Determinism, Complexity, and Predictability in Computer Performance

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 10, 2014

Abstract

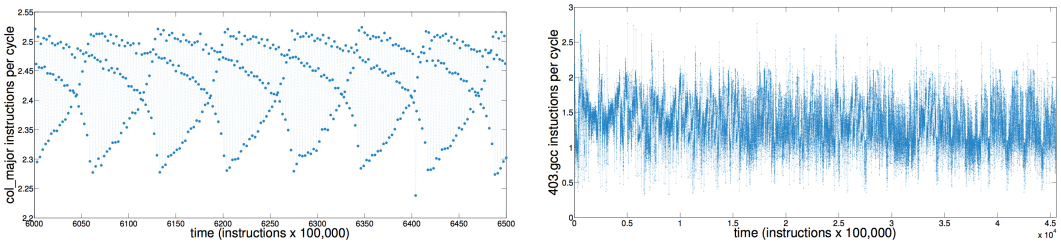
Computers are deterministic dynamical systems [1]. Among other things, that implies that one should be able to use deterministic forecast rules to predict aspects of their behavior. That statement is sometimes—but not always—true. The memory and processor loads of some simple programs are easy to predict, for example, but those of more-complex programs like `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

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

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

produce *chaotic* performance traces [1], as shown in Figure 1a, and chaos places fundamental limits on predictability.



(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, 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 **??**, 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, nonlinearity, and nonstationarity 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 Modeling Computer Performance

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 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 **??**, embedded following this procedure, is shown in Figure **??**.

[[Add a plot of an embedding...]]

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 **??**—is an indication of determinism[1]. That structure can also be used to build a forecast model.

Given a nonlinear model of a deterministic dynamical system in the form of a delay-coordinate embedding like Figure **??**, 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 **??**, as shown in Figure 2. On the other hand, if we use the same approach 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 3.

Table **??** presents detailed results about the prediction accuracy of this algorithm on four different examples: the `col_major` and `482.sphinx3` programs in Figures 2 and 3, as well as another simple microkernel that initializes the same matrix as `col_major`, but in row-major order, and 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

[1] A deeper analysis of Figure **??**—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] Instructions per cycle, or IPC

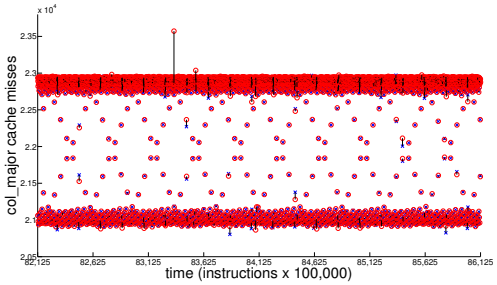


Figure 2: A forecast of the last 4,000 points of the signal in Figure **??** using an LMA-based strategy on the embedding in Figure **??**. Red circles and blue ×s are the true and predicted values, respectively; vertical bars show where these values differ.

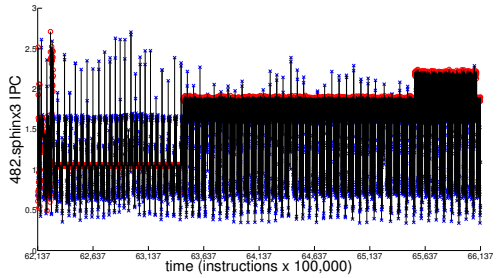


Figure 3: An LMA-based forecast of the last 4,000 points of a processor-load performance trace from the `482.sphinx3` benchmark. Red circles and blue ×s are the true and predicted values, respectively; vertical bars show where these values differ.

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.

3 Measuring Complexity

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;

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

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* $\{x_t\}_{t=1,\dots,T}$*. 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* $\{x_t\}_{t=1,\dots,T}$*:*

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

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 $x_{t+1}, \dots, x_{t+n}$, 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.:

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

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. 4. 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 differnet levels of complexity and predictability exist. Maybe point at some of the predicting predictability papers.]]

