inversion vectors. The latter was shown by Keller et al. [16], their idea can be adapted for the former straightforward.

Precomputation of Patterns			
Padding	D		$D \cdot N$
Pattern Computation	$D \cdot N$		
Pattern Adjustments	D^2		
Pattern Sorting			$D \cdot N \cdot \log N$
Iteration over d			
Iteration over m			
Transform $\boldsymbol{w}^{(\cdot,d,m)} \to \boldsymbol{w}^{(\cdot,d,m-1)}$	$-$	$D \cdot N$	$D^2 \cdot N \log N$
Compute Frequencies	N		
Compute Entropies	N		
Transform $\boldsymbol{w}^{(\cdot,d,m)} \to \boldsymbol{w}^{(\cdot,d-1,m+1)}$	$D \cdot N$		
Resort Patterns	$D \cdot N \log N$		

Figure 8. Complexity Analysis of our Algorithm, where D describes the maximal pattern order and N describes the length of the data vector.

The padding of the artificial data can be realized in $\mathcal{O}(D)$ insertions which amortized needs $\mathcal{O}(D)$ time. the data is stored in a dynamic array. The adjustment of the last patterns concerns $\frac{1}{2}D(D-1) \in \mathcal{O}(D^2)$ elements. In total, the precomputation of the patterns happens in $\mathcal{O}(D \cdot N)$ time.

In the next step, the patterns get sorted. For this, typical sorting algorithms can be used. Pattern comparisons need D comparisons of their elements in the worst case. Hence the sorting needs $\mathcal{O}(D \cdot N \log N)$. Please note that in practice, there are less comparisons needed. The probabilities of the first k elements being equal is quickly decreasing in k; in particular, it is given by $\sum_{\boldsymbol{w} \in \mathcal{W}_{d,m}} \Pr(\boldsymbol{w})^2$, where Pr denotes the (usually) unknown probability of the word $\boldsymbol{w}$.

The inner loop takes $\mathcal{O}(N)$ time: The transformation takes no additional time since it can be reached by simply ignoring the respective last elements. To attain the changes between the sorted patterns and thereby the frequencies, an iteration through all patterns is needed; for the entropy, all the $\mathcal{O}(N)$ pattern frequencies must be processed. All iterations need $\mathcal{O}(D \cdot N)$ time altogether. Each of the subsequent transformations of the patterns consists of D bit shifts. Together with the resorting of the transformed patterns, all $\mathcal{O}(D)$ iterations take $\mathcal{O}(D^2 \cdot N \log N)$ time in total.

In summary, our algorithm has a time complexity of $\mathcal{O}(D^2 \cdot N \log N)$ for $\frac{1}{2}D(D+1)$ entropies. In principle, for a single entropy each of the loops has to be performed only once, leading to the complexity of $\mathcal{O}(D \cdot N \log N)$.

4.4. Runtime Analysis

In this section, we test our algorithm on artificial data in order to evaluate its performance and compare it to other available programs. The artificial data is generated by a simple random

number generator which gives uniformly distributed random numbers on $[0,1]$. The data is chosen to be uniformly distributed because under this assumption also the ordinal patterns are uniformly distributed. This corresponds to the Shannon entropy taking its maximum. In this sense, the case of uniformly distributed data can be considered as the worst case scenario from the computational viewpoint. In contemplation of investigating the independence of the results from the distribution, we additionally performed our algorithms on normally distributed data. Our computations were performed on MATLAB 2018b on a Windows 10 computer equipped with an Intel Core i7-3770 CPU and 16 GB RAM. We repeated the computations fifty times in order to exclude external distortions and took the mean value of the results.

4.4.1. Computational Time for Pattern and Entropy Computation

First, we restrict our analysis to the pattern computation. We compare our binary vector approach with implementations which determines the inversion and permutation vector representations. For all three representations, we compare the naïve, patternwise approach with the successive methods introduced in Section 3.

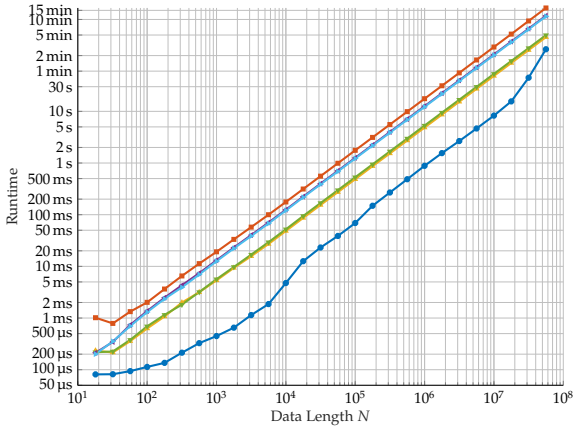
As shown in the prior section, the computational time depends on the data length N and the maximal pattern order D. In our implementation in MATLAB, both parameters are limited due to the following reasons. In a binary vector of order D, its elements are integers with up to D bits. Since arithmetic computations are needed when the bit shifts are performed, D is physically limited to 64 when using unsigned long integers. In practice, it is even limited to 51 since internally, all arithmetic calculations are performed on floating-point numbers in MATLAB. From the 64 bits only 52 bits are reserved for the mantissa from which one bit is needed for the frequency trick. Recall that the maximal possible precision in Matlab is given by approximately $2.2204 \cdot 10^{-16}$.

The data length is in our implementation limited by the machine's memory. In our case, a 16 GB RAM leads to approximately $4 \cdot 10^9$ array elements. Therefore, to store all N patterns of order D, $N \approx 10^8$ data points are possible to process on our machine.

