Statistically relaxing to generating partitions for observed time-series data

Michael Buhl* and Matthew B. Kennel†
Institute For Nonlinear Science, University of California, San Diego, La Jolla, California 92093-0402, USA
(Received 26 February 2004; published 22 April 2005)

We introduce a relaxation algorithm to estimate approximations to generating partitions for observed dynamical time series. Generating partitions preserve dynamical information of a deterministic map in the symbolic representation. Our method optimizes an essential property of a generating partition: avoiding topological degeneracies. We construct an energylike functional and use a nonequilibrium stochastic minimization algorithm to search through configuration space for the best assignment of symbols to observed data. As each observed point may be assigned a symbol, the partitions are not constrained to an arbitrary parametrization. We further show how to select particular generating partition solutions which also code low-order unstable periodic orbits in a given way, hence being able to enumerate through a number of potential generating partition solutions.

DOI: 10.1103/PhysRevE.71.046213 PACS number(s): 05.45.Tp, 07.05.Kf, 02.60.Pn, 05.10.−a

I. INTRODUCTION

When a chaotic signal is measured experimentally it represents a challenge to the experimenter. Compared to regular systems, chaotic systems are more difficult to model, and, if a prior model exists, care must be taken when comparing it to the data. We would not expect the output of the model to agree exactly with the measured data—as small initial uncertainties in the data will exponentially grow until any two trajectories are completely dissimilar. Instead of a direct comparison, often we try to reconstruct the dynamics from the data and compare this to the model. There are many ways to try to reconstruct the dynamics, one important way is to look at symbolic dynamics, where the dynamics is described by transitions between a finite set of symbols.

The general concept of symbolic dynamics is to partition the state space into a finite number of nonoverlapping regions, with each region colored with a different symbol. Orbits in the continuous state space correspond to sequences of symbols drawn from a finite alphabet. The evolution operator of the dynamics becomes merely a "shift operator" in symbolic space, moving the "point" one step into the future along a countable sequence of symbols. Topological considerations in the continuous space turn into grammatical considerations in the symbolic space, concerning the presence or absence of legal transitions and various "words" occurring in the symbolic sequence. By reducing the dynamics this way, many sorts of analysis of the dynamics, both theoretically and for observed time series, become much simpler, especially in opening up opportunities from information and communication theory. In addition, the theoretical study of symbolic dynamics has been quite active, and there has been much work connecting the symbolic dynamics to the unstable periodic orbits in the attractor.

We assume that the deterministic dynamics are either already a mapping $\boldsymbol{x}_{i+1}=\boldsymbol{f}(\boldsymbol{x}_i)$, or have been so projected by a Poincaré section. Here we shall also assume that $\boldsymbol{f}(\boldsymbol{x})$ is invertible; although the methods described here can easily be extended to noninvertible systems. A partition, denoted as $\mathcal{P}$, defines a discretization of the observed sequence's state space, and hence a projection to a sequence of symbols, $s_i=\mathcal{P}(\boldsymbol{x}_i)$, $i=1\cdots N$ where s is drawn from an alphabet of size A, $s\in 0,1\cdots A-1$. Each region of the partition carries the "code" for a particular symbol in the alphabet, and defines the transformation from points in the state space (which fall inside only one partition element each) into symbols. Note that the partition can have multiple disjoint regions for the same symbolic code.

Imagine a bi-infinite orbit of the dynamics $\boldsymbol{x}_i$, $i\in\mathbb{Z}$. Given a partition $\mathcal{P}$, we find not only $s_i=\mathcal{P}(\boldsymbol{x}_i)$, but also the symbols of all the future and past iterates of $\boldsymbol{x}_i$, yielding bi-infinite symbol sequence S_i centered around time point i, $S_i=[\cdots s_{i-2}s_{i-1}\cdot s_i s_{i+1}s_{i+2}\cdots]$. In the symbolic space, the analog of the dynamical operator $\boldsymbol{f}$ is the shift operator σ, $S_{i+1}=\sigma(S_i)$, which moves the point (the marker for the current time) one step to the right. By adding a metric and excluding a set of symbol sequences which are forbidden in the dynamics, the S_i live in a *shift space*, the symbolic analog of the dynamical state space. We can define a partition-dependent map $\phi_{\mathcal{P}}(\boldsymbol{x})$ from the original d dimensional state space $\mathbb{R}^d$ to the shift space Σ which contains the S_i. Σ is a subset of the space of all possible bi-infinite symbol sequences. For a *generating partition* $\phi_{\mathcal{P}}$ is injective [1] except for a set of measure zero, i.e., $\phi_{\mathcal{P}}(\boldsymbol{x})=\phi_{\mathcal{P}}(\boldsymbol{x}')$ implies $\boldsymbol{x}=\boldsymbol{x}'$. As $\boldsymbol{x}=\boldsymbol{x}'$ implies $\phi_{\mathcal{P}}(\boldsymbol{x})=\phi_{\mathcal{P}}(\boldsymbol{x}')$ as the partition is deterministic, $\phi_{\mathcal{P}}(\boldsymbol{x})$ is a homeomorphism.

We desire that if $\|\phi_{\mathcal{P}}(\boldsymbol{x})-\phi_{\mathcal{P}}(\boldsymbol{x}')\|$ is small, so is $\|\boldsymbol{x}-\boldsymbol{x}'\|$. With finite data, we cannot take limits to zero, but we may attempt to minimize close distances. By construction, sufficiently near points in $\mathbb{R}^d$ have close symbolic sequences in their most significant digits, except near partition boundaries. Additionally, in a good partition nearby points in Σ remain close when mapped back into $\mathbb{R}^d$. By contrast, bad partitions induce topological degeneracies where similar symbolic words map back to globally distant regions of state space, the problem described in Ref. [2]. The map $\phi_{\mathcal{P}}^{-1}$ is no longer one

*Electronic address: mbuhl@clack.ucsd.edu
†Electronic address: mkennel@ucsd.edu

to one. Conceptually, as we add symbols to our symbol sequence, we ought to localize down to smaller and smaller regions of state space—with an improper, nongenerating, partition, this will fail to happen even with increasingly long amounts of data.

For noninvertible one dimensional maps, the generating partition (GP) can be easily found with known results. A GP exists with partition boundaries at the critical points of the map. For nonhyperbolic higher-dimensional systems it has been conjectured that the partition line is along the homoclinic tangencies, points where the stable and unstable manifolds are tangent [3]. However, this is still not sufficiently constraining, and not all such choices yield valid partitions [4]. In addition, the tangencies can be difficult to locate, especially from observed data alone. Locating the homoclinic tangencies requires accurate estimates of the derivatives. This can be quite difficult in the situation we address herein, when we only have a time series, and the equations of motion are not available. There exist methods of connecting the tangencies which will create generating partitions [4]; however, these require the location of tangencies outside the attractor to be known. These would be particularly hard to find from a time series.

Davidchack *et al.* [5] and Plumecoq and Lefranc [6] proposed schemes to estimate a generating partition based on the unstable periodic orbits (UPOs) of the system. Under a generating partition, each point in the attractor defines a unique bi-infinite symbolic sequence. Thus each UPO should have a unique periodic symbolic sequence. An approximate generating partition can thus be constructed by successively assigning symbols to regions near UPOs of increasing order so that distinct UPOs give distinct symbolic codes. A generating partition for the Ikeda attractor, using the equations of motion, was found in Ref. [5]. Unfortunately, these methods can present difficulties with realistic experimental data. While it is possible to find UPOs from a time series [7–10], it is often difficult to find the UPOs with sufficient accuracy and quantity to apply the methods in Ref. [5,6]. Most UPO-locating methods use the points in the time series to create a model of the dynamics and then find UPOs from the model. However, an accurate model requires a large amount of rather noise free data. In particular, with noisy, finite length time series, it is especially hard to find the high period orbits which are necessary to localize the partition boundary. Plumecoq and Lefranc [6] were able to find a partition from a clean laser time series in this manner, but, in situations where the amount of data is more limited, we believe alternate methods to directly estimate the partition without going through the UPOs are needed.