4 Results

In order to analyze correctness of each prediction we split each time series into two pieces: the first 90% refereed to as the "learning/training" signal, $\{X_{i,obs}\}_{i=1}^{i=n}$ and the last 10% known as the "test/correct" signal $\{c_j\}_{j=n+1}^{k+n+1}$. The learning signal is used to build the initial models (e.g., LMA, ARIMA, naïve) as described in Section 2. The test signal is used both to test the forecasts and to refit the models after forecasting. In particular, using the models we perform k 1-step predictions $\{p_j\}_{j=n+1}^{k+n+1}$, after each prediction we append the training signal with the the next point in the correct signal. 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.

In order to compare the resulting forecasts we calculate the Mean Absolute Squared Error (MASE)[15] between the true and predicted signals:

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

The scaling term here:

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

is the average in-sample forecast error for a random walk prediction ($p_j = c_{j-1}$). This error method was introduced in [15] as a "generally applicable measurement of forecast accuracy without the problems seen in the other measurements." The major advantage of MASE is that it allows fair

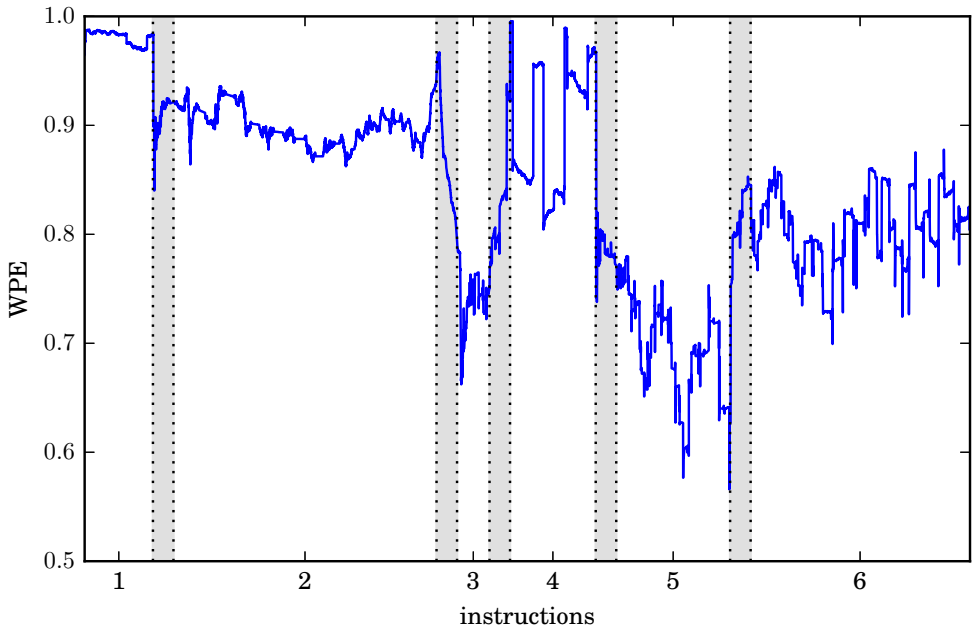


Figure 4: [[Joshua: I think adding the colored SVD trace to this would be good or putting it above this figure]]The weighted permutation entropy of one run of SVD. The gray bands are regions where the window overlaps regimes. The window size used is $5,000 \times 100,000$ instructions and the word length is 4.

comparison across methods, prediction horizons and varying signal scales. A $MASE < 1$ means that the prediction method gives, on average, smaller errors than the 1-step errors from the random walk forecast strategy. Analogously, a $MASE > 1$ means that the prediction method did worse, on average than the 1-step errors from the random walk forecast strategy. In Table 1 we provide the distribution of MASEs for 15 runs of each of the 8 signals across the 3 prediction forecast methods. For comparison Table 1 also has weighted permutation

Table 1: Average MASE 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.