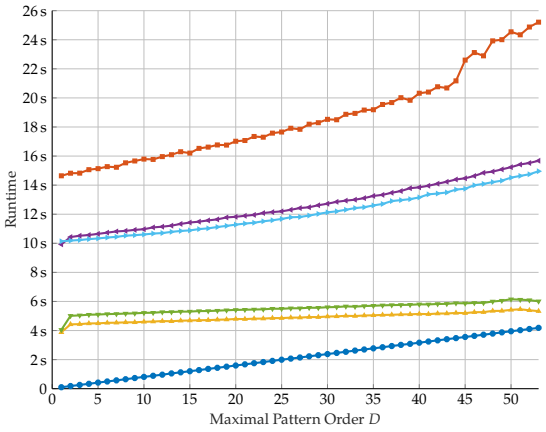
Clearly, for MATLAB or other programming languages there are several possibilities for memory management and high-precision integer arithmetics (c. f. the vpi toolbox [22]), which are offered either by the maintainer or by third-party programs. Anyhow, we restricted ourselves to basic methods of MATLAB.

We tested the runtime of our algorithm for varying N and D on randomly generated data vectors by usage of MATLABs `tic` and `toc` commands, which measure the time between both function calls. The results can be seen in Figure 9a–d. We compare uniformly distributed Figure 9a,b and normal distributed Figure 9c,d data. Both distributions lead to similar results. For both, the asymptotic tendency found in the previous section is observable in the graphs: All implementations have linear tendency in N. The successive implementations are also linear in D, whereas a superlinear behavior can be examined on the naïve approaches. For each representation, the successive implementation is significantly higher than its naïve counterpart. In addition, the successive binary vector approach is the fastest in total. On average, it is 5.38 times faster than the successive algorithm for the inversion vectors and 5.89 times faster than the respective permutation vector algorithm.

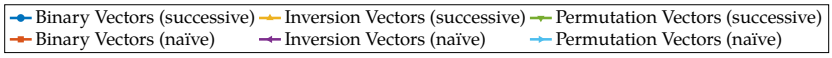
In addition to the runtime tests for the patterns, we tested the time consumption of the whole program. Again, the theoretical considerations of Section 4.3 are confirmed. For increasing data length N, the runtime follows an $N \cdot \log N$ curve. We assume that the deviation for small N is due to the strictly linear time complexity of pattern computation. The proportion of computing time for the patterns tends to 10%. For varying order D, the quadratic regression we did on our results for the parameter D gives an almost perfect fit (compare Figure 9f). With increasing maximal pattern order D, the proportion of computing time for the pattern calculation relative to the total time decreases considerably. Most of the computation time is spent on the frequency computation and the sorting of the patterns. Therefore, this part seems to be a promising starting point for future improvements.

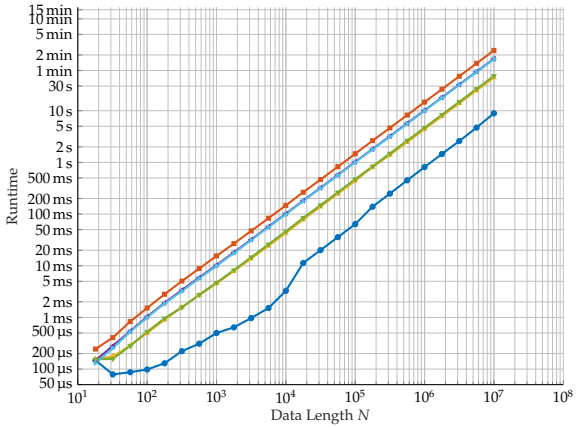


(a) Runtime of pattern computation for fixed $D = 10$ and uniformly distributed random data with changing length N.

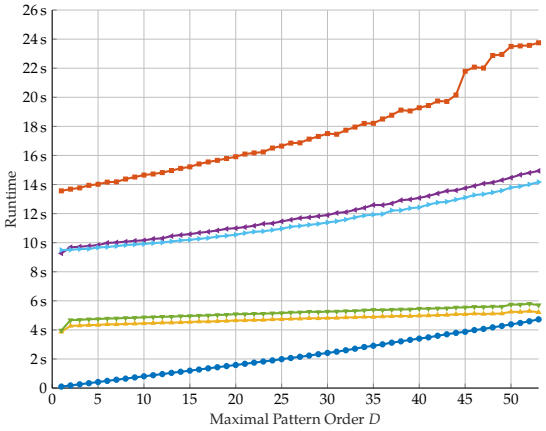


(b) Runtime of pattern computation for changing D and uniformly distributed random data with fixed length $N = 10^6$.

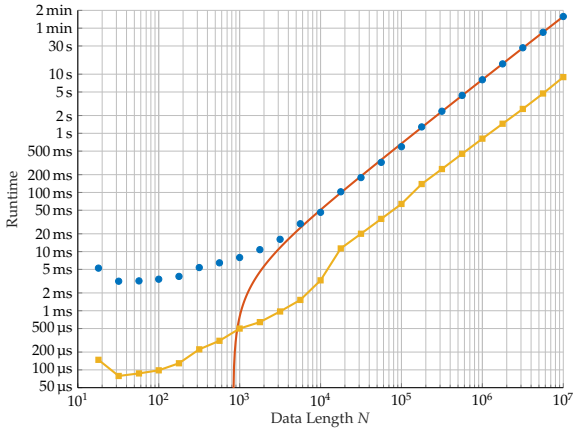




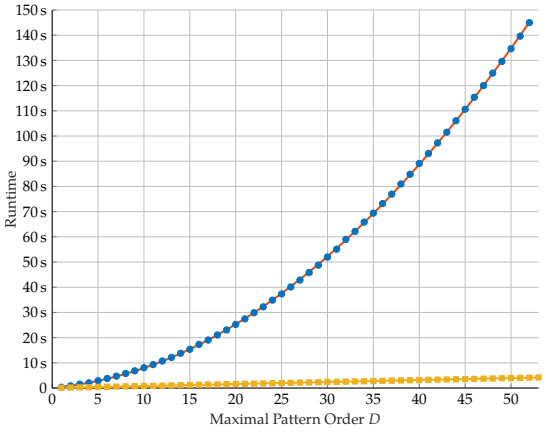
(c) Runtime of pattern computation for fixed $D = 10$ and normal distributed random data with changing data length N.



