Determinism, Complexity, and Predictability in Computer Performance
Quantifying structured complexity and its role in predictability: With applications to predicting computer performance.
Quantifying Predictability through Structural Complexity
Quantifying Predictability using Structural Complexity

Joshua Garland [*1], Ryan James [†1], and Elizabeth Bradley [‡1,2]

[1]Department of Computer Science, University of Colorado at Boulder, Colorado, USA
[2]Santa Fe Institute, New Mexico, USA

February 16, 2014

Abstract

[[Joshua: Needs to be rewritten to be more general and use computers as the application not the focus]] Computers are deterministic dynamical systems [1]. Among other things, that implies that one should be able to use deterministic forecast rules to predict aspects of their behavior. That statement is sometimes—but not always—true. The memory and processor loads of some simple programs are easy to predict, for example, but those of more-complex programs like `403.gcc` are not. The goal of this paper is to determine why that is the case. We conjecture that, in practice, complexity can effectively overwhelm the predictive power of deterministic forecast models. To explore that, we build models of a number of performance traces from different programs running on different Intel-based computers. We then calculate the *permutation entropy*—a temporal entropy metric that uses ordinal analysis—of those traces and correlate those values against the prediction success.

1 Introduction

BRAINSTORMING

[*]joshua.garland@colorado.edu
[†]ryan.james@colorado.edu
[‡]lizb@colorado.edu

- Complexity need not be hard to predict (can point at the simple predictions paper) [[move to introduction]]
- random walk for example is best predicted by guess what just happend[[move to introduction]]
- The kind of complexity present matters, i.e., that is whether the complexity is structured or not.[[use here and mention in intro]]
- `col_major` brings about the point nicely that some prediction strategies cannot utilize the processes internal information transfer method. That is a nonlinear internal information transfer system cannot be predicted effectively with a linear strategy. This gives a practitioner leverage on when to give up and when to keep working. [[use in this section as bridge to next section]]

Things to add to introduction

1. Different kinds of complexity exist in time series and this makes choosing prediction models difficult

   NOTE: RW and chaos are both complex. One is predictable and one is not.

2. Make an argument that Computer Performance is a great testing ground as it omits signals that completely cover the spectrum of complexity `col_major` ... `403.gcc`
3. When deterministic structure even complex structure exists that structure can be utilized for prediction.
4. For noisy real-valued time series distinguishing randomness (WN,RW) complexity from structured nonlinear / chaotic /high period / high dimensional etc complexity is (until now) very hard.

   for this provide predictions of `403.gcc` and `col_major` side by side and discuss "How can we tell if we did a bad job because the method is inadequate vs the signal being too complex. Lead this into is it possible to tell if there exists structure in a time series to know if we should find a better model or not. Maybe even having 4 predictions. top being ARIMA of the above signals and bottom being LMA of the above signals. Show that one improved and one did not. Is it that we used the wrong method to predict or is it that we simply can't predict the signal better than a random walk due to high levels of internal signal complexity.

5. Introduce the two main contributions of the paper which are outlined at the begining of the results section

Computers are among the most complex engineered artifacts in current use. Modern microprocessor chips contain multiple processing units and multi-layer memories, for instance, and they use complicated hardware/software strategies to move data and threads of computation across those resources. These features—along with all the others that go into the design of these chips—make the patterns of their processor loads and memory accesses highly complex and hard to predict. Accurate forecasts of these quantities, if one could construct them, could be used to improve computer design. If one could predict that a particular computational thread would be bogged down for the next 0.6 seconds waiting for data from main memory, for instance, one could save power by putting that thread on hold for that time period (e.g., by migrating it to a processing unit whose clock

speed is scaled back). Computer performance traces are, however, very complex. Even a simple "microkernel," like a three-line loop that repeatedly initializes a matrix in column-major order, can produce *chaotic* performance traces [1], as shown in Figure 2a, and chaos places fundamental limits on predictability.

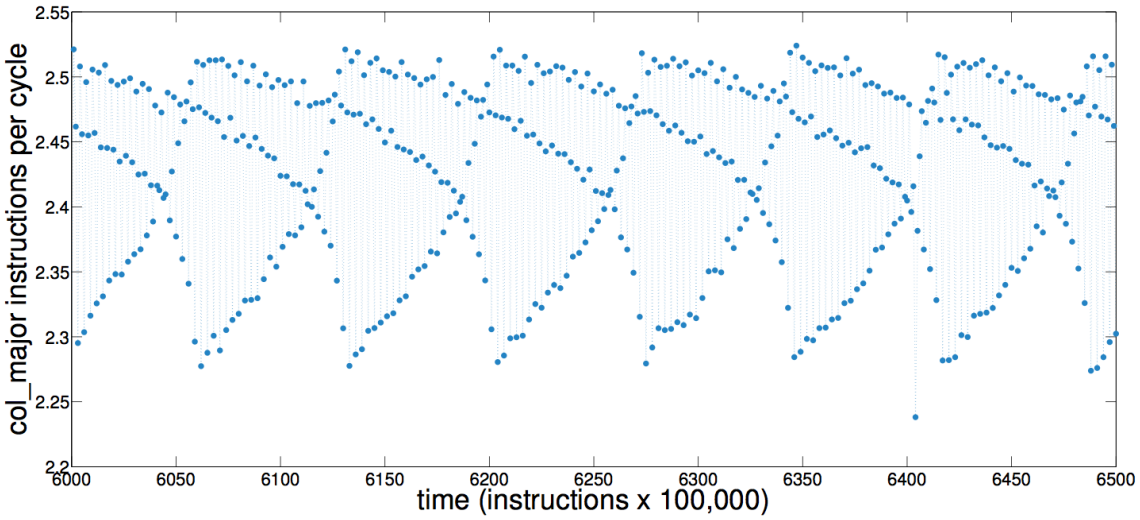
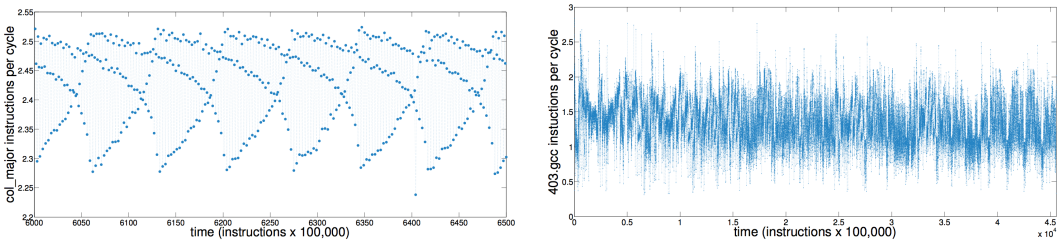