	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

1. First paragraph: WPE is a good measure of predictability. Figure: best athlete MASE vs. its WPE.

   As can be seen in Table 1 the relationship between prediction accuracy and the weighted permutation entropy (WPE) is much as we conjectured: performance traces with high WPE are indeed harder to predict using the forecasting models described in Section 2. Figure 5 demonstrates this primary finding, with the exception of one outlier whose behavior we can explain. In Figure 5 we plot the best prediction (i.e., the lowest MASE over methods on that signal) for each of the 8 signals. We find that the relationship between the two is roughly linear. The single outlier, `dgesdd` regime 1, does not fit the trend due to a weakness in the WPE: namely that in the absence of any large features, as all the other regimes have, the WPE effectively falls back to the standard PE which the noisy behavior drives toward 1.0.

   This primary finding: "Signals that are more complex are harder to predict." At first glance this finding may seem incredibly trivial, of course a complex signal is harder to predict than a trivial one, but this is *not* the core result here as we will elucidate now with Figure **??**.

2. Talk about structure analysis. Just because there is forward information transfer, does not mean that linear predictors can get at this.

3. For this show a figure of (a) ARIMA vs MASE (b) LMA vs MASE in a side by side plot.

4. full results. Image: MASE vs. WPE for both LMA & ARIMA. Points to make:

   (a) clusters are distributed differently

   (b) clusters are shaped differently—tight or not

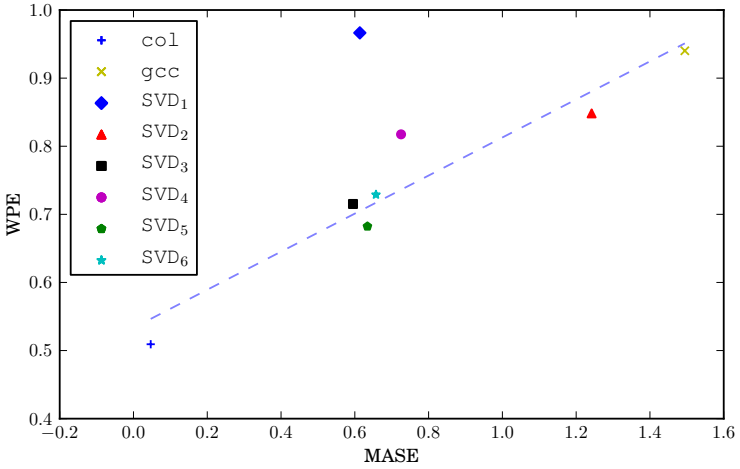
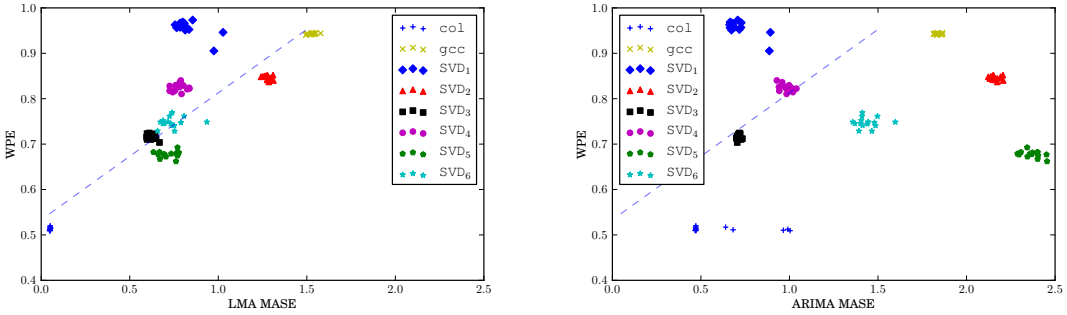


Figure 5: The best MASE among all runs and prediction methods vs weighted permutation entropy. For each of these, the word length used is 6. The dashed line is a least-squares linear fit of all the points except for $\texttt{SVD}_1$ which we have excluded for reasons explained in the text.