(d) Runtime of pattern computation for changing D and normal distributed random data with fixed data length $N = 10^6$.



(e) Runtime of entropy computation for fixed $D = 10$ and uniformly distributed random data with changing data length N (**blue** dots). For comparison, the **yellow** curve marks the time for the pattern computation. The **red** line is a nonlinear fit with the fitting model $a \cdot N \ln N + b$ and $a = -3.3 \cdot 10^{-3}$ and $b = 5.8615 \cdot 10^{-7}$; $MSC = 3.925 \cdot 10^{-3}$.



(f) Runtime of entropy computation for changing D and uniformly distributed random data with fixed data length $N = 10^6$ (**blue** dots). For comparison, the **yellow** curve marks the time for the pattern computation. The **red** line is a quadratic fit with the fitting model $a \cdot D^2 + b \cdot D + \cdot c$ and $a = 0.04789$, $b = 0.2983 \cdot 10^{-7}$ and $c = 0.1667$; $SSE = 1.4487$, $R^2 = 1$.

Figure 9. Runtime analysis of the entropy and pattern computation dependent on the data length N and the maximal pattern order D. The resulting runtimes are averaged over 50 trials.

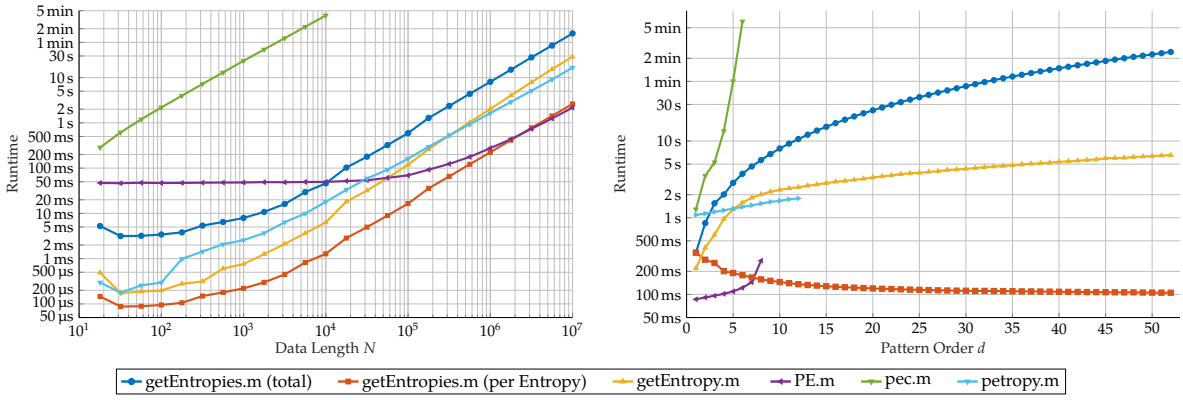
4.4.2. Comparison with Other Implementations

We compare our algorithm to the implementations by V. Unakafova (see *PE.m*, [23], detailed explanations in the accompanying literature [24]), G. Ouyang (*pec.m*, [25]) and A. Müller (*petropy.m* [26], description in [9]) in the context in which the programs are comparable. None of the other implementations can compute entropies for ordinal words. The program *pec.m* computes a single entropy for a given pattern order and supports time delays. *petropy.m* again gives only one entropy, but allows multiple time delays. In addition it offers several ways to treat equal values in the data. The implementation in *PE.m* computes the permutation entropy for sliding windows over the data set, giving the entropy for each window.

Therefore we restrict our comparisons to the case of simple ordinal patterns ($m = 1$). Though our algorithms can be extended easily to support time delays, we will not use time delays in any implementation. At last the sliding window in *PE.m* was chosen such that the number of windows equals one.

For further comparability, we divide the runtime for our algorithm by the number of computed entropies $\frac{1}{2}D \cdot (D+1)$ to get an approximation for the mean time needed for a single entropy. In addition, we tested a modified version of our algorithm that computes the entropy for a single word configuration.

Figure 10 shows the results of our analysis. Again, we analyzed the runtime in dependence of the data length N on the left and pattern order D on the right.



(**a**) Runtime comparison of different implementations for permutation entropy with fixed $D = 8$ and uniformly distributed random data with changing data length N. Measurements of *pec.m* were only performed up to $N = 10^4$ due to the long runtimes.

(**b**) Runtime comparison of different implementations for permutation entropy with changing D and uniformly distributed random data with fixed data length $N = 10^6$. Again, the runtime of *pec.m* was only measured for $D < 8$ because of its bad performance. *PE.m* is only implemented for $D < 9$; *petropy.m* works only if $D \cdot \log_{10} D > 15$ which means $D < 14$.

Figure 10. Comparision of the performance of different MATLAB scripts for permutation entropy computation. The **blue** curve is the total runtime of our approach while the **red** curve gives the runtime per entropy. The modified version for a single entropy is marked **yellow**. Both are compared with the approaches from V. Unakafova (**purple**), G. Ouyang (**green**) and A. Müller (**cyan**).

In *pec.m*, the first method from our discussion in Section 4.1 was chosen. This led to very long computing times even for small N and D, making it by far the slowest approach (the green line in Figure 10).

The performance of *PE.m* (purple line) varies. It is clearly the fastest of the compared approaches for small pattern orders. Though, for $d = 8$, bigger data sets are needed to compete with the theoretical average time per entropy of our method. Responsible for the high computation speed is mainly the usage of lookup tables to determine succeeding patterns. While this method clearly reduces computation time, these tables have to be available and increase superexponentially in size,

which limits its usage to pattern orders of size 8 or smaller. With increasing size of the lookup table, its speed advantages decrease.