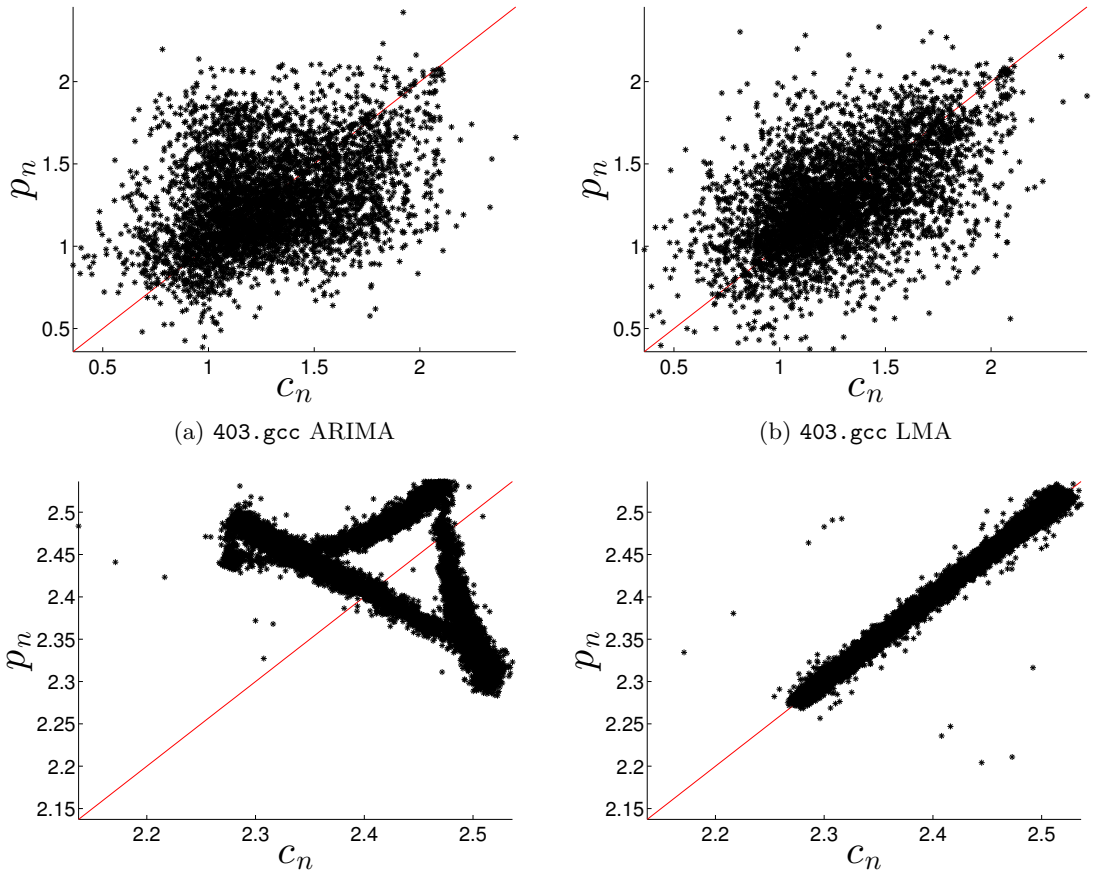
Figure 1: A small snippet of the instructions per cycle(ipc) of `col_major` , a three-line C program that repeatedly initializes a matrix in column-major order, running on an Intel i7®-based machine. Even this simple program exhibits chaotic performance dynamics.



(a) A short segment of the instructions executed per CPU clock cycle (IPC) during the execution of `col_major` . Each point is the IPC during a 100,000 cycle period. There is a great deal of periodicity in this time series.

(b) The full IPC time series during the execution of `403.gcc` . This time series has very little structure.

The computer systems community has applied a variety of prediction strategies to traces like this, most of which employ regression. An appealing alternative builds on the recently established fact that computers can be effectively modeled as deterministic nonlinear dynamical systems [1]. This result implies the existence of a deterministic forecast rule for those dynamics. In particular, one can use *delay-coordinate embedding* to reconstruct the underlying dynamics of computer performance,



(a) `403.gcc` ARIMA

(b) `403.gcc` LMA

(c) `col_major` ARIMA

(d) `col_major` LMA

Figure 3: For each of these, we plot the predicted value p_n against the correct value c_n. On this type of plot a perfect prediction lies exactly on the diagonal, that is the line $p_n = c_n$, e.g., 3d is a near perfect prediction where-as 3a is a very poor prediction.

then use the resulting model to forecast the future values of computer performance metrics such as memory or processor loads [2]. In the case of simple microkernels like the one that produced the trace in Figure 1, this deterministic modeling and forecast strategy works very well. In more-complicated programs, however, such as speech recognition software or compilers, this forecast strategy—as well as the traditional methods—break down quickly.

This paper is a first step in understanding when, why, and how deterministic forecast strategies fail when they are applied to deterministic systems. We focus here on the specific example of computer performance. We conjecture that the complexity of traces from these systems—which results from the inherent dimension, non-linearity, and non-stationarity of the dynamics, as well as from measurement issues like noise, aggregation, and finite data length—can make those deterministic signals *effectively* unpredictable. We argue that *permutation entropy* [3], a method for measuring the entropy of a real-valued-finite-length time series through ordinal analysis, is an effective way to explore that conjecture. We study four examples—two simple microkernels and two complex programs from the SPEC benchmark suite—running on different Intel-based machines. For each program, we calculate the permutation entropy of the processor load (instructions per cycle) and memory-use efficiency (cache-miss rates), then compare that to the prediction accuracy attainable for that trace using a simple deterministic model.

It is worth taking a moment to consider the theoretical possibility of this task. We are not attempting to predict the state of the CPU at an arbitrary point in the future — this, at least with perfect accuracy, would be tantamount to solving the halting problem. What we are attempting is to predict aspects or functions of the running of the CPU: instructions executed per second, cache misses per 100,000 instructions, and similar statistics. Prediction of these quantities at some finite time in the future, even with perfect accuracy, does not violate the Rice-Shapiro theorem.

2 Experimental Methods and Modelling

Section outline:

1. Experimental methods (how we collect the time series and what the times series are
2. Description of DCE and parameter estimation
3. Description of auto ARIMA

   (this should be limited and explain it is meant to be out of the box) point at the paper for auto arima for more details.
4. Description of the two naive methods (random walk and mean), make sure to explain that these methods are naive and simple but not necessarily bad.
5. Add a section talking about evaluation methods i.e., MASE, this text is currently written and just sitting at the beginning of the results.