Our approach to find approximate generating partitions is to search for a candidate partition such that the implied map from the shift space containing the symbol sequences to the observed continuous space data is approximately homeomorphic. The desired goal, satisfied by a generating partition, is that *short sequences of consecutive symbols localize the corresponding continuous state space point as well as possible*. Equivalently this says that nearby symbol sequences should correspond to nearby points in state space. Otherwise, the partition will map two similar symbol sequences to remote parts of the attractor, an improper topological degeneracy. In the usual circumstance that the observed points come from stationary ergodic dynamics, i.e., the natural measure, one cannot make estimates of the partition boundaries outside the attractor from the observed data alone.

In our previous work [11] we showed a concrete algorithm and partition representation for estimating generating partitions with that criterion. Here, we improve upon that method in a number of ways. First, we employ a time-symmetrical symbolic metric which is also one typically used in theoretical symbolic dynamics. Second, we now individually assign every observed point to a symbolic code, meaning that there are no artificial boundaries imposed by a particular choice of empirical basis functions which were used to parametrize the partition in Ref. [11]. Third, we define appropriate neighborhood sizes automatically and adaptively with a model weighting inspired by the minimum description length principle for statistical model selection. Fourth, we use an optimization metric which is in the form of a separable energylike functional, which in turn permits using recently developed optimization algorithms inspired by problems of statistical mechanics: we stochastically relax from higher energy states and poor partitions to lower energy states and good partitions. Finally, we show how the user may optionally fix the symbolic coding of lower-period UPOs (the easiest to locate numerically), in order to preferentially select the different generating partitions compatible with those various codings. As there is no unique generating partition for any system, it may be desirable to impose additional constraints to choose one of the multiple valid solutions. In other words, we can enumerate over a certain number of different generating partitions according to the codes they assign to low-order UPOs, as long as these UPOs lie close to the observed data.

II. STATISTICAL PARTITION REFINEMENT

We first outline the general approach of our algorithm. In subsequent subsections we provide additional details regarding its specific steps.

Recall that we have a time series of points $\boldsymbol{x}_i \in \mathbb{R}^d$, and we wish to assign symbols s_i to each observed point. We want to choose a partition where neighbors in the symbolic representation Σ are neighbors in continuous state space $\mathbb{R}^d$. To do this we need to define what we mean by "neighbors in Σ," and then optimize the partition to best satisfy this criterion. We view the problem as similar to finding a ground state of a spin glass in statistical mechanics. Each point is assigned a configuration-dependent energy: points whose symbolic neighbors are close physical neighbors are given lower energy. The optimal partition is defined as the configuration of s_i which yields the lowest global energy. It is analogous to a spin glass because the total energy Hamiltonian is separable, but there are global configuration interactions. In addition, naive local minimization of energy need not lead to the global minimum.

The notion of distance in the original continuous state space is usually already obvious (we use Euclidean distances in this work), but we must define "close" for symbolic sequences. We want neighboring symbol sequences to share a

large central block: symbolic neighbors should have the same symbolic future and past for some finite number of steps. Previously [11] we defined a metric between symbol sequences by embedding the symbol sequences in Σ into the points $\boldsymbol{y}$ in the unit square $[0,1]\times[0,1]$:

$$\boldsymbol{y}_i = \left(\sum_{k=1}^{k_{\max}} s_{i-(k-1)}/A^k, \sum_{k=1}^{k_{\max}} s_{i+k}/A^k \right). \tag{1}$$

Now $\|\boldsymbol{y}_i - \boldsymbol{y}_j\|$ defines the distance between points S_i and S_j in the symbol space. This gives a metric on Σ and a useful visual representation, but it has undesirable properties. For example, the sequences $1.000\overline{0}$ and $0.111\overline{1}$ are close in this metric even though they have no symbols in common. We use the notation $\overline{s}$ to mean that the symbol s repeats to infinity. We can use a Gray coding to fix some of these problems, but even with Gray coding some neighbors are still nonoptimal. For example, the symbol sequence $1.000\overline{0}$ will be closer to the sequence $1.100\overline{0}$ then the sequence $1.110\overline{0}$ is even though it differs from both in the second symbol.

A traditional metric used in the theoretical study of symbolic dynamics [12] is

$$D(S_1,S_2) = \begin{cases} 2^{-k} & \text{if } S_1 \neq S_2 \\ 0 & \text{if } S_1 = S_2 \end{cases}, \tag{2}$$

where k is the largest integer such that $S_1(k)=S_2(k)$ and $S_1(-k)=S_2(-k)$ and with the convention $k=-1$ if $S_1(0) \neq S_2(0)$. In essence, this measures the number of symbols that are the same in the central block, with no effect from symbols after the common central block. We can easily see that this satisfies all the usual requirements for a metric: $D(S_1,S_1)=0$, $D(S_1,S_2)=D(S_2,S_1)$, and the triangle inequality $D(S_1,S_2)+D(S_2,S_3) \geqslant D(S_1,S_3)$. Under this new metric, the sequences $1.1\overline{10}$ and $1.10\overline{0}$ are now equidistant to the sequence $1.000\overline{0}$. The Gray coding used in the implementation of [11] becomes unnecessary. Now, by definition, the distance between the sequences $1.00\overline{0}$ and $0.11\overline{1}$ is maximal as they share no symbols. The downside is that the projection of a symbolic time series into symbolic space cannot be graphically displayed in an intuitive manner.

After defining what we mean by close neighbors using Eq. (2), we can look for a good partition. Previously [11], we defined a cost function which measured the number of points which the partition did not localize well (where the state space distance to their symbolic neighbors was large) and minimized it using differential evolution [13], an algorithm for global multidimensional optimization in continuous spaces which does not require derivatives. A flaw of that approach is that the partition was defined by a small set of radial basis functions (RBFs). The number of those was an external arbitrary parameter, and their centers and strengths the free optimization variables. Unfortunately, the complexity of the representable partition boundaries was limited by the number of RBFs used. Using too many RBFs made the optimization problem significantly slower and less likely to find a near-global minimum, i.e., the estimated partition could often be worse than using a smaller representation, even though the larger RBF basis set could be more flexible in representing partition boundaries.

The alternative we presently employ is to define an energy functional upon the full configuration space—assigning individual symbols to every observed point—which we minimize to find the estimated partition. As in a spin glass or Ising system, we assign a local energy to each point. Also, like a spin glass, changing the configuration of any point not only affects the energy of that point but also the energy of its interacting points, so that the energy function is global. There now exist several good stochastic techniques for minimizing energies of state of spin glasses and similar combinatorial optimization problems. An optimization method we have found to be suitable for our problem is called τ-extremal optimization (τ-EO) [14], applicable to any discrete optimization problem whose cost function H is separable, i.e., can be written as a sum of local energy interactions:

$$H = \sum_{i=1}^{N} H_i(s_i,\{s_j\}), \tag{3}$$

for spin glasses, e.g., $H_i = s_i \Sigma_j J_{ij} s_j$, with s_i the spin of particle i and J_{ij} the interaction matrix. Neither of the global error function statistics we used in Ref. [11] is of this form.