The implementation of *petropy.m* (cyan line) has a similar performance to our modified one-entropy-programm, mainly because the program uses a similar approach to ours. It uses the same method of frequency determination. However, it uses the factorial number representation for the ordinal patterns. This choice leads to the upper limit of the pattern order since these numbers can be as high as $D!$. With an maximal possible integer value of 2^{53} in MATLAB, this results in $D = 13$ as the largest possible pattern order, a slightly higher range for D than *PE.m*.

In direct comparison, our approach that computes all $\frac{D\cdot(D+1)}{2}$ entropies takes (apart of the by far slowest *pec.m*) the most time for computing. On the other hand, the average time per entropy is on wide parameter ranges the fastest. Only for small d, the approach of V. Unakova is faster. The modified version competes well with the other approaches. Nevertheless it is slower than the average computing time per entropy since the structures used for optimizing the computation of multiple entropies cannot be exploited.

Clearly, the main advantage of our algorithm lies in the extended parameter range for the pattern order d. If over the top the behavior of the entropy in dependence of d is from particular interest, our approach saves much time compared to computing alls the entropies serially.

5. Limits of Ordinal Pattern Based Entropies

As already mentioned in the introduction, most complexity measures for dynamical systems are strongly linked to the Kolmogorov-Sinai entropy (abbrev KSE). In our following discussion of ordinal pattern-based entropies in that context, we forgo to give detailed definitions of this concept. Instead, we refer for details and more background of the following to [13]. Furthermore we restrict the following considerations to the most simple case of a one-dimensional dynamical system. Please note that with the appropriate generalizations, stochastic processes could be included in the discussion withal.

By a (measurable) *dynamical system* $(\mathcal{X}, \mathcal{A}, T, \mu)$ we understand a probability space $(\mathcal{X}, \mathcal{A}, \mu)$ equipped with a map $T : \mathcal{X} \hookleftarrow$ being measurable with respect to $\mathcal{A}$, where μ is invariant with respect to T. The latter means that $\mu(T^{-1}(A)) = \mu(A)$ for all events $A \in \mathcal{A}$ and describes stationarity of the system. X is considered as the *state space* and T provides the dynamics. A dynamical system as given produces a time series $\boldsymbol{x} = (x_n)_{n=0}^{N}$, when $x_0 \in \mathcal{X}$ is given and x_n is the n-th iterate of x_0 with respect to T for $n = 1, 2, \ldots, N$. If the probability measure is ergodic, as we assume in the following, statistical properties of the given system can be assessed from such time series for sufficiently large N. In particular, probabilities of patterns can be estimated by their relative frequencies within the time series. With this setting there are several ways for estimating the Kolmogorov-Sinai entropy from ordinal pattern-based entropies given a time series from the system.

In theory, the following quantifiers had shown to be good estimators of the Kolmogorov-Sinai entropy for sufficiently large N, d and m:

$$\tfrac{1}{d}\boldsymbol{H}^{(d,1)} \text{ if } T \text{ is an interval map with countably many monotone pieces,} \tag{7}$$

$$\tfrac{1}{m}\boldsymbol{H}^{(d,m)} \text{ and } \boldsymbol{H}^{(d,m+1)} - \boldsymbol{H}^{(d,m)} \text{ generally.} \tag{8}$$

Please note that $\frac{1}{d}\boldsymbol{H}^{(d,1)}$ is called the *empirical Permutation entropy of order d* (abbrev. ePE) and $\boldsymbol{H}^{(d,2)} - \boldsymbol{H}^{(d,1)}$ the *Empirical Conditional entropy of Ordinal Patterns of order d* (abbrev. eCE). Both are estimators of the respective entropies of the dynamical system which base on the time series $\boldsymbol{x}$ and whose precise definitions are specified in the given literature.

For (7) the statement follows directly from a recent result of Gutjahr and Keller [10] generalizing the celebrated result of Bandt et al. [3] that for piecewise monotonous interval maps the KSE and the Permutation entropy are coinciding. As reported in [12,13], the latter entropy seems to be a good estimator of the KSE for sufficiently large d with some plausible reasoning, but this point is not completely understood.

The problem in the statement above is what sufficiently large means. First of all, in the case of existence of many different ordinal patterns in a system, N must be extremely large in order to realize them all in a time series (for practical recommendations relating N and d, see e.g., [8,9]). Even if arbitrarily long time series would be possible, there would be serious problems to reach the KSE. Let us demonstrate this for the ePE (compare (7)).

For $d \in \mathbb{N}$ it holds

$$\tfrac{1}{d}\boldsymbol{H}^{(d,1)} \leq \frac{\ln((d+1)!)}{d} = \frac{\sum_{k=1}^{d+1} \ln k}{d} \tag{9}$$

since maximal Shannon entropy is given for the uniform distribution. The right side of this formula is very slowly increasing. For example, for $d = 100$ it is less than 4, saying that for maps with KSE larger than 4 one needs ordinal patterns of order larger than 140 to be able to get a sufficiently good estimation. The larger the KSE is, the larger the d for its reliable estimation theoretically must be, and the KSE can be arbitrarily large. One reason for this is that if a map T has some KSE c, then its n-th iterate $T^{\circ n}$ has KSE $n \cdot c$ (see e.g., [27]).

Formula (9) is also interesting from the simulation viewpoint. Given an (one-dimensional) dynamical system and some precision, then usually simulations stick to the precision: Given a value in its precision, the further iterates in their precision are determined. This shows that the precision bounds the number of possible ordinal patterns independent of d.

For example, consider the state space $[0,1]$ and the usage of double-precision IEEE 754 floating-point numbers, which is the standard data type for numerical data in the most programming languages. Recall that these numbers have the form $s \cdot m \cdot 2^e$, where s, m and e are coded within 64 bits of information. One bit is reserved for the sign s and 11 bits are used for the exponent e. The remaining 52 bits are left for the fraction value (mantissa) m. The exponent generally ranges from -1022 to 1023, for our particular state space, only negative exponents are needed. For each choice for the exponent, there are 2^{52} possible values of the mantissa. Therefore, there are in total $1022 \cdot 2^{52} \approx 4.6 \cdot 10^{18}$ distinct numbers possible in a simulation, and more distinct ordinal patterns cannot be found. Since already for $d = 20$ there are $21! \approx 5.1 \cdot 10^{19} > 4.6 \cdot 10^{18}$ possible ordinal patterns, simulations do not provide an ePE larger than $\frac{\ln 21!}{d} \leq \frac{\ln 21!}{20} \leq 2.269$. Because there are one-dimensional maps with arbitrary KSE, this limits the estimation of the KSE by the simulation.