2.1 Reconstructing hidden dynamics

Delay-coordinate embedding allows one to reconstruct a system's full state-space dynamics from a *single* scalar time-series measurement—provided that some conditions hold regarding that data. Specifically, if the underlying dynamics and the measurement function—the mapping from the

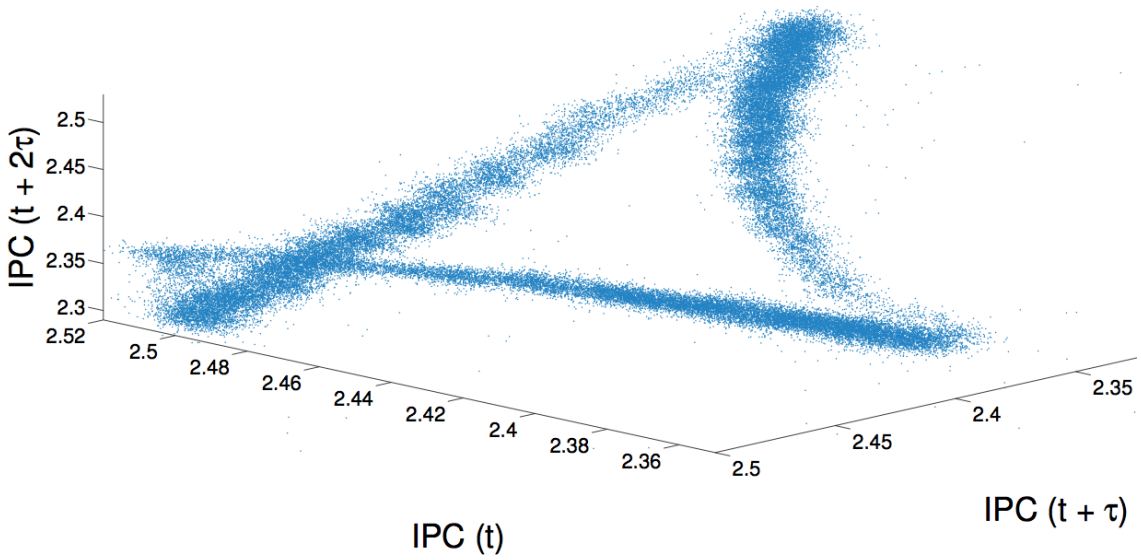


Figure 4: A 3D projection of a delay-coordinate embedding of the trace from Figure 1 with a delay (τ) of 100,000 instructions.

unknown state vector $\vec{X}$ to the scalar value x that one is measuring—are both smooth and generic, Takens [4] formally proves that the delay-coordinate map

$$F(\tau, m)(x) = ([x(t)\ x(t+\tau)\ \ldots\ x(t+m\tau)])$$

from a d-dimensional smooth compact manifold M to Re^{2d+1}, where t is time, is a diffeomorphism on M—in other words, that the reconstructed dynamics and the true (hidden) dynamics have the same topology.

This is an extremely powerful result: among other things, it means that one can build a formal model of the full system dynamics without measuring (or even knowing) every one of its state variables. This is the foundation of the modeling approach that is used in this paper. The first step in the process is to estimate values for the two free parameters in the delay-coordinate map: the delay τ and the dimension m. We follow standard procedures for this, choosing the first minimum in the average mutual information as an estimate of τ [5] and using the false-near(est) neighbor method of [6], with a threshold of 10%, to estimate m. A plot of the data from Figure 1, embedded following this procedure, is shown in Figure 4.

The coordinates of each point on this plot are differently delayed elements of the `col_major` L2 cache miss rate time series $y(t)$: that is, $y(t)$ on the first axis, $y(t+\tau)$ on the second, $y(t+2\tau)$ on the third, and so on. Structure in these kinds of plots—clearly visible in Figure 4—is an indication of determinism[1]. That structure can also be used to build a forecast model.

[1]A deeper analysis of Figure 4—as alluded to on the previous page—supports that diagnosis, confirming the presence of a chaotic attractor in these cache-miss dynamics, with largest Lyapunov exponent $\lambda_1 = 8000 \pm 200$ instructions, embedded in a 12-dimensional reconstruction space [1].

2.2 LMA: Using dynamics in forecasting

Given a nonlinear model of a deterministic dynamical system in the form of a delay-coordinate embedding like Figure 4, one can build deterministic forecast algorithms by capturing and exploiting the geometry of the embedding. Many techniques have been developed by the dynamical systems community for this purpose (e.g., [7, 8]). Perhaps the most straightforward is the "Lorenz method of analogues" (LMA), which is essentially nearest-neighbor prediction in the embedded state space [9]. Even this simple algorithm—which builds predictions by finding the nearest neighbor in the embedded space of the given point, then taking that neighbor's path as the forecast—works quite well on the trace in Figure 1, as shown in Figure 5. On the other hand, if we use the same approach

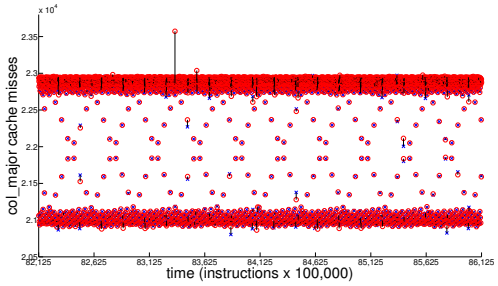


Figure 5: A forecast of the last 4,000 points of the signal in Figure 1 using an LMA-based strategy on the embedding in Figure 4. Red circles and blue ×s are the true and predicted values, respectively; vertical bars show where these values differ.

to forecast the processor load[2] of the `482.sphinx3` program from the SPEC cpu2006 benchmark suite, running on an Intel i7®-based machine, the prediction is far less accurate; see Figure 6.

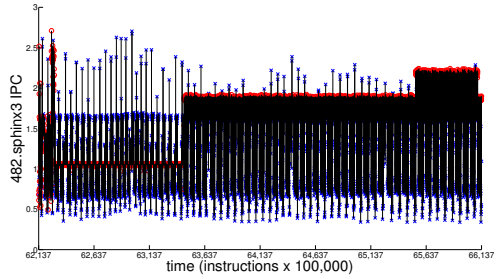


Figure 6: An LMA-based forecast of the last 4,000 points of a processor-load performance trace from the `482.sphinx3` benchmark. Red circles and blue ×s are the true and predicted values, respectively; vertical bars show where these values differ.