We want our energy statistic to be a measure of the *symbolic false nearest neighbors* (SFNN): points which are neighbors in symbol space but not in state space. This is in the spirit of the false nearest neighbors [15,16] statistic used to find embedding dimension: both count large-deviation "mistakes" in a related space which result from topological misembedding in the tested space. The natural local energy term is hence a measure of this mismatch. We start from an initial assignment of symbols to all points. Define

$$D_{i,\alpha} = \|\boldsymbol{x}_i - \boldsymbol{x}_{\mathcal{N}_\alpha(i)}\|. \tag{4}$$

Here $\mathcal{N}_\alpha(i)$ means the index of the nearest symbolic neighbor of S_i, given that $s_i=\alpha$, and all other symbols in the time series are unchanged. We shall discuss the case where there is more than one near neighbor with the closest distance (in symbol space) later in the text. $D_{i,\alpha}$ is the mismatch induced by having s_i set to symbol α in the context of a fixed background. It turns out that defining the local energy as the difference relative to the existing configuration works better than the absolute energy measure. The local energy for point index i is

$$H_{i,\alpha} = \max_{\beta \neq \alpha} (D_{i,\alpha}^2 - D_{i,\beta}^2), \tag{5}$$

the difference induced by flipping s_i from its current assignment $s_i=\alpha$ to the best of all the alternatives. As α is the current symbol of point index i we can write this as just H_i. The global energy is $H_{\mathrm{SFNN}}=\Sigma_i H_i$. This has the feature that points which contribute the most to the energy are, by definition, the points which would reduce the energy the most if their symbols are changed. Over a binary alphabet the maximum reduces to simply the difference between the squared distances using the two symbols in s_i. This relative energy breaks the analogy to physical spin glasses somewhat but

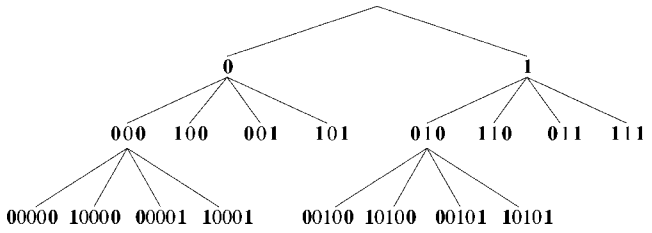

FIG. 1. The symbolic tree for a binary alphabet. Shown are the labels of each node, where the bold faced symbols are the symbols that have been added to the node. Contained within each node are all the points whose symbolic codes match the label.

improves the optimization performance. In sum, optimizing the empirical partition is minimizing H_{SFNN} over the configuration, the assignment of symbols to each observed datum. By attempting to minimize the corresponding difference in continuous space when symbolic distances are small, this drives the partition implied by the current configuration closer and closer to injectivity.

A. Neighborhoods in symbolic space

Once a metric has been imposed on Σ via Eq. (2) we can use it to find nearest neighbors. As the distance only depends on the number of common symbols in a central block, the structure of metric naturally leads to the construction of a symbol tree. This will find symbolic neighbors in order $\log N$ time with construction of the tree taking $N \log N$. Except for the root, every node of the tree has A^2 children, contains some number of points from the time series, and is given a symbolic label; the root only has A children. In the root node we place all of the points of time series. Its children are given labels $\sigma_0 \in 0 \cdots A-1$ and contain the points such that $s_i = \sigma_0$. Their children are given labels $\sigma_{-1}\sigma_0\sigma_1 : \sigma_{\pm 1} \in 0 \cdots A-1$ and contain the points $s_{i-1}s_i s_{i+1} = \sigma_{-1}\sigma_0\sigma_1$. The next layer of children looks at the next symbol on both sides and has labels $\sigma_{-2}\sigma_{-1}\sigma_0\sigma_1\sigma_2$ and contains the points such that $s_{i-2}s_{i-1}s_i s_{i+1}s_{i+2} = \sigma_{-2}\sigma_{-1}\sigma_0\sigma_1\sigma_2$. This continues for as long as there are points from the time series to populate the child nodes. For an example of the tree with a binary alphabet, see Fig. 1.

This structure makes it very easy to search for all points which are symbolic neighbors of any given point. All that needs to be done is to descend the tree to the node before the terminal node (which would be a solitary node containing only the index point we are searching around). All the other points in this node are the symbolic nearest neighbors of S_i under the metric given by Eq. (2). Unlike the metric used in Ref. [11] it is likely there are multiple equidistant nearest neighbors. Given two symbol sequences S_i and S_j which share a central block, if a third sequence S_k also has that central block, then all three sequences are equidistant under this metric. One obvious way to account for this phenomenon is to take all the symbolic nearest neighbors into account. We find their centroid $\boldsymbol{x}_c$ in the continuous space, and using that in Eq. (6) to define the distance to near symbolic neighbors:

$$D_{i,\alpha} = \|\boldsymbol{x}_i - \boldsymbol{x}_{c,\alpha}(i)\|. \tag{6}$$

We define the list $\mathcal{N}_\alpha(i)_j$ to be the indexes of all the symbolic neighbors around point i, and the index j runs between one and N_N, the number of symbolic neighbors. The centroid is then

$$\boldsymbol{x}_{c,\alpha}(i) = \frac{1}{N_N}\sum_j \boldsymbol{x}[\mathcal{N}_\alpha(i)_j]. \tag{7}$$

For a good partition, the neighbors $\mathcal{N}_\alpha(i)_j$ define a closed region in state space which contains $\boldsymbol{x}_i$, so the distance between $\boldsymbol{x}_i$ and the center of the region will be small.

In practice we need to define a maximum time difference k to distinguish points S; if k is sufficiently large then we can guarantee that with finite data all symbolic points are distinguished by looking at blocks $[-k,k]$ around the center point. One small technical problem is how to treat the beginning and end of the time series. In order to create the tree, all points must have k future iterates and k past iterates. The first and last k points cannot simply be ignored, for a correct symbol must be assigned to them. Our solution is to add ghost points to the beginning and end of the time series. The nearest neighbor to point $\boldsymbol{x}_1$ is found, and, for purposes of creating the symbolic tree, it is assumed that the point $\boldsymbol{x}_1$ has the same symbolic past as its nearest spatial neighbor. The same is done for the last point in the time series $\boldsymbol{x}_N$; it is given the same symbolic future as its nearest neighbor. If the partition is close to generating, then we expect that a point should have a very similar symbol sequence to its neighbors. Adding these ghost points does introduce small errors, but as long as $N \gg 2k$ this can safely be neglected. Typically, we choose k so that A^{-k} is near the computational numerical precision.

B. Spatial smoothing

The optimization procedure that we use can often produce partitions whose boundaries are not smooth. A smooth partition is desirable as it means $\phi_{\mathcal{P}}^{-1}(\boldsymbol{x})$ is also well behaved—we expect that the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ have similar symbol sequences for $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ small enough. We can cause the algorithm to favor partition boundaries by smoothing the raw $D^2_{i,\alpha}$ with the $D^2_{i,\alpha}$ of the points neighbors in the continuous space, and using that in the optimization criterion. Let $\mathcal{N}^{\mathrm{s}}(i)$ be the list of spatial—not symbolic—neighbors to the point $\boldsymbol{x}_i$. The squared distances $\tilde{D}^2$ are smoothed by a kernel K:

$$\tilde{D}^2_{i,\alpha} = \frac{\sum_{\mathcal{N}^{\mathrm{s}}(i)} K(\boldsymbol{x}_{\mathcal{N}_s(i)} - \boldsymbol{x}_i) D^2_{\mathcal{N}^{\mathrm{s}}(i),\alpha}}{\sum_{\mathcal{N}^{\mathrm{s}}(i)} K(\boldsymbol{x}_{\mathcal{N}_s(i)} - \boldsymbol{x}_i)}. \tag{8}$$

We would like a kernel which decreases for larger distances and also has finite support, so that we only need to sum over a finite number of neighbors. One such frequently used kernel is the Epanechinov polynomial kernel [19],

$$K(z) = 1 - \frac{|z|^2}{R_K^2}. \tag{9}$$

Here R_K is the size of the kernel, which is a free parameter. This kernel is often used for density estimation. As they do not change during the optimization, the nearest neighbors in state space $\mathcal{N}^{\mathrm{s}}(i)$ to each point and the associated kernel

weights need only be computed once, at the beginning of the minimization. The standard k-dimensional tree algorithm [20] can be used to find the spatial neighbors within the support of the kernel.