The kind of simulation described, which we refer to as the naïve simulation, does not reproduce the real-world situation. The measuring process itself goes hand in hand with precision loss and therefore measuring errors occur. By iterating through these erroneous values, the error cumulates. In consequence due to the chaotic behavior, the values obtained after a few iterations have nothing left in common with the real values. For example, when identifying a real-world system with a dynamical model, the simulation can in consequence produce periodic sequences of values, although the sequence of measured values does not contain periodicities. This can reduce the number of ordinal patterns and so corresponding entropies.

In real-world systems, the measuring process and its belonging errors have no influence to the system. There—under assumption of other sources of noise—the system generates its time-dependent values of arbitrary accuracy exactly. Only in the moment where we want to know a certain iterate, we measure it with the given limited precision. Therefore, we desire to have a simulation where the iteration itself can be performed exactly for starting values of unlimited precision.

Here for demonstration purposes we are especially interested in the system $([0,1], \mathcal{B}, L, \mu)$, where $\mathcal{B}$ are the Borel sets on $[0,1]$, L is the logistic map defined by $L(x) = 4x(1-x)$ for $x \in [0,1]$ and μ is the probability measure on $\mathcal{B}$ with a density p on $[0,1]$ given by $p(x) = \frac{1}{\pi\sqrt{x(1-x)}}$ for $x \in]0,1[$.

To mimic the real situation, we generate a sequence $x'_0, x'_1, x'_2, \ldots$ by using other systems that are topologically (semi-)conjugated to the logistic map. These systems show to be suitable to our simulation requirements.

In the first step, we take advantage of the well-known fact that the system $([0,1], \mathcal{B}, L, \mu)$ is semi-conjugated to the angle doubling map on $[0, 2\pi]$ which is given by $A(\beta) = 2\beta \bmod 2\pi$ equipped

with the Lebesgue measure (compare [28] for this and the following statements). The semi-conjugacy is given by the map ψ with

$$\psi(\beta) = \sin^2(\beta)$$

for $\beta \in [0, 2\pi]$. This means that

$$L(\psi(\beta)) = \psi(A(\beta)) \tag{10}$$

for all $\beta \in [0, 2\pi]$, i.e., applying L on $[0, 1]$ corresponds to doubling the arc length of a segment on the unit cycle and thereby doubling the angle. Furthermore, μ is the image of λ, i.e., $\mu(B) = \lambda(\phi^{-1}(B))$ for all $B \in \mathcal{B}$. The relationship is illustrated in Figure 11.

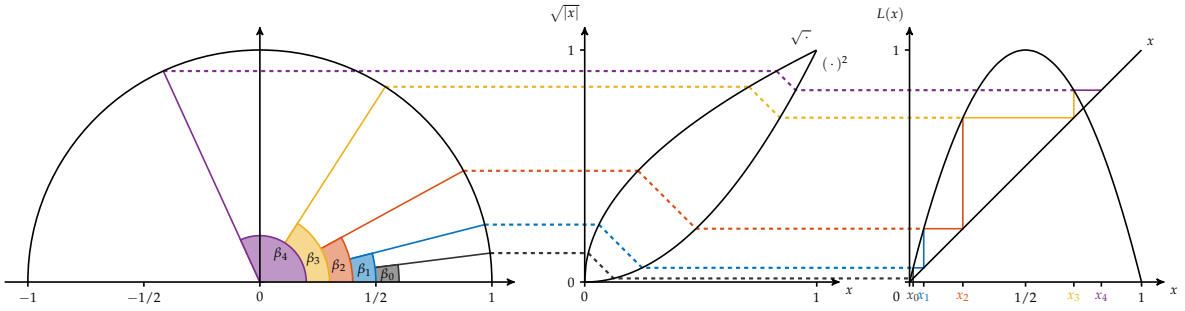


Figure 11. Semi-Conjugacy between the orbits of the logistic map L and the angle-doubling function.

As the second step, we use another conjugacy, namely that the angle doubling map is conjugated to the shift function on all infinite 0-1-sequences, i.e., $\{0, 1\}^\infty$ equipped with the $(\frac{1}{2}, \frac{1}{2})$-Bernoulli measure ν. The latter assigns to each cylinder set of size n of $\{0, 1\}^\infty$ the measure 2^{-n}. The shift is defined as

$$S : \{0, 1\}^\infty \hookleftarrow, (b_1, b_2, b_3, \dots) \mapsto (b_2, b_3, b_4, \dots).$$

The conjugation between a 0-1-sequence $(b_j)_{j\in\mathbb{N}}$ and an angle $\beta \in [0, 2\pi]$ is given by the binary expansion

$$\beta = \phi((b_j)_{j\in\mathbb{N}}) = 2\pi \sum_{j=1}^{\infty} b_j 2^{-j}.$$

Please note that ϕ is only almost everywhere bijective, for example $\phi((1, 0, 0, \dots)) = \phi((0, 1, 1, \dots)) = \pi$. Analogously to (10), it follows that

$$A(\phi((b_j)_{j\in\mathbb{N}})) = \phi(S((b_j)_{j\in\mathbb{N}})) \tag{11}$$

for all $(b_j)_{j\in\mathbb{N}} \in \{0, 1\}^\infty$. Concluding (10) and (11), the following diagram commutes:

$$\begin{array}{ccccc} (\{0,1\}^\infty, \nu) & \xrightarrow{\phi} & ([0,1], \lambda) & \xrightarrow{\psi} & ([0,1], \mu) \\ \downarrow S & & \downarrow A & & \downarrow L \\ (\{0,1\}^\infty, \nu) & \xrightarrow{\phi} & ([0,1], \lambda) & \xrightarrow{\psi} & ([0,1], \mu) \end{array}$$