Table **??** presents detailed results about the prediction accuracy of this algorithm on four different examples: the `col_major` and `482.sphinx3` programs in Figures 5 and 6, as well as another simple microkernel that initializes the same matrix as `col_major`, but in row-major order, and

[2]Instructions per cycle, or IPC

another complex program (`403.gcc`) from the SPEC cpu2006 benchmark suite. Both microkernels were run on the Intel Core Duo® machine; both SPEC benchmarks were run on the Intel i7® machine. We calculated a figure of merit for each prediction as follows. We held back the last k elements[3] of the N points in each measured time series, built the forecast model by embedding the first $N-k$ points, used that embedding and the LMA method to predict the next k points, then computed the Root Mean Squared Error (RMSE) between the true and predicted signals:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{k}(c_i - \hat{p_i})^2}{k}}$$

To compare the success of predictions across signals with different units, we normalized RMSE as follows:

$$nRMSE = \frac{RMSE}{X_{max,obs} - X_{min,obs}}$$

The results in Table **??** show a clear distinction between the two microkernels, whose future behavior can be predicted effectively using this simple deterministic modeling strategy, and the more-complex SPEC benchmarks, for which this prediction strategy does not work nearly as well. This begs the question: If these traces all come from deterministic systems—computers—then why are they not equally predictable? Our conjecture is that the sheer complexity of the dynamics of the SPEC benchmarks running on the Intel i7® machine make them effectively impossible to predict.

2.3 Comparing Prediction Accuracy

In order to analyze correctness of each prediction we split each time series into two pieces: the first 90% referred to as the "learning" or "training" signal, $\{X_{i,obs}\}_{i=1}^{n}$ and the last 10% known as the "test" or "correct" signal $\{c_j\}_{j=n+1}^{k+n+1}$. The learning signal is used to train an initial model (e.g., LMA or ARIMA) as described in Section 2. The test signal is used both to assess the models forecasting accuracy and for any refitting that may be necessary. In particular, we perform k 1-step predictions, after each 1-step prediction[4] we append the training signal with the next point in the correct signal c_j, refit the model taking into account the new system measurement and perform another prediction. This is repeated k times to obtain $\{p_j\}_{j=n+1}^{k+n+1}$.

As a figure of merit we calculate the Mean Absolute Squared Error (MASE)[15] between the true and predicted signals:

$$MASE = \sum_{j=n+1}^{k+n+1} \frac{|c_j - p_j|}{\frac{k}{n-1}\sum_{i=2}^{n}|X_{i,obs} - X_{i-1,obs}|}$$

The scaling term for MASE:

$$\frac{1}{n-1}\sum_{i=2}^{n}|X_{i,obs} - X_{i-1,obs}|$$

is the average in-sample forecast error for a random walk prediction ($p_i = X_{i-1,obs}$). This error method was introduced in [15] as a "generally applicable measurement of forecast accuracy without

[3]Several different prediction horizons were analyzed in our experiment; the results reported in this paper are for k=4000

[4]We would like to note that this rebuilding occurs due to a problem with ARIMA models converging to a mean prediction if too long of a prediction horizon is used, this is not a handicap of either LMA or naïve.

the problems seen in the other measurements." The major advantage of MASE is that it allows fair comparison across methods, prediction horizons and varying signal scales. When a forecast results in a $MASE < 1$ this means that the prediction method gave, on average, smaller errors than the 1-step errors from the in-sample random walk forecast strategy. Analogously, $MASE > 1$ means that the prediction method did worse, on average than the 1-step errors for the in-sample random walk forecast strategy. In Table 1 we provide the distribution [[Joshua: Ryan, Is this the right word? we give mean ± std. dev but some have very skewed right tails]] of MASEs for each of the 8 signals and 3 prediction strategies, these are averaged over 15 runs of each type (signal + method). For comparison Table 1 also has the distribution of weighted permutation entropies for word lengths of $l = 5$ and $l = 6$.

Table 1: MASE distributions for 1-step predictions at a 10% prediction horizon over 15 runs for each signal and average wpe at word length 5 and 6 for each signal.[[Joshua:Maybe delete $l = 5$ as we don't use it in any figures, also if we leave this (before the next section we need to point forward to the next section for PE or remove it from this table]]

	MASE LMA	MASE ARIMA	MASE naïve	$l = 5$	$l = 6$
`403.gcc`	1.5296 ± 0.0214	1.8366 ± 0.0157	1.7970 ± 0.0095	0.9510 ± 0.0011	0.9430 ± 0.0013
`col_major`	0.0500 ± 0.0018	0.5989 ± 0.2114	0.5707 ± 0.0017	0.5636 ± 0.0031	0.5131 ± 0.0034
`dgesdd` Reg. 1	0.8273 ± 0.0755	0.7141 ± 0.0745	2.6763 ± 4.3282	0.9761 ± 0.0084	0.9572 ± 0.0156
`dgesdd` Reg. 2	1.2789 ± 0.0196	2.1626 ± 0.0265	3.0543 ± 0.0404	0.8760 ± 0.0052	0.8464 ± 0.0044
`dgesdd` Reg. 3	0.6192 ± 0.0209	0.7129 ± 0.0096	31.3857 ± 0.2820	0.7768 ± 0.0073	0.7157 ± 0.0056
`dgesdd` Reg. 4	0.7789 ± 0.0358	0.9787 ± 0.0321	2.6613 ± 0.0739	0.9073 ± 0.0080	0.8246 ± 0.0077
`dgesdd` Reg. 5	0.7177 ± 0.0483	2.3700 ± 0.0505	20.8703 ± 0.1915	0.7333 ± 0.0076	0.6776 ± 0.0068
`dgesdd` Reg. 6	0.7393 ± 0.0682	1.4379 ± 0.0609	2.1967 ± 0.0830	0.8101 ± 0.0135	0.7475 ± 0.0106

3 Measuring Structural Complexity

THINGS TO DO

1. rewrite for consistency in notation. For example, some parts of the paper call word length l and some call it n.
2. make sure it is clear and we justify that detecting and characterizing structural complexity is hard for real-valued noisy time series.
3. Quantifying structured and unstructured complexity is nontrivial in the case of real-valued noisy time series but WPE does this. [[talk about again here but justify in information theory section]]
4. justify and explain the regime split in `dgesdd`

For the purposes of this paper, one can view entropy as a measure of complexity and predictability in a time series. A high-entropy time series is almost completely unpredictable—and conversely. This can be made more rigorous: Pesin's relation [10] states that in chaotic dynamical systems, the

Shannon entropy rate is equal to the sum of the positive Lyapunov exponents, λ_i. The Lyapunov exponents directly quantify the rate at which nearby states of the system will diverge with time: $|\Delta x(t)| \approx e^{\lambda t} |\Delta x(0)|$. The faster the divergence, the more difficult prediction becomes.