We want R_K to scale with the size of the attractor. We first find the cumulative distribution of *random* two-point distances $\|\boldsymbol{x}_\alpha - \boldsymbol{x}_\beta\|$ for the time series. We then choose R_K such that only a small fraction η of the random two point distances are smaller. Here η is a free parameter that we shall discuss in Sec. III, typically $\eta \approx 0.05$, meaning that typically the local smoothing is over 5% of points.

After the distances are smoothed, we construct the cost function by assigning a local energy or fitness H_i (larger is worse) to each point:

$$H_i = \max_{\beta \neq \alpha} (\tilde{D}_{i,\alpha}^2 - \tilde{D}_{i,\beta}^2). \tag{10}$$

This quantifies the appropriateness of the current symbol choice ($s_i = \alpha$) compared to alternatives. For a binary alphabet, this simply reduces to the difference between the current symbol and the flipped symbol. The total cost function is then simply

$$H_{\mathrm{SFNN}} = \sum_{i=1}^{N} H_i. \tag{11}$$

With the addition of smoothing, this is the same as Eq. (5) and thus appropriate for energy minimization techniques.

C. Combinatorial optimization

Once a metric has been defined on the symbol space, we can define a cost function, the minimum of which is our approximate generating partition. However, finding that minimum is a difficult computational problem: a naïve brute force approach would take A^N operations. Previously, in Ref. [11] we made the search for good partitions computationally feasible by defining the partition with respect to an *ad hoc* parametrization employing a few radial basis functions whose centers can move continuously around the attractor. That parametrization was less flexible than our current method, in that there were only a finite number of centers which could define the partition boundary to limited accuracy. In the present work, we have the freedom to assign any symbol to every observed point, allowing nearly arbitrary partition shapes. Ordinarily that would result in a very difficult optimization problem. The spatial smoothing we apply biases towards more continuous partition boundaries, and also smoothes the energy landscape. This tends to makes the optimization problem easier, as there are now fewer local minima in which it can be trapped.

We now wish to assign a symbol to each point in the time series such that H_{SFNN} is minimized. At this stage any global combinatorial search method may be employed. We describe herein the one we used which empirically gave successful results with computational efficiency, but do not make claims about its superiority over other methods. We search for the best symbol assignment with a stochastic search method [14] called τ-EO, or "τ-extremal optimization," based upon ideas from nonequilibrium statistical mechanics as opposed to the equilibriumlike principles of simulated annealing. In the most naïve version of this algorithm (extremal optimization), the fitness H_i is computed for each point, and the largest (least-fit) H_i is selected for a *forced* configurational change (unlike simulated annealing). The H_i are recomputed given the new symbol configuration, and the algorithm continues until it produces a step which increases $\Sigma_i H_i$. The limitation of this deterministic strategy is that it often becomes trapped in local minima. To find global minima, we ought to occasionally take steps which do not reduce $\Sigma_i H_i$ as well as randomize the perturbations, which is the point of τ-EO. First, let all H_i of the current configuration be sorted in decreasing order. Instead of always choosing the point with the maximum local energy to perturb (i.e., H_1), we choose the Kth largest such H_i, drawing a random integer $K \in [1, N]$ from the power law distribution,

$$P(K) \propto K^{-\tau}. \tag{12}$$

We discuss how to draw K in the Appendix. The symbol corresponding to the point with the Kth largest energy is flipped to a new configuration, choosing the new symbol which gives the least local energy—excluding the original symbol as a legal choice. After that symbol is flipped, then the point's fitness is recomputed and the list resorted. We use the heapsort algorithm so resorting the already mostly sorted list is fast. For large τ, K is nearly always 1, and we have the original EO scheme. For $\tau = 0$, all symbols have equal probability to be flipped, and the algorithm is equivalent to a random walk. For $1 \lesssim \tau \lesssim 2$ the configuration does a biased random walk towards our desired solution. As we do not always flip the symbol which reduces H_{SFNN} the most, we do not get trapped in local minima as easily. As the distribution is still heavily weighted towards low K, we still tend towards minimizing H_{SFNN}.

Before we compute the cost function H_{SFNN}, we must first find the distance in state space from each point $\boldsymbol{x}_i$ to its symbolic neighbors and compute $D_{i,\alpha}$, allowing the point to be each of the A possible symbols. However, when we change the symbol of $\boldsymbol{x}_i$ we have a potentially serious problem: the symbol tree can no longer be efficiently used to find nearest symbolic neighbors. We wish to calculate $\|\boldsymbol{x}_i - \boldsymbol{x}_{c,\alpha}(i)\|$ for $s_i = 0 \ldots A-1$, but when the symbol of $\boldsymbol{x}_i$ is changed, then the tree that was previously built is no longer a faithful representation. Recall that when the symbol s_i for $\boldsymbol{x}_i$ is changed, the symbolic codes of all the points $S_{i \pm k}$ also change. The tree could be recreated after every single flip of a symbol; however, that is computationally intensive.

If the number of points in the time series N is large, instead of recreating the tree we can take a mean field approximation and use the tree as it was for a number of iterations. This is in the spirit of the mean field approach in statistical mechanics where, say in a spin glass, we neglect the effect each individual spin has on the mean field when we compute the energy of the system. Only the interaction of the spin and the mean field is considered, not how the spin changes the mean field. In our case, we are neglecting the effect the point $\boldsymbol{x}_i$ has on the partition when we calculate $D_{i,\alpha}$ for that point. This is only looking at the first order change, and ignoring

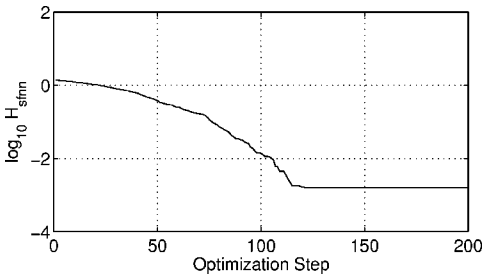

FIG. 2. An example of the average distance to the nearest symbolic neighbors H_{SFNN} (dimensionless) as a function of optimization step. We can see that after about step 120, the algorithm no longer decreases H_{SFNN}.

how changing the symbol of $\boldsymbol{x}_i$ affects the symbolic codes of $\boldsymbol{x}_{i\pm j}$ and thus how changing the symbol affects the partition. This will only have an effect if $\boldsymbol{x}_i$ is a symbolic neighbor of $\boldsymbol{x}_{i+j}$ for $|j|<k$ where k is the maximum depth of the tree, and for $N \gg k$ that should happen infrequently. We attempt to flip points n times before doing a full global regeneration of the symbolic tree, code lengths, and symbolic neighbors. If n is too small, the optimization will go slowly because it is constantly recomputing all the symbolic neighbors. If n is too large, the mean field approximation begins to break down. In practice, at each major step in the optimization, we choose n to be the number of points with $H_i>0$; i.e., the number of points which locally might seem to benefit from changing their configuration. This automatically adapts the bundling of the major iteration so that in a poor initial configuration (many $H_i>0$) many local steps are taken at once, but when the configuration is closer to the optimum (few $H_i>0$) the full global effects are recomputed after a small number of configuration changes. We further bound n to be in the range $n \in [n_{\min}, n_{\max}]$, two user-set external parameters. Typically we let $n_{\min}=1$ and $n_{\max}=40$.