(a) The MASE of LMA vs weighted permutation entropy. For each of these, the word length used is 6. Prediction error vs WPE values largely fall near the fit from Fig. 5 indicating that LMA is a near- optimal prediction method for this data.

(b) The MASE of ARIMA vs weighted permutation entropy. For each of these, the word length used is 6. Prediction error vs WPE values for many of the traces fall far to the right of the fit from Fig. 5 indicating that ARIMA is a poor choice to predict this data.

(c) clusters move differently between LMA and ARIMA

(d) finally, the diagonal line is important. If you're below it, you could do better.

In Figures 6a and 6b we directly compare the performance of the LMA and ARIMA prediction methods (respectively) to the value of the weighted permutation entropy for all runs of each program under consideration. The LMA MASE values are largely similar to those of the best predictions, primarily because LMA often performed superior to ARIMA and the naive method. On the other hand, the ARIMA MASE values are largely uncorrelated with WPE values. The fact that ARIMA is uncorrelated with WPE is in large part one of the major findings of this work. More specifically, say we tried to predict an arbitrary noisy real-valued time series with an "out-of- the-box" prediction strategy like ARIMA as proposed in [**?**], and say you got inconsistent and bad forecasts, (i.e., perform worse than the naïve random walk strategy (MASE$>$ 1). How do we determine if the prediction strategy is not adequate for the prediction task, or if the signal is simply too complex to predict. If a signal is too complex and little forward information transfer is present we may simply not be possible to do better than the random walk, in which case we should not worry ourselves over finding a more complicated prediction strategy. However if we measure the complexity to be low, WPE < 0.85 (see Fig. 5), we can likely do better than the random walk and should search for more adequate prediction strategies.

By way of example, consider `col_major` (the blue +s in Figure **??**). If we use the out-of-the-box ARIMA method from [**?**] we get MASE values ranging from ~ 0.5 to ~ 1.0. From that information we may assume that the best predictions that can be done are twice as good as random-walk, and in some cases predictions will only do as well as random-walk. In the latter such cases, the random-walk prediction strategy is probably the best bet as it is so simple and produces very similar error. However, if we calculate the WPE we see that `col_major` has a WPE of 0.5131 ± 0.0034. This incredibly low entropy implies a great deal of deterministic structure which can be utilize for prediction, even though this did not appear to be the case with ARIMA. In fact using LMA on this signal we get an average MASE value of 0.05 ± 0.0018, an error that is on average 20 times better than the random walk forecast, and 10 times better than the best ARIMA forcast, with little variance. Alternatively, we can look at the other end of the spectrum: `403.gcc` . This signal has a WPE of 0.9430 ± 0.0013 and each prediction method we applied was significantly out performed by a simple random walk (the best strategy in the case of structureless time series). Since `403.gcc` has such little structure according to WPE it is safe to assume that he best strategy will be random walk and that continuing to search for a more accurate prediction strategy will likely be a fruitless process.

This however does not contradict our hypothesis, it just means that ARIMA is ill-suited for the time series under investigation. To see this more clearly, consider Figure **??**. Since larger MASE values constitute worse predictions, ARIMA performed worse than LMA in all traces except $\texttt{SVD}_1$.

The WPE is sensitive to both linear and nonlinear structure. When you have a low WPE and a high ARIMA it could be that the structure WPE is picking up is simply nonlinear structure that LMA can handle but ARIMA cannot. So while the ARIMA prediction look really bad there is plenty of structure present as suggested by WPE and taken advantage of by LMA but since it is nonlinear ARIMA can't take it into account and does bad.

5 Conclusions & Future Work

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 [16]. We are working on alternative methods that sidestep that complexity.

6 New Figures and Tables

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] Joshua Garland and Elizabeth Bradley. Predicting computer performance dynamics. In *Proceedings of the 10th International Conference on Advances in Intelligent Data Analysis X*, pages 173–184, Berlin, Heidelberg, 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] Liangyue Cao, Alistair Mees, and Kevin Judd. Dynamics from multivariate time series. *Physica D*, 121:75–88, 1998.