Utilizing entropy as a measure of temporal complexity is by no means a new idea [11, 12]. Its effective usage requires categorical data: $x_t \in \mathcal{S}$ for some finite or countably infinite *alphabet* $\mathcal{S}$, whereas data taken from real-world systems is effectively real-valued. To get around this, one must discretize the data— typically achieved by binning. Unfortunately, this is rarely a good solution to the problem, as the binning of the values introduces an additional dynamic on top of the intrinsic dynamics whose entropy is desired. The field of symbolic dynamics studies how to discretize a time series in such a way that the intrinsic behavior is not perverted, but these methods are fragile in the face of noise and require further understanding of the underlying system, which defeats the purpose of measuring the entropy in the first place.

Bandt and Pompe introduced the *permutation entropy* (PE) as a "natural complexity measure for time series" [3]. Permutation entropy employs a method of discretizing real-valued time series that follows the intrinsic behavior of the system under examination. Rather than looking at the statistics of sequences of values, as is done when computing the Shannon entropy, permutation entropy looks at the statistics of the *orderings* of sequences of values using ordinal analysis. Ordinal analysis of a time series is the process of mapping successive time-ordered elements of a time series to their value-ordered permutation of the same size. By way of example, if $(x_1, x_2, x_3) = (9, 1, 7)$ then its *ordinal pattern*, $\phi(x_1, x_2, x_3)$, is 231 since $x_2 \leq x_3 \leq x_1$. This method has many features; among other things, it is generally robust to observational noise and requires no knowledge of the underlying mechanisms.

Definition (Permutation Entropy)**.** *Given a time series* $\{x_t\}_{t=1,\ldots,T}$*. Define* $\mathcal{S}_n$ *as all* $n!$ *permutations* π *of order* n*. For each* $\pi \in \mathcal{S}_n$ *we determine the relative frequency of that permutation occurring in* $\{x_t\}_{t=1,\ldots,T}$*:*

$$p(\pi) = \frac{|\{t | t \leq T - n, \phi(x_{t+1}, \ldots, x_{t+n}) = \pi\}|}{T - n + 1}$$

Where $|\cdot|$ *is set cardinality. The* permutation entropy *of order* $n \geq 2$ *is defined as*

$$H(n) = -\sum_{\pi \in \mathcal{S}_n} p(\pi) \log_2 p(\pi)$$

Notice that $0 \leq H(n) \leq \log_2(n!)$ [3]. With this in mind, it is common in the literature to normalize permutation entropy as follows: $\frac{H(n)}{\log_2(n!)}$. With this convention, "low" entropy is close to 0 and "high" entropy is close to 1. Finally, it should be noted that the permutation entropy has been shown to be identical to the Shannon entropy for many large classes of systems [13].

Here we will be utilizing a variation of the permutation entropy, the *weighted permutation entropy* (WPE) [14]. The weighted permutation entropy attempts to correct for observational noise which is larger than some trends in the data, but smaller than the larger scale features — for example, a signal that switches between two fixed points with noise about those fixed points. The weighted permutation entropy would be dominated by the switching rather than by the stochastic fluctuation. To accomplish this, the *weight* of a permutation is taken into account:

$$w(x_{t+1:t+n}) = \frac{1}{n} \sum_{x_i \in \{x_{t+1:t+n}\}} (x_i - \bar{x}_{t+1:t+n})^2$$

where $x_{t+1:t+n}$ is a sequence of values $x_{t+1}, \ldots, x_{t+n}$, and $\bar{x}_{t+1:t+n}$ is the arithmetic mean of those values.

The weighted probability of a permutation is then:

$$p_w(\pi) = \frac{\sum_{t \leq T-n} w(x_{t+1:t+n}) \cdot \delta(\phi(x_{t:t+n}), \pi)}{\sum_{t \leq T-n} w(x_{t+1:t+n})}$$

where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise. Effectively, this weighted probability enhances permutations involved in "large" features and demotes permutations which are small in amplitude relative to the features of the time series. The weighted permutation entropy is then:

$$H_w(n) = -\sum_{\pi \in \mathcal{S}_n} p_w(\pi) \log_2 p_w(\pi),$$

which can also be normalized by dividing by $\log_2(n!)$, and will be in all the results of this paper.

In practice, calculating permutation entropy and weighted permutation entropy involves choosing a good value for the word length n. The primary consideration is that the value be large enough that forbidden ordinals are discovered, yet small enough that reasonable statistics over the ordinals are gathered: e.g.:

$$n = \operatorname*{argmax}_{\ell} \{T \gtrapprox 100\ell!\},$$

assuming an average of 100 counts per ordinal is sufficient. In the literature, $3 \leq n \leq 6$ is a standard choice — generally without any formal justification. In theory, the permutation entropy should reach an asymptote with increasing n, but that requires an arbitrarily long time series. In practice, what one should do is calculate the *persistent* permutation entropy by increasing n until the result converges, but data length issues can intrude before that convergence is reached.

The weighted permutation entropy for the `SVD` program is given in Fig. 9. To generate this image a window of 5,000 values slid over the time series. Within each of those windows, the statistics over words of length 4 are computed and the WPE is calculated. The gray bands denote regions where the 5,000 value window overlapped visually-distinct regimes. It can be seen that the behaviors of the weighted permutation entropy vary between regimes. [[I think here it would be good to add a paragraph explaining the windowed WPE was used for regime choices on SVD...emphasizing that over a time series permutation entropy fluctuates illustrating within a single time series different levels of complexity and predictability exist. Maybe point at some of the predicting predictability papers.]]

4 Validation of Primary Findings

In this section we validate the two key primary findings of this work which we introduced in Section 1:

1. The existence of predictable structure in noisy real-valued time series is quantifiable by WPE and as a result WPE is correlated with prediction accuracy (MASE) of an ideal predictor.