Most of the time, we use a very simple starting and stopping criterion. To start, we typically choose an initial partition where all symbols are chosen at random. We stop after the optimization algorithm was iterated a fixed number of steps. Typically, as a function of optimization step, H_{SFNN} descends rapidly, and then remains nearly constant. During this time, H_{SFNN} typically only decreases slightly, if at all. An example plot of H_{SFNN} as a function of optimization step is shown in Fig. 2. The total number of optimization steps is chosen such that the constant region has been reached, and the number of steps that have been attempted since H_{SFNN} was last reduced is comparable to the total number of optimization steps.

D. Weighting symbolic neighbors over depths

One remaining question now is how should we terminate the process of finding symbolic neighbors in this tree? The obvious choice is to descend the tree as deep as possible; that is, to the deepest matching node which still has more than one point. This works well for many cases, but we have developed an alternative, more sophisticated procedure. This extra complexity may be optional for some uses, but it holds some value in improving results.

With a noisy time series, there will be a maximum spatial resolution of actually useful data. This corresponds to a maximum depth in the symbolic tree beyond which it does not make sense to go. In fact, it would actually be detrimental to go deeper as deeper nodes would contain fewer points and thus unable to estimate $\boldsymbol{x}_c$ as accurately. Also, if the current partition choice is bad (which it will be at the start of an optimization), descending deeper into the tree will not necessarily help localize a point better.

Our point of view is to imagine the tree as defining nested probabilistic models for describing the density of points in the original state space. At each node we will model the density of points as a multidimensional Gaussian; for simplicity, a product of 1D Gaussians in each coordinate instead of a model with a true covariance matrix. Then the free parameters of this model are the center location and variance for each coordinate. Each node has a corresponding centroid which we could use in Eq. (6). Any symbolic point S_i matches a whole branch of possible nodes, corresponding to larger and larger central blocks of matching symbols, and fewer points. Is there an optimal *weighting* of all those possible centroids which will automatically have good properties: properly going down very deep when the localization is good, and otherwise preferring higher nodes (with more points and hence better statistics) when the localization is poor?

We view this as a general model selection or model weighting problem which can be addressed with Rissanen's [17] minimum description length theory and Bayesian weighting. At each node of the tree we estimate the *stochastic complexity* (code length) L of the node. This is a measure of two things: first, how much information it takes to describe the data described by that node (i.e., the location of all points in the continuous space) relative to the model in the node. In addition, it includes the amount of information specifying the model's parameters itself. In our case, it a measure of how well the node localizes the points in the node, with smaller code lengths being superior.

Since we will model each coordinate independently the local complexities will be additive for each coordinate and we need to consider one-dimensional distributions for the moment. The code length of each node for each coordinate is defined to be

$$S = -\ln \int f(x|\boldsymbol{p})\pi_1(p_1|\boldsymbol{p},\boldsymbol{q})\pi_2(p_2|\boldsymbol{p},\boldsymbol{q})\cdots\pi_n(p_n|\boldsymbol{p},\boldsymbol{q})d\boldsymbol{p}, \tag{13}$$

where $f(x|\boldsymbol{p})$ is the likelihood of x given the model f and parameters $\boldsymbol{p}$, and $\pi_i(p_i|\boldsymbol{q})$ are some prior distributions of the parameters p_i given some initial estimate of their values $\boldsymbol{q}$. As a rough approximation, the distribution of the points in each symbolic region can be approximated as Gaussian with mean μ and variance τ,

$$f(x|\mu,\tau) = \frac{\exp\left[-\frac{1}{2\tau}\sum_{i=1}^{n}(x_i-\mu)^2\right]}{(2\pi\tau)^{n/2}}. \tag{14}$$

This is an approximation to the true distribution, but for our purposes it will give us good enough code lengths to esti-

mate reasonable tree weightings. We assume a Gaussian prior on μ centered about some value p,

$$\pi_\mu(\mu|p,\tau) = \frac{e^{-(n/2\tau)(\mu-p)^2}}{2\pi\tau/n}. \quad (15)$$

As usual, we take the variance on the distribution of the mean to be τ/n; i.e., the variance of the mean is $1/n$ the variance of the distribution itself. We can use any prior estimate for the mean as the value of p.

After Rissanen [17] we take the prior on τ to be

$$\pi_\tau(\tau|\nu) = \sqrt{\frac{3\nu}{2\pi}} e^{-(3\nu/2\tau)} \tau^{3/2} \quad (16)$$

with ν as a free parameter. The maximum of $\pi_\tau(\tau|\nu)$ occurs at $\nu=\tau$, so it is reasonable to take ν to be a prior estimate of the variance of the distribution. Plugging the above distributions in the definition of the code length, the code length of each node is given by

$$S = -\ln \int_0^\infty \int_{-\infty}^\infty f(x|\mu,\tau)\pi_\mu(\mu|p,\tau)\pi_\tau(\tau|\nu)d\mu d\tau. \quad (17)$$

This can be integrated to give

$$S = \frac{n+1}{2}\ln\left(\sum (x_i - p)^2 + \sum (x_i - \overline{x})^2 + 6\nu\right) + \frac{n}{2}\ln\left(\frac{\pi}{2}\right) + \frac{1}{2}\ln\left(\frac{\pi}{3\nu}\right) - \ln\Gamma\left(\frac{n+1}{2}\right), \quad (18)$$

where p and ν are the sample arithmetic mean and variance of points in the parent node, and $\overline{x}$ is the arithmetic mean of points in the current node, all for the current coordinate. The root node has no parent, so we take its code length to be infinite. This code length is only for a one dimensional Gaussian. For higher dimensional distributions, we approximate the distribution as the product of Gaussians in each dimension; the total code length is hence just the sum of the code lengths in each dimension $L_n = \Sigma_d S_n(d)$.

Given a code length for each node, we can use an analogous procedure to the symbolic time series modeling in Ref. [18], called there *context tree weighting*, to give a good predictor. At each node n, define the weighted code length:

$$L_w = \ln\left(\frac{e^{-L_n} + e^{-L_c}}{2}\right), \quad (19)$$

where here L_c is the sum of the weighted code lengths of all the children nodes of n, $L_c = \Sigma L_w(\text{children})$. For the terminal nodes of the tree, we set $L_w = L_c$. We can then use these code lengths to estimate any quantity we wish from the data. In Ref. [18] they estimated the entropy rates, here we wish to estimate the position of the centroid. For each node in the matching branch we can estimate the centroid of the node, $\boldsymbol{x}_n$, the arithmetic mean of the the points in the node. Then, for each node, the *weighted* centroid $\boldsymbol{x}_w$ is

$$\boldsymbol{x}_w = w_n \boldsymbol{x}_n + w_c \boldsymbol{x}_w(\text{child}) \quad (20)$$

with weights

$$w_n = e^{-L_w}/(e^{-L_w} + e^{-L_c}), \quad (21)$$

$$w_c = e^{-L_c}/(e^{-L_w} + e^{-L_c}),$$

and $\boldsymbol{x}_w$(child) is the weighted centroid of the child node. Note that we combine centroids from only the single matching child on the branch, but to define the weighting factor we must sum all children to get L_c. We can recursively compute this weighted average over all nodes that contain the point $\boldsymbol{x}_i$. If the partition localizes $\boldsymbol{x}_i$ well, then the code length of a node near the leaves of the tree will be small and the largest contribution to the centroid will thus be from that node. However, if the partition is such that $\boldsymbol{x}_i$ is never localized well, then no node will have a small code length and the contribution from nodes close to the root of the tree will be more influential. In practice, we see that most of the contribution to the overall weighted centroid comes from nodes near the root at the start of the optimization, when it is initialized with a bad partition. As the partition is refined, nodes deeper and deeper in the tree become important.