We can generate an orbit $x_0, x_1, x_2, \dots$ for L as follows: Take a random sequence $b_1, b_2, b_3, \dots$, of elements in $\{0, 1\}$ meaning that each b_n is taken from the symbols $\{0, 1\}$ with equal probability and in an independent way and let

$$x_n = \psi(\phi(S^{\circ n}((b_j)_{j\in\mathbb{N}}))) = \sin^2\left(2\pi \sum_{j=1}^{\infty} b_{j+n} 2^{-j}\right).$$

Let $p \in \mathbb{N}$ be the number of binary digits defined by a given precision. The approximation up to precision p given by

$$x'_n = \sin^2\left(2\pi \sum_{j=1}^{p} b_{j+n} 2^{-j}\right).$$

provides a sequence $x'_0, x'_1, x'_2, \ldots$ as desired. The pleasant point is that in the simulation we can start from $b_1, b_2, \ldots, b_p$ and than can append b_{p+1} by random choice and delete b_1, then add b_{p+2} and delete b_2, then add b_{p+3} and delete b_3, etc. We refer to this simulation (see Algorithm 1) as the advanced one.

Algorithm 1: LOGISTIC-MAP-SIMULATION

Input: number of iterations n, precision p.
Output: array $\boldsymbol{L}$ with simulated values of the logistic map.
begin
  choose start value $\alpha \sim U(0,1)$ randomly ; // $\beta = 2\pi \cdot \alpha$
  $\alpha_p \longleftarrow \lfloor 2^p \cdot \alpha \rfloor \cdot 2^{-p}$; // round to precision
  $L[0] \longleftarrow \sin^2(2\pi \cdot \alpha_p)$;
  for $k = 1, \ldots, n$ **do**
    $b \sim B(1, \frac{1}{2})$ chosen randomly ;
    $\alpha_p \longleftarrow (2\alpha_p \bmod 1) + b \cdot 2^{-p}$; // add new precision digit
    $L[k] \longleftarrow \sin^2(2\pi \cdot \alpha_p)$;
  end
end

We have determined the ePE and the eCE for time series related to the logistic map L and its iterates $L^{\circ 2} = L \circ L, L^{\circ 3} = L \circ L \circ L$ in dependence on the order d. The results based on time series of length 10^7 are presented by Figure 12. The time series behind the pictures on the left side ((a) and (c)) was obtained by the naïve simulation and on the right side ((b) and (d)) by the advanced simulation, both under the double precision described above. In (a) and (b), we have added **blue** colored curves showing the upper bound of the ePE (9) for d. Please note that the results were obtained by our algorithm from Section 4. It allowed to compute all values at once for each curve.

First of all, for L and $L^{\circ 2}$ (**red** and **yellow** colored curves) with KSE $\ln 2 \approx 0.693147$ and $2 \ln 2 \approx 1.38629$, respectively, the values of eCE restore the KSE much better than the ePE. For a very good performance of the eCE, consider d in the interval between 6 and 16 for L and between 6 and 9 for $L^{\circ 2}$. For d left of the intervals, the ordinal patterns seem to be too short to capture the full complexity, for d right of the intervals the time series seem to be to short relative to d to capture the ordinal pattern complexity. As already reported in [12,13] for L, convergence of the ePE seems to be too slow to get good estimations for moderately long time series.

It is not excluded that for L or $L^{\circ 2}$ the naïve algorithm sometimes runs in a loop or, more general, reduces variety of ordinal patterns. For $L^{\circ 3}$, we see such behavior in Figure 12 (cf. the **purple** curves), namely larger empirical entropies are reached for the advanced simulation than for the naïve one. It is remarkable that the eCE for $d = 8$ is near to the KSE $3 \ln 2 \approx 2.07944$, it seems however that the set of too small and the set of too large d as described above are overlapping, such that the KSE is not reached completely.