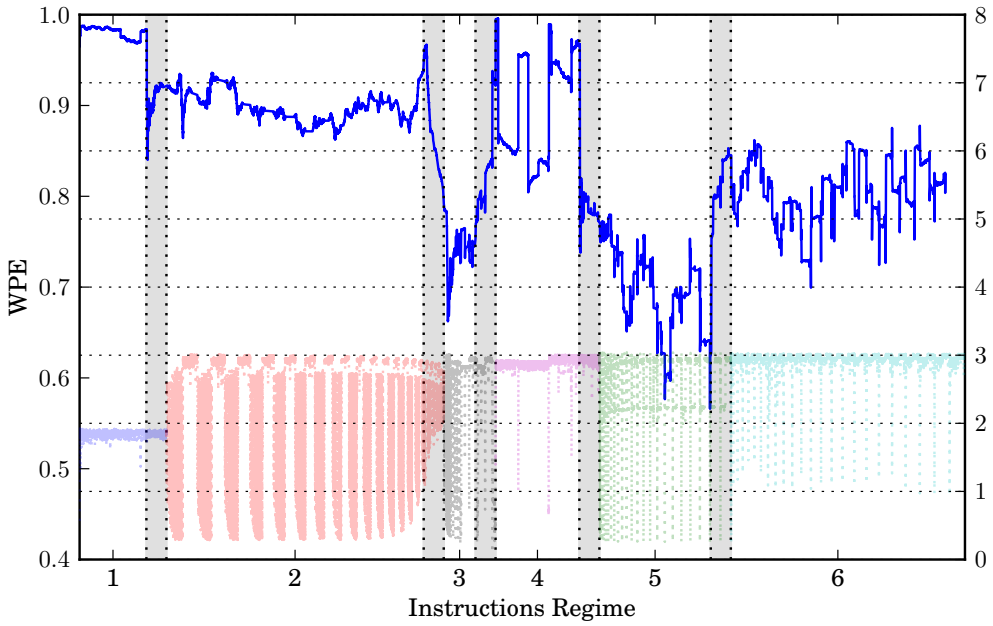


Figure 7: [Joshua: I think adding the colored SVD trace to this would be good or putting it above this figure but need to figure how to line them up properly. Also we need to label that the numbers on the bottom of WPE are regimes not instructions...]]The weighted permutation entropy of one run of SVD. The gray bands are regions where the window overlaps regimes. The window size used is 5,000 $\times$ 100,000 instructions and the word length is 4.[[Joshua:why is $l = 4$ and not 5,6 like the rest of the paper?]]For reference the instructions per cycle of `dgesdd` are plotted as a ghost behind this plot. Each color on the ghosted time series corresonds to the different regimes as selected by rapid shifts in WPE. From left to right each change in color represents a change in regime for 6 regimes in total.

2. The way structure/information/complexity is processed internally by a given process plays a crucial role in predictability.

As we discussed in Section 3 distinguishing structured from unstructured complexity in the case of real-valued time series is non trivial, i.e., distinguishing when a complex signal omits predictable structure and when the signal is effectively random is not a trivial task. As described in Section 1, for a practitioner this can be frustrating because it can be nearly impossible to find the source of faulty predictions: Is it simply that we need to use a more ideal (possibly more advanced) predictor or is it the case that the time series is simply so complex that using a simple (yet inconsistent) forecast strategy such as random walk is the best we can do. Fortunately, weighted permutation entropy (WPE) allows us to make this distinction for noisy real-valued time series.

In Figure 10 we plot the best prediction (i.e., the lowest MASE over all 3 methods[5] over all 15 runs of that program) for each of the 8 programs under investigation. What we see here is that the

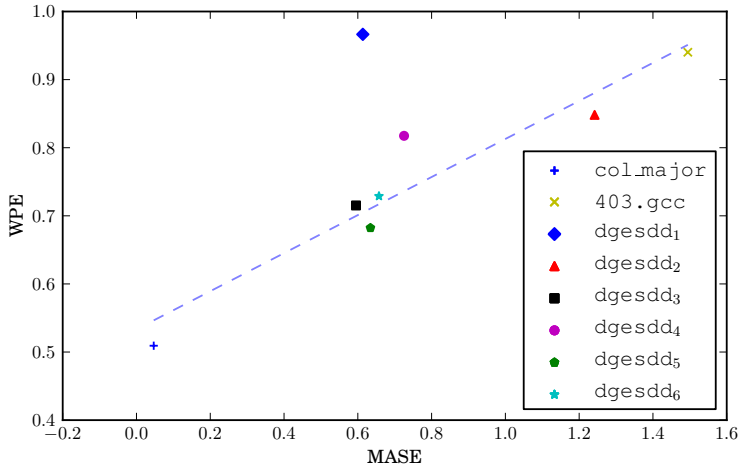


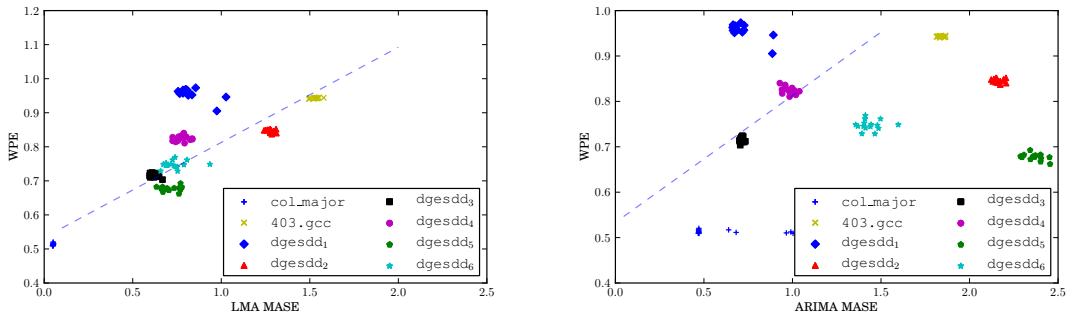
Figure 8: The best MASE among all runs and prediction methods vs weighted permutation entropy. For each of these, the word length used is 6. The dashed line is a least-squares linear fit of all the points except for $\texttt{SVD}_1$ which we have excluded for reasons explained in the text.

relationship between prediction accuracy (MASE) and the weighted permutation entropy is much as we conjectured: The existence of predictable structure in noisy real-valued time series as quantified by low to moderate WPE is correlated with prediction accuracy (MASE) of the "best"predictor[6]. We will not further validate this claim through some examples.

To further validate this finding we present an in-depth analysis of three examples (`col_major` , $\texttt{dgesdd}_5$ and `403.gcc`) which nicely cover the range of complexity from low to high respectively. In Figures 11 and 11b we plot the MASE of LMA and ARIMA (respectively) against the weighted permutation entropy of each of the 15 runs of all 8 programs.

[5]LMA,ARIMA,naïve

[6]We do not claim to have found the ideal predictor for each signal, simply the best predictor over what we used; Which we believe to be a fair sampling of standard prediction techniques for this type of signal.