E. Summary of algorithm

1. Find spatial nearest neighbors and kernel weights for all points $\boldsymbol{x}_i$. Knowing the spatial nearest neighbors for $\boldsymbol{x}_1$ and $\boldsymbol{x}_N$ find the "ghost points." Assign initial symbols s_i, either at random or input from another algorithm.

2. Create the symbol tree from the current symbol configuration, defining the k symbols before s_1 and the k symbols after s_N to be identical to the corresponding symbols of their ghosts.

3. For every node in the tree, compute the code length summed over all coordinates.

4. For each point i, temporarily set s_i to α and find $D_{i,\alpha}$, either using the centroid of the deepest matching node or the weighted version thereof.

5. Smooth $D_{i,\alpha}$ over continuous-space neighbors using a Epanechinov kernel to produce $\tilde{D}_{i,\alpha}$.

6. Find the fitness of each point $H_i = \max_{\beta\neq\alpha}(\tilde{D}^2_{i,\alpha} - \tilde{D}^2_{i\beta})$ where $s_i = \alpha$.

7. Sort the fitnesses in descending order.

8. Set the number of symbols to change, n, to be the number of symbols with $H_i > 0$. Subsequently bound n between some $n_{\max}$ and $n_{\min}$.

(a) Choose an integer K randomly from a power law distribution $P(K) \propto K^{-\tau}$.

(b) Choose the Kth point from the sorted list and change its symbol from α to β where $\tilde{D}_{i,\beta} = \min_{\gamma\neq\alpha}\tilde{D}_{i,\gamma}$.

(c) Find the new fitness for this point in the mean field.

(d) Resort the list of fitnesses.

(e) If fewer than n symbols have been changed, then return to (a).

9. Return to step 2.

III. RESULTS

As the first test of this method we look at the Ikeda map, defined by

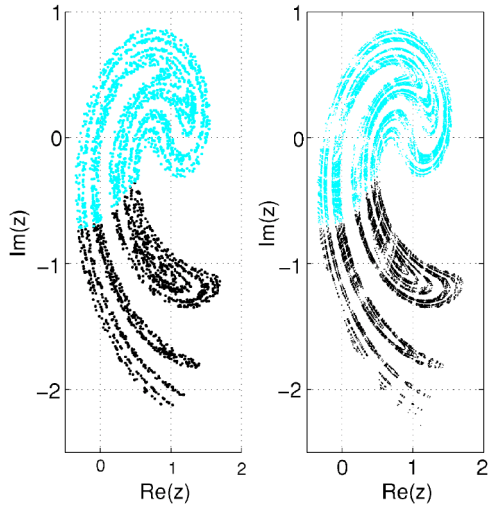

FIG. 3. (Color online) Left: partition estimated by optimizing H_{SFNN} on 3000 data points from the Ikeda map. Right: partition calculated from the UPOs, numerically extracted from the equation of motion. The partition we estimate from observed data alone is quite close to a presumably correct one, calculated from the method of Ref. [5]. The measure on the two figures is not the same: the left figure is a sample of the natural measure, whereas the right shows UPOs up to period 15. They avoid regions of homoclinic tangencies, contributing to the blank spaces.

$$z_{n+1} = p + Rz_n \exp\left(i\kappa - \frac{i\alpha}{1+|z_n|^2}\right) \tag{22}$$

with the standard parameter values $p=1$, $R=0.9$, $\kappa=0.4$, $\alpha=6$. Taking the real and imaginary parts gives a map in $\mathbb{R}^d$. The partition that we find is given in Fig. 3 on the left. On the right in Fig. 3, we show the partition found using the UPOs by the method of Davidchack [5]. To the eye, the partitions are nearly identical. In Fig. 4 we see intermediate steps in the optimization procedure which led to the partition in Fig. 3. Here, the initial partition was a random distribution

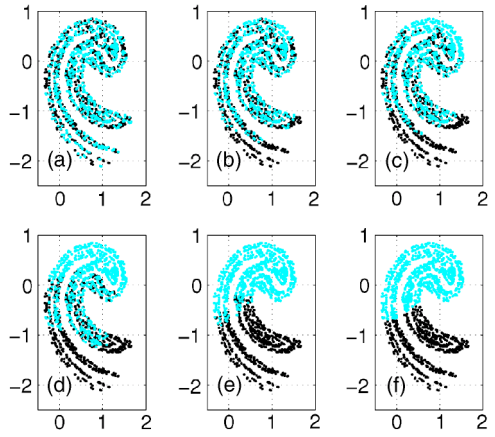

FIG. 4. (Color online) An example of the relaxation towards generating partitions. Panel (a) shows the initial random partition. The subsequent panels show the optimization after 10 (b), 20 (c), 50 (d), and 100 (e) optimization steps. Panel (f) shows the final partition after no improvement is detected for many iterations.

of symbols. Near the beginning of the optimization the partition is still mostly random, but it rapidly converges on a boundary.

To quantitatively test the accuracy of the partition, we look at the symbolic degeneracy of the unstable periodic orbits (UPOs). In a generating partition, each distinct UPO has a unique symbolic code; this is the basis for the partition finding scheme proposed by Davidchack *et al.* [5]. As the period p approaches infinity, the number of periodic points of each period should scale like

$$N_p \propto e^{h_T p}, \tag{23}$$

where h_T is the topological entropy.

As a measure of the accuracy of a partition, we can look at the mismatch between the topological entropy that we find and the true topological entropy. All UPOs up to a given period are calculated from the equations of the map. The points in the UPOs are assigned the same symbol of the closest point in the time series. Thus each of the UPOs is given a symbolic code and the number of unique symbolic codes is counted. If we see $\tilde{N}_p$ distinct symbolic codes given the partition, then the estimate of the mismatch of the topological entropy is

$$\delta h_T = \lim_{p\to\infty} \frac{1}{p} \ln\left(\frac{\tilde{N}_p}{N_p}\right), \tag{24}$$

where N_p is the actual number of periodic orbits of period p. For a generating partition, all UPOs have unique symbolic codes, so $\delta h_T = 0$. For partitions which are not generating, δh_T measures the amount of the attractor which is miscoded, and thus we can use it as a way to validate the partition. Since the equations of motion are used to find the UPOs, we cannot use δh_T to validate partitions when only a time series is available. In practice we use orbits with periods up to 15 when evaluating the limit in Eq. (24). For the Ikeda map the total topological entropy was estimated as 0.6033, again using UPOs of up to period 15. Note that this is different from the *metric* entropy. The topological entropy is sensitive to the number of different allowed words, while the metric entropy also weights the sequences by their probability, using the Shannon formula.