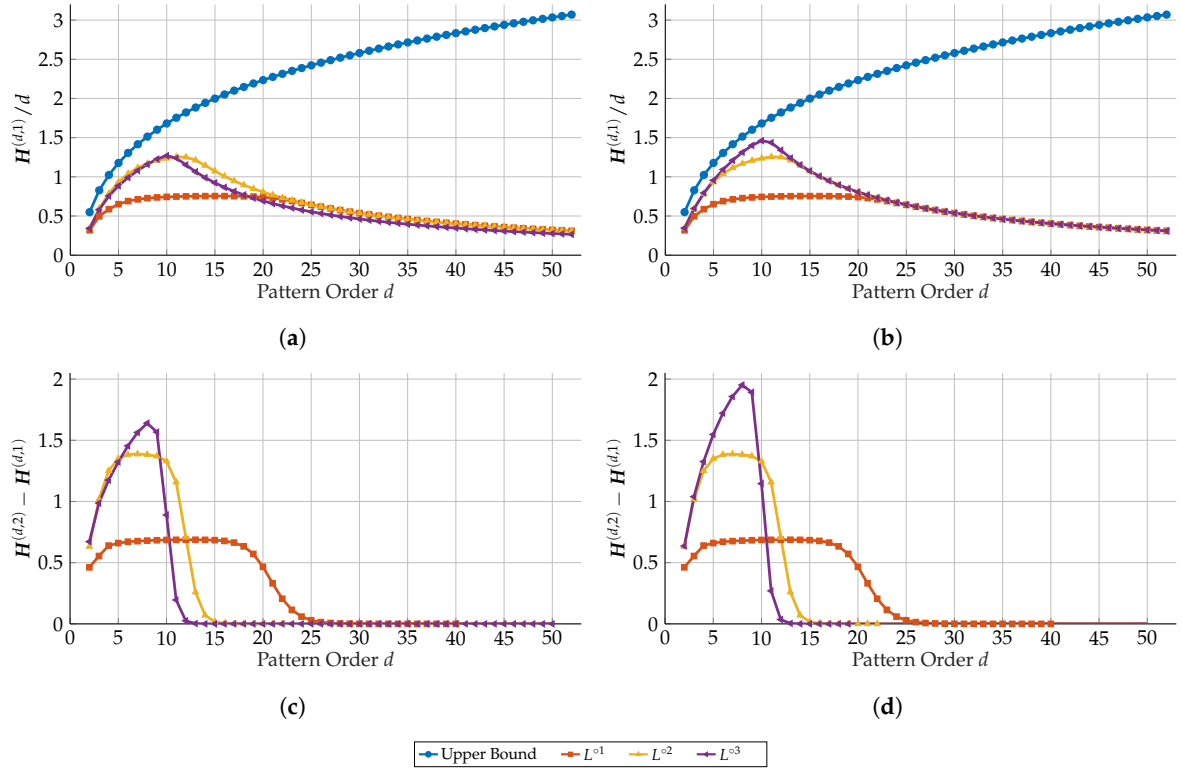


Figure 12. ePE (**a**,**b**) and empirical conditional entropy of ordinal patterns (**c**,**d**) for the logistic map $L = L^{\circ 1}$ and its iterates $L^{\circ 2}, L^{\circ 3}$ in dependence on the pattern order d: Left side pictures (**a**,**c**) show results based on the naïve simulation and right side pictures (**b**,**d**) on the advanced simulation, each based on $N = 10^7$ iterates. For comparison, in (**a**,**b**) the upper bound of the ePE (9) for d was added (**blue** curves).

Our exemplary considerations underline that some deeper thinking about the (mathematical) structure of systems and measurements, possibly also on the base of symbolic dynamics, is important for simulating data and beyond. To develop powerful and reliable tools for data analysis, moreover, a better understanding of ordinal pattern-based entropies for complexity quantification and their relationship to other complexity measures remains a permanent challenge.

Author Contributions: A.B.P. wrote the main parts of the paper and the complete software, and prepared all figures. I.S. substantially contributed to Sections 1 and 2.2 and K.K. to Sections 2.2 and 5. Moreover, K.K. supervised writing the paper. All authors have read and approved the final manuscript.

Funding: This research received no external funding.

Acknowledgments: The first author thanks Max Bannach for fruitful discussions about complexity and data structures as well as for insights from a theoretical computer scientists point of view.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Algorithms

Algorithm A1: GET-ENTROPIES

Input: Data array $\boldsymbol{x} \in \mathcal{X}^{N+1}$, pattern order D.
Output: Array $\boldsymbol{H}$ of Shannon entropies.
begin
  ; /* Infinity padding for lower order patterns */
  $x[N+1:N+D-1] \longleftarrow \infty$;
  $\boldsymbol{p} \longleftarrow$ COMPUTE-BINARY-PATTERNS$[\boldsymbol{x}, D]$;
  ; /* adjustments for the frequency trick */
  for $k = 1, \ldots, D-2$ **do**
    for $l = D+1-k, \ldots, D$ **do**
      $\boldsymbol{p}[N-D+k, l] \longleftarrow 2^l$;
    end
  end
  sort the rows of $\boldsymbol{p}$ in lexicografical order ;
  for $d = D, D-1, \ldots, 1$ **do** // iterate over pattern orders
    for $m = D+1-d, D-d, \ldots, 1$ **do** // iterate over word lengths
      $\boldsymbol{h} \longleftarrow$ GET-PATTERN-FREQUENCIES$[\boldsymbol{p}, d, m]$;
      $\boldsymbol{H}[d, m] \longleftarrow \ln(N-d-m+2) - \frac{1}{N-d-m+2}\boldsymbol{h}^{\mathrm{T}} \ln[\boldsymbol{h}]$;
    end
    ; /* Transform the patterns to lower order words by Equation (4) */
    for $n = 0, \ldots, N-2$ **do**
      for $k = 1, \ldots, D-d+1$ **do**
        $\boldsymbol{p}[n, d+k-1] \longleftarrow (\boldsymbol{p}[n, d+k-1] \gg k) \ll k$
      end
    end
    resort the rows of $\boldsymbol{p}$ in lexicografical order ;
  end
end

Algorithm A2: COMPUTE-BINARY-PATTERNS

Input: Data array $\boldsymbol{x} \in \mathcal{X}^{N+1}$, pattern order D.
Output: Array of patterns $\boldsymbol{p} \in \mathbb{N}^{N+1-D \times D}$.
begin
  $N \longleftarrow length[\boldsymbol{x}] - 1$;
  for $n = 0, \ldots, N-D$ **do**
    $\boldsymbol{p}[n, D] = \sum_{l=0}^{D-1} \mathbf{1}_{x_{n+l} \succ x_{n+D}} 2^l$;
  end
  for $k = D-1, \ldots, 0$ **do**
    $\boldsymbol{p}[0, k] \longleftarrow \sum_{l=0}^{k-1} \mathbf{1}_{x_l \succ x_k} 2^l$;
    $\boldsymbol{p}[1:N-D, k] \longleftarrow \boldsymbol{p}[0:N-D-1, k+1] \gg 1$;
  end
end

Algorithm A3: GET-PATTERN-FREQUENCIES

Input: Array of patterns $\boldsymbol{p} \in \mathbb{N}^{N+1-D \times D}$, pattern order d, word length m.
Output: Array of frequencies $\boldsymbol{h}$.
begin
  $i \longleftarrow 0$;
  for $n = 0, \ldots, N - D$ **do**
    if $\|\boldsymbol{p}[n+1, 0:D-1] - \boldsymbol{p}[n, 0:D-1]\| > 0$ **then**
      $i \longleftarrow i + 1$;
    end
    $\boldsymbol{h}[i] \longleftarrow \boldsymbol{h}[i] + 1$;
  end
end