(a) The MASE of LMA vs weighted permutation entropy.

(b) The MASE of ARIMA vs weighted permutation entropy.

Figure 9: For each of these, the word length used is 6. The MASE values for LMA against ARIMA. The dashed line is the identity, delineating the traces for which either LMA or ARIMA performed better. All traces except those from `dgesdd` $_1$ lie above the line, indicating that LMA is better suited prediction method for the computer traces considered.

We will start with the lowest complexity program: `col_major` . Conceptually this program is very simple but due to computer design choices this program can result in highly complex even chaotic time series traces, see [1]. If we forecast this signal using an out-of-the-box linear forecaster (ARIMA) or an out-of-the-box constant predictor we receive MASE (0.59 ± 0.21 and 0.57 ± 0.001 respectively) which suggest that on average we can only predict this signal twice as well as the simple random walk prediction. This would suggest that there is some predictive structure present but not that much. However, if we quantify the predictive structure present in this signal using WPE we see that this signal is highly structured with a WPE of (0.51 ± 0.003.) The existence of structured which we quantified with WPE implies that an ideal predictor capable of using this signals information should be able to predict this signal much better than the random walk due to the high amount of predictive structure or equivalently the low complexity. This is in fact the case. `col_major` is a complex signal, i.e., chaotic, but it is complex in a structured way. Using a prediction technique that can process nonlinear information yields forecasts with a MASE of 0.05 ± 0.001, this is a prediction 20 times more accurate than a random walk forecast. The existence of predictive structure as quantified by WPE could be precisely the information needed to continue searching for an ideal predictor like LMA.

Maybe even more convincing than a low complexity signal like `col_major` is a moderately complex signal like `dgesdd`$_5$. This signal omits a WPE of 0.677 ± 0.006, this is a significantly higher complexity than `col_major` but still implies the existence of a great deal of predictive structure. However, this increase in complexity is enough to really throw off the constant and linear prediction strategies. The MASE scores for ARIMA and naïve imply 2.5 and 20 times worse prediction than using a simple random walk forecast. This may be discouraging to someone using an out-of-the-box predictor that the signal is too complex to predict. However by *knowing* that there exists predictable in this signal we can try to use a more advanced technique, in particular LMA. If we do this we get forecasts which score 1.5 times better than the random walk. Showing that this usable structure (although hidden to the linear and constant method) was quantifiable by WPE.

Finally, we consider the most complex program: `403.gcc` . The complexity of `403.gcc` is 0.94 ± 0.001. This implies that almost no predictive structure exists in this structure. Recall that a WPE ≈ 1 are similar in complexity to white noise or a random walk process as they do not transmit any information from past to future. This would mean that the WPE of > 0.95 would suggest little to no predictive structure exists and that the signal is simply too complex. In this case it should be safe to assume that a random walk forecast would be (on-average) superior to forecast methods which required structure or information to intelligently predict the next time step. This is in fact what we see, with each prediction scheme we used we see a MASE which implies (on-average) 1.5-1.8 times worse predictions that performing a simple random walk forecast.

There is a single outlier in this study which is $\texttt{dgesdd}_1$. This signal has a WPE of 0.95 ± 0.02 and both LMA and ARIMA do slightly better than random walk (0.82 ± 0.08 and 0.7141 ± 0.07 respectively). The naïve method did significantly worse than random walk and was highly inconsistent with an average MASE of 2.67 but a standard deviation of 4.33 due to the massive right tail this distribution exhibited in reaction to wildly inconsistent forecasts. We do not believe this outlier is really a problem but it does bring about a potential weaknesses of this method. MASE is scaled by a random walk forecast which can be very inconsistent in the case of drastic signal changes which occur on a regular basis, e.g., a signal which oscillates from the top to the bottom of the signals scale at every time click. In this case a random walk forecast would have error with magnitude of the scale of the signal at every forecast, since guessing the last step will be exactly out of phase with the new forecast. A signal like this would normally cause a very low WPE much like `col_major` and a very high Random Walk error resulting in low MASE for methods that can take into account this structure like ARIMA or LMA.

To understand why this causes $\texttt{dgesdd}_1$ to be an outlier we examine Figure 12 in more detail. This is a small snippet of the time series of $\texttt{dgesdd}_1$ to illustrate the strucutre of this signal. We provide a small snippet as opposed to the full time series simply for visual clarity. As can be seen

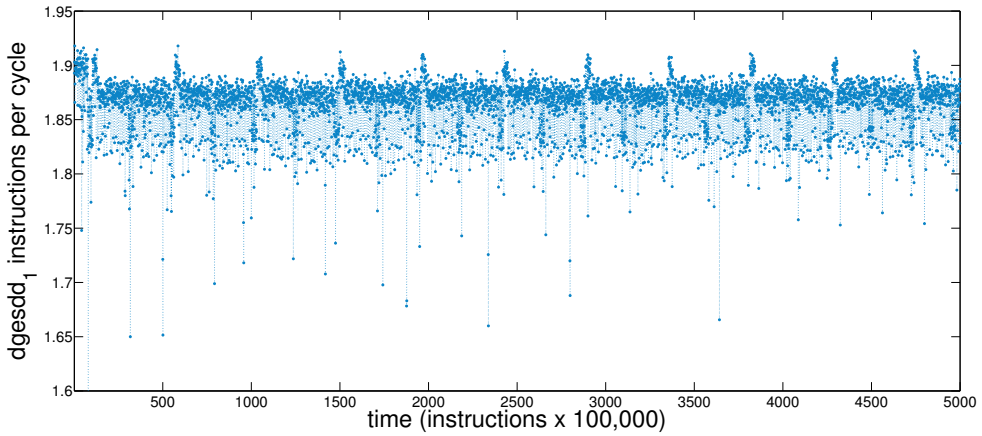


Figure 10: For each of these, the word length used is 6. The MASE values for LMA against ARIMA. The dashed line is the identity, delineating the traces for which either LMA or ARIMA performed better. All traces except those from $\texttt{dgesdd}_1$ lie above the line, indicating that LMA is better suited prediction method for the computer traces considered.

in Figure 12, `dgesdd`$_1$ has a highly unstructured small band on top, which is approximately 0.05 instructions per cycle wide. Approximately 80% of the signal exists within this band of noise, this causes the WPE to be driven up as many words are being omitted by this band of the signal. The remaining 20% of the signal are drops in instructions per cycle away from this band, ranging in size from 0.1 to 0.6 instructions per cycle[7]. These drastic drops seem to happen every 5-7 points but the drop locations are inconsistent as well as the drops magnitude (ranging from 0.1-0.6 instructions per cycle). As discussed in Section **??** drops of this fashion are the bane of random walk forecasting. At every one of these drops a random walk predictor would produce two forecasts with an error 2-12 times the magnitude of the smaller band, i.e., about 40% of the time the random walk forecast would predict with error 2-12 times the magnitude of the scale exhibited by ≈80% of the signal. This inconsistency in random walk would cause cause random walk error to be high and as a result MASE scores to be artificially low. The combination of a band of noise with a high number of random drops is a weakness of this method which is present in `dgesdd`$_1$ and as a result makes this signal an outlier.