We checked the metric entropy rate under different partitions using a recently developed Bayesian entropy rate estimator [21] (based on previous results in Ref. [18]) which shows low bias. We took 5000 points from the Ikeda map and found their symbols relative to partitions estimated using the UPO-based method of Davidchack *et al.* [5] (which we assume is a correct partition up to the resolution of the data considered, given that the equations of motion were known), and our data-based method operated upon a different sample of 5000 points. The test time series was symbolized by assigning the same symbol as that given to each test point's nearest neighbor in the data base (partition generated from the UPO-coloring algorithm or our data-driven algorithm). Partitioning by homoclinic tangency resulted in an identical symbol stream as the UPO-coloring method. We estimated an entropy rate of $\hat{h}=0.5206$ (nats/iteration) for symbols relative to the UPO-based partition, and $h=0.5155$ with the par-

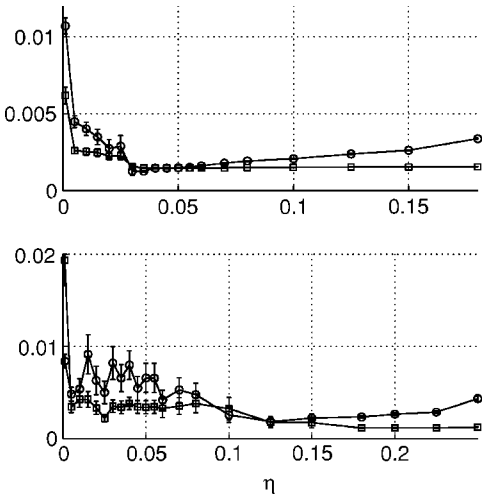

FIG. 5. The average final energy and mismatch of topological entropy as the state space smoothing η is varied. *x* axis: smoothing parameter η (dimensionless); *y* axis: mismatch in topological entropy δh_T (circles, dimensionless), and the average distance to the nearest symbolic neighbors H_{SFNN} (squares, dimensionless). The top graph is from partitions found from the Ikeda attractor and the bottom is from the Henon attractor, both using 5000 points and averaged over 100 starting partitions. The error bars show the standard deviation of the ensemble. It is entirely a numerical coincidence that the energy and δh_T may be plotted on the same scale for these two cases. What is important, however, is that the η which yields the minimum δh_T, which is unknown to the data analyst as the true partition is unknown, is close to the location of the minimum in H_{SFNN}, the empirically computed energy to minimize. This is evidence that our energy function quantifies generatinglike partitions, and that it can be used for selecting η.

tition found from our method. The estimated partition shows only a small discrepancy from the true (which always decreases entropy), though these were within the 90% confidence intervals of each estimate computed using the methods in Ref. [21]. The metric entropy should be equal to the largest Lyapunov exponent, which is estimated to be $\lambda_1 = 0.508$, computed from renormalized trajectory divergences, integrating the equations of motion over a 10^6 iteration ergodic sample. The fact that this Lyapunov exponent is slightly less than the estimated entropies is most likely from estimation bias because only 5000 symbols were used—for estimating entropy rate there is a positive modeling bias for finite *N*, as subtle correlations in the conditional probability structure (which must lower entropy rate) are not discerned sufficiently well.

One important free parameter in the partition refinement process is the amount of smoothing η. If η is too low, then the optimization becomes too difficult; however, when η is too large, details in the partition boundary are lost. We can calculate δh_T of the found partition as a function of η, and the results from both the Ikeda and Henon maps are plotted in Fig. 5. The Henon map is given by

$$x_{n+1} = A - x_n^2 + by_n, \quad y_{n+1} = x_n \tag{25}$$

with $A = 1.4$, $b = 0.3$. We see in both cases that the minimum in δh_T is close to the minimum in H_{SFNN}. In the realistic case

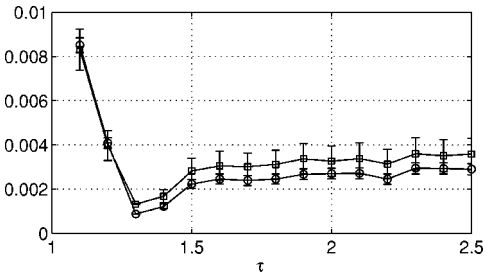

FIG. 6. Partition accuracy versus optimization parameter for 5000 point data set from Ikeda map. *x* axis: power law parameter τ (dimensionless); *y* axis: topological entropy mismatch δh_T, and the average distance to the nearest symbolic neighbors H_{SFNN} (both dimensionless). The topological entropy mismatch δh_T is plotted with circles, and H_{SFNN} is plotted with squares. For a range of τ similar partitions are seen. Plotted are averages over ensembles of 100 randomly chosen initial symbolic configurations. The data set remained constant. Similarly to Fig. 5, the location of minima over free parameter τ fortuitously coincides for H_{SFNN} and the unobservable δh_T.

when the UPOs needed to compute δh_T are not known, partitions can be found from a range of η, and the partition which yields the lowest value of H_{SFNN} is generally the best choice. As the initial partition used can affect the final partition found, here we averaged over an ensemble of 100 random initial partitions. The curves are not smooth because although we averaged over 100 initial partitions, in all cases we used the same time series. We could also average over an ensemble of different time series; however, that is a technique generally not available when working with real data.

Another free parameter in the optimization is τ, which controls the distribution of random variates. If τ is too small, then the time that the optimization needed to run increases, because the pressure to minimize the energy is too weak. For τ too large, the optimization freezes configurations rapidly (as it becomes the greedy search algorithm) and can get caught in a local minimum. Results from varying τ are shown in Fig. 6. In all cases 1000 major optimization steps were taken. For large τ we can see that more steps were needed, and the high value of H_{SFNN} can be taken to mean the optimization had not finished. The graph also shows a rise in H_{SFNN} for smaller τ as the optimization becomes stuck in local minima. We used $\tau = 1.5$ for the rest of the examples in this paper.

In our earlier work [11] we found approximate generating partitions by positioning a fixed numbers of centers about which we defined radial basis functions. A disadvantage that this had was that it limited the resolution of the partition boundary to the number of radial basis functions used, which could not become too large or the optimization procedure would fail. In Fig. 7, we compare this method with the method described in this paper by computing δh_T for the partitions found for several values of *N*, the length of the time series. Again, an ensemble of 100 random initial partitions were used and the mean δh_T is shown. The new partition refinement method typically finds partitions with about half the δh_T as the older scheme.

Similarly, we can check if the tree weighting of Sec. II D helps find better partitions. In Fig. 8, δh_T is plotted as a

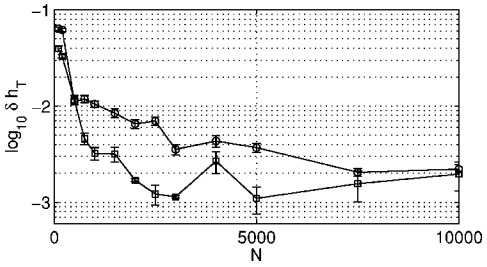

FIG. 7. Comparison of the mismatch of topological entropy for the partition refinement method discussed in this paper to the method previously proposed [11]. *x* axis: length of data set (dimensionless); *y* axis, logarithm of topological entropy mismatch (dimensionless). The circles are from the previous method minimizing K_{SFNN} using six centers and a population of 30. The squares are the method described here using $\eta=0.035$ and $\tau=1.5$. Both cases are averaged over an ensemble of 100 different initial partitions. For most of the lengths of time series considered, the present partition refinement technique yields a significantly smaller δh_T than the radial-basis function optimization method.

function of η when context tree weighting is used, shown with square points, and when it is not used, shown with circles. The time series was 5000 points from the Ikeda map. With large smoothing we can see that there is no substantial difference. However, the context weighting improves the results significantly for the smaller smoothing parameters. The dependency of the result on the external parameter η is reduced. At least for the data sets considered here, the information based tree weighting may be a superfluous complexity, though it may assist with other data sets.