References

1. Alcaraz Martínez, R. Symbolic Entropy Analysis and Its Applications. *Entropy* **2018**, *20*, 568. [CrossRef]
2. Bandt, C.; Pompe, B. Permutation entropy—A natural complexity measure for time series. *Phys. Rev. E* **2002**, *88*, 174102. [CrossRef]
3. Bandt, C.; Keller, G.; Pompe, B. Entropy of interval maps via permutations. *Nonlinearity* **2002**, *15*, 1595–1602. [CrossRef]
4. Kurths, J.; Schwarz, U.; Witt, A.; Krampe, R.T.; Abel, M. Measures of complexity in signal analysis. *AIP Conf. Proc.* **1996**, *375*, 33–54.
5. Daw, C.S.; Finney, C.E.A.; Tracy, E.R. A review of symbolic analysis of experimental data. *Rev. Sci. Instrum.* **2003**, *74*, 915–930. [CrossRef]
6. Zanin, M.; Zunino, L.; Rosso, O.A.; Papo, D. Permutation entropy and its main biomedical and econophysics applications: A review. *Entropy* **2012**, *14*, 1553–1577. [CrossRef]
7. Amigó, J.M.; Keller, K.; Kurths, J. Recent progress in symbolic dynamics and permutation complexity. Ten years of permutation entropy. *Eur. Phys. J. Spec. Top.* **2013**, *222*, 241–598. [CrossRef]
8. Cuesta-Frau, D.; Murillo-Escobar, J.P.; Orrego, D.A.; Delgado-Trejos, E. Embedded Dimension and Time Series Length. Practical Influence on Permutation Entropy and Its Applications. *Entropy* **2019**, *21*, 385. [CrossRef]
9. Riedl, M.; Müller, A.; Wessel, N. Practical considerations of permutation entropy. *Eur. Phys. J. Spec. Top.* **2013**, *222*, 249–262. [CrossRef]
10. Gutjahr, T.; Keller, K. Equality of Kolmogorov-Sinai and permutation entropy for one-dimensional maps consisting of countably many monotone parts. *Discret. Contin. Dyn. Syst. A* **2019**, *39*, 4207–4224. [CrossRef]
11. Keller, K.; Maksymenko, S.; Stolz, I. Entropy determination based on the ordinal structure of a dynamical system. *Discrete Contin. Dyn. Syst. B* **2015**, *20*, 3507–3524. [CrossRef]
12. Unakafov, A.; Keller, K. Conditional entropy of ordinal patterns. *Phys. D* **2014**, *269*, 94–102. [CrossRef]
13. Keller, K.; Mangold, T.; Stolz, I.; Werner, J. Permutation Entropy: New Ideas and Challenges. *Entropy* **2017**, *19*, 134. [CrossRef]
14. Piek, A.B. Fast Ordinal Pattern and Permutation Entropy Computation. MATLAB Central File Exchange. Available online: https://www.mathworks.com/matlabcentral/fileexchange/71305-fast-ordinal-pattern-and-permutation-entropy-computation (accessed on 15 May 2019).
15. Bandt, C. Ordinal time series analysis. *Ecol. Model.* **2005**, *182*, 229–238. [CrossRef]
16. Keller, K.; Sinn, M.; Emonds, J. Time Series from the Ordinal Viewpoint. *Stochast. Dyn.* **2007**, *7*, 247–272. [CrossRef]
17. Amigó, J.M.; Monetti, R.; Aschenbrenner, T.; Bunk, W. Transcripts: An algebraic approach to coupled time series. *Chaos* **2012**, *22*, 013105. [CrossRef] [PubMed]
18. Amigó, J.M. *Permutation Complexity in Dynamical Systems. Ordinal Patterns, Permutation Entropy and all that*; Springer Series in Synergetics; Springer: Dordrecht, The Netherlands, 2010; ISBN 978-3-642-04083-2.
19. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; MIT Press: Cambridge, MA, USA, 2009; pp. 191–194, 304, 333, 484–504; ISBN 978-0-262-03384-8.

20. Howell, R.R. *On Asymptotic Notation with Multiple Variables*; Technical Report; Dept. of Computing and Information Sciences, Kansas State University: Manhattan, KS, USA, 2008. Available online: http://people.cs.ksu.edu/~rhowell/asymptotic.pdf (accessed on 13 March 2019).
21. Knuth, D.E. *The Art of Computer Programming Volume 3: Sorting and Searching*; Addison Wesley Longman Publishing Co., Inc.: Redwood City, CA, USA, 1998; ISBN 0-201-89685-0.
22. D'Errico, J. Variable Precision Integer Arithmetic. MATLAB Central File Exchange. Available online: https://www.mathworks.com/matlabcentral/fileexchange/22725-variable-precision-integer-arithmetic (accessed on 15 May 2019).
23. Unakafova, V. Fast Permutation Entropy. MATLAB Central File Exchange. Available online: https://www.mathworks.com/matlabcentral/fileexchange/44161-permutation-entropy--fast-algorithm (accessed on 15 May 2019).
24. Unakafova, V.; Keller, K. Efficiently Measuring Complexity on the Basis of Real-World Data. *Entropy* **2013**, *15*, 4392–4415. [CrossRef]
25. Ouyang, G. Permutation Entropy. MATLAB Central File Exchange. Available online: https://www.mathworks.com/matlabcentral/fileexchange/37289-permutation-entropy (accessed on 15 May 2019).
26. Müller, A. PETROPY—Permutation Entropy. MATLAB Central File Exchange. Available online: http://tocsy.pik-potsdam.de/petropy.php (accessed on 15 May 2019).
27. Walters, P. *An Introduction to Ergodic Theory*; Springer: New York, NY, USA, 2000; pp. 91–92, ISBN 978-0-387-95152-2.
28. Choe, G.H. *Computational Ergodic Theory*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 62–64, ISBN 978-3-540-23121-9.