[[Joshua: I am not sure if this paragraph is strong enough or even worth having]]These results support that time series with low to moderate complexity ($0 \leq WPE \leq 0.85$) can be predicted more efficiently than a naïve random walk *and* that complexity can be qualitatively measured for a real-valued noisy time series using WPE. This will allow practitioners to stop spinning their wheels in the case of signals who are simply better predicted with a simple strategy like random walk. The analysis of these results illuminates an interesting point: The way structure, information and complexity are processed by a generating process plays a crucial role in the success of a given prediction scheme.

Comparing Figures 11a and 11b also elucidates another key finding: usable and quantifiable predictive structure can be present in a time series without a prediction scheme being able to utilize it. In particular, information may be transferred from past to future through the present but because of the mechanism the underlying process uses to process that information (e.g., linear or nonlinear) certain prediction strategies may be blind to or not be able to efficiently utilize this information. For example, consider `col_major` , programs like this have been shown to exhibit deterministic chaos [1]. If this were the case with `col_major` , an out-of-the-box linear method like ARIMA would simply be ill-equipped to model and utilize the kind of structure present, as is evident in Figure 11b. In contrast, a nonlinear predictor like LMA which is built to handle deterministic chaos, can interpret and utilize this type of structure just fine. We believe that many of the shifts in forecast accuracy for low-to-moderate WPE programs between ARIMA and LMA is precisely happening for this reason: Just because there is forward information transfer, does not mean that an arbitrary predictor can interpret or utilize it, but luckily WPE can tell us when this structure is present as shown in Figure 11.

Other noteworthy features of the LMA and ARIMA results are the cluster locations and distributions. The WPE values of each run of any particular program tend to have little variance, leading to the clusters in Figures 11b and 11a to be fairly constrained in the y direction. For most traces, the LMA and ARIMA variance is low as well, resulting is small, tight clusters. The ARIMA MASE values of the `col_major` traces, however, have a large variance resulting in the spread seen in Figure 11b. Not only are the MASE values of that cluster bad, in that other predictors vastly outperform it, but they are inconsistent. Furthermore, since LMA can predict nonlinear behavior while ARIMA cannot, we see that the clusters in Figure 11b are mostly further to the right than those in Figure 11a.

[7]Some of the large drops are not shown for the sake of clarity of the smaller scale

5 Conclusions & Future Work

1. This need to be rewritten to conclude and address this paper...

The results presented here suggest that permutation entropy—a ordinal calculation of forward information transfer in a time series—is an effective metric for predictability of computer performance traces. Experimentally, traces with a persistent PE $\gtrapprox 0.97$ have a natural level of complexity that may overshadow the inherent determinism in the system dynamics, whereas traces with PE $\lessapprox 0.7$ seem to be highly predictable (viz., at least an order of magnitude improvement in nRMSPE).Further, the persistent WPE values of 0.5– 0.6 for the `col_major` trace are consistent with dynamical chaos, further corroborating the results of [1].

If information is the limit, then gathering and using more information is an obvious next step. There is an equally obvious tension here between data length and prediction speed: a forecast that requires half a second to compute is not useful for the purposes of real-time control of a computer system with a MHz clock rate. Another alternative is to sample several system variables simultaneously and build multivariate delay-coordinate embeddings. Existing approaches to that are computationally prohibitive [17]. We are working on alternative methods that sidestep that complexity.

Acknowledgment

This work was partially supported by NSF grant #CMMI-1245947 and ARO grant #W911NF-12-1-0288.

References

[1] T. Myktowicz, A. Diwan, and E. Bradley. Computers are dynamical systems. *Chaos*, 19:033124, 2009. doi:10.1063/1.3187791.

[2] J. Garland and E. Bradley. Predicting computer performance dynamics. In *Proceedings of the 10th International Conference on Advances in Intelligent Data Analysis X*, pages 173–184, Porto, Portugal, 2011.

[3] C. Bandt and B. Pompe. Permutation entropy: A natural complexity measure for time series. *Phys Rev Lett*, 88(17):174102, 2002.

[4] F. Takens. Detecting strange attractors in fluid turbulence. In D. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence*, pages 366–381. Springer, Berlin, 1981.

[5] A. Fraser and H. Swinney. Independent coordinates for strange attractors from mutual information. *Physical Review A*, 33(2):1134–1140, 1986.

[6] M. B. Kennel, R. Brown, and H. D. I. Abarbanel. Determining minimum embedding dimension using a geometrical construction. *Physical Review A*, 45:3403–3411, 1992.

[7] M. Casdagli and S. Eubank, editors. *Nonlinear Modeling and Forecasting*. Addison Wesley, 1992.

[8] A. Weigend and N. Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Santa Fe Institute, 1993.

[9] E. N. Lorenz. Atmospheric predictability as revealed by naturally occurring analogues. *Journal of the Atmospheric Sciences*, 26:636–646, 1969.

[10] Ya B Pesin. Characteristic Lyapunov exponents and smooth ergodic theory. *Russian Mathematical Surveys*, 32(4):55, 1977.

[11] C. E. Shannon. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:50–64, 1951.

[12] RN Mantegna, SV Buldyrev, AL Goldberger, S. Havlin, CK Peng, M. Simons, and HE Stanley. Linguistic features of noncoding DNA sequences. *Physical review letters*, 73(23):3169–3172, 1994.

[13] J. Amigó. *Permutation Complexity in Dynamical Systems: Ordinal Patterns, Permutation Entropy and All That*. Springer, 2012.

[14] Bilal Fadlallah, Badong Chen, Andreas Keil, and José Príncipe. Weighted-permutation entropy: A complexity measure for time series incorporating amplitude information. *Physical Review E*, 87(2):022911, 2013.

[15] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, pages 679–688, 2006.

[16] R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(3):1–22, 7 2008.

[17] Liangyue Cao, Alistair Mees, and Kevin Judd. Dynamics from multivariate time series. *Physica D*, 121:75–88, 1998.