It should be emphasized that although we have shown many different variations of the partition finding algorithm, they all find reasonably good partitions. We found that using context tree weighting is an improvement; however, good partitions can still be found without it. This algorithm finds partitions significantly better than the earlier method given in Ref. [11], but again the older method could still reliably find reasonably good partitions. Although there are a number of

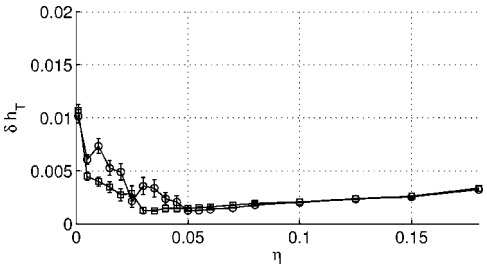

FIG. 8. The mismatch of topological entropy as smoothing in state space is varied, comparing nearest-neighbor centroids and weighted centroids. *x* axis: smoothing parameter η (dimensionless); *y* axis, topological entropy mismatch δh_T (dimensionless). In the circle points, the centroids were found by the mean of the positions all the nearest symbolic neighbors of each point. For the square points, the effective centroids were found by context tree weighting over all the symbolic neighbors, not just the nearest ones. 5000 points from the Ikeda map were used to find the partition, averaged over an ensemble of 100 random initial symbolic assignments.

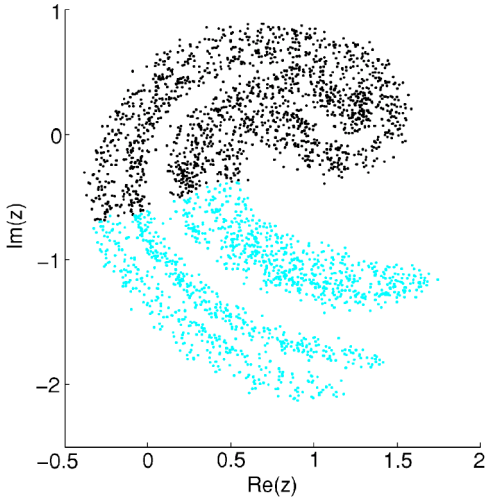

FIG. 9. (Color online) The partition found from 5000 points from the Ikeda map with Gaussian observational noise with standard deviation 5% the attractor size. Here $\eta=0.035$. The partition found is virtually identical to the partition found from the noise free time series.

free parameters, we can find good partitions over a wide range of values. The important parts to the algorithm are that in all cases we are finding partitions which localize short symbol sequences optimally, and we are also enforcing some smoothing in state space to ensure continuity when going from state space to symbol space. In the earlier algorithm in Ref. [11] this smoothing is done implicitly when we choose radial basis functions; in the method described in this paper the smoothing is done explicitly.

Noise is often an unavoidable component of a measured time series. To test the robustness of the algorithm under the presence of noise, white Gaussian noise with a standard deviation of 5% the size of the attractor was added to a time series from the Ikeda map. As seen in Fig. 9, a partition can still be found quite well. Comparing to the partition in Fig. 3, the partition from the noisy time series is visually indistinguishable from the true partition.

Before we try to find a partition, a proper embedding space for the time series must first be found. If the embedding dimension *d* is not chosen correctly, then a correct partition may not be found. Specifically, if the *d* is chosen too low, then we will have a large number of false nearest neighbors in the state space. These false neighbors are part of different regions of the attractor and thus should have completely different symbol sequences. Therefore the map $\phi_{\mathcal{P}}(\boldsymbol{x})$ from $\mathbb{R}^d$ to the symbol space is no longer a homeomorphism and our methods for finding partitions will fail.

One particular case where a correct embedding dimension is important is if the dynamics were generated by a one dimensional (1D) map. In that case, there is no stable manifold to the dynamics, so the past symbols are no longer needed to localize a point. The location of the generating partition for 1D maps is a solved problem, the partition boundaries are on the critical points of the map; however, it is still important that our methods find proper partitions in this case.

If we recognize that the dynamics come from a one dimensional system, we can easily modify our methods to only

look at future symbolic iterates. Specifically, our metric on the symbol space in Eq. (2) would change so that the integer k is chosen such that only $s_1(k)=s_2(k)$ is true; we no longer care if $s_1(-k)=s_2(-k)$. If we do this we can easily find the partitions of one dimensional systems.

More of a concern is what happens if we mistakenly treat a system generated by one dimensional dynamics as two (or higher) dimensional. This could easily happen from a measured time series if the time series has some noise in it. In this case we would be using both future and past symbolic iterates when only the future is needed. However, even so we can still find a good partition; if two symbol sequences are close using the past iterates, they will also be close if the past iterates are excluded. If we include the past, there will be points which are no longer neighbors; however, all points which are neighbors when the past is included will be neighbors even if the past is excluded.

IV. FIXING PERIODIC ORBITS

Generating partitions are not unique: there are several partition choices which will yield equivalent descriptions of the symbolic dynamics but with different partitions of the same state space. The dynamics are different in the sense that the specific grammar may differ, yet equivalent in the sense that the entropy and number of periodic orbits are the same. The structure of equivalences between symbolic dynamics on shift spaces is a major ongoing research project in theoretical symbolic dynamics [12], which we do not attempt to address here. A trivial example of an alternative generating partition is to take an existing generating partition and iterate it under the map; the iterated partition is also generating. More complicated examples also exist as shown by Eisele [22].

Experimentally, these alternative partitions may be desirable sometimes. The minimization algorithm will typically relax to one (or a small number of) particular estimated generating partition, depending the initial conditions and peculiarities of the specific statistic we use. Some partitions may localize points better in some regions of state space than in other regions, which could be desired in an experiment. Furthermore, some partitions may induce symbolic dynamics which are easier to estimate or predict (e.g., Ref. [18]). For some time series, the method we have described here will find several different partitions depending on the initializing partition (usually randomly chosen); it would be useful to have a way to constrain which partition will be found.

One way to try to find and choose between these alternative partitions it to fix, by hand, the symbolic codes of a few low period UPOs and insist that the estimated partition assign the given codes to them. The low period UPOs are first found from the time series; this can be done by looking for close returns or by methods which try to reconstruct the dynamics [7–10]. These UPOs are now assigned symbolic codes and inserted into the continuous state space and indirectly influence energy functional. However, as the periodic orbits are not actually part of the time series they are treated slightly from the data points.

We assume that the symbol choice that each of the points on the selected UPOs is correct, and given by the user. The effect of the UPOs on the estimated partition is mediated by the smoothing in the continuous state space, not continuity in the symbolic space. The UPO points are treated specially. They do not enter into the total energy H_{SFNN}, and their symbols are not ever altered. However, they are assigned a $D^2_{k,\alpha}$ distance as described below, and hence by being spatial neighbors to true data points they influence the partition in the kernel smoothing from D to $\tilde{D}$, the sum in Eq. (8) now covering points on the UPO in addition to the actual data points. As there are typically far more points in the time series than there are fixed UPOs, we must weight the UPO points to be worth more than a single datum and this weighting ought to scale with the number of points. For every point $\boldsymbol{z}_k$ which is one of the points on the fixed UPOs, assigned symbol s_k, we define $D_{k,\alpha}$ as

$$D^2_{k,\alpha}=\begin{cases}\epsilon N\|\boldsymbol{z}_k-\boldsymbol{x}_{\mathcal{N}_{U_\alpha(k)}}\|^2 & \text{if } \alpha\neq s_k\\ 0 & \text{if } \alpha=s_k\end{cases}, \tag{26}$$

where $U_\alpha(k)$ is the symbolic stream for the kth UPO point with the center symbol replaced by α. For instance, a period-3 point with code 012 corresponds to three points in shift space, namely $U(1)=[\ldots 012\,012.012\,012\ldots]$, and its two iterates. Then $U_\alpha(1)$ is $[\ldots 012\,012.\alpha 12\,012\ldots]$. $\boldsymbol{x}_{\mathcal{N}_{U_\alpha}(k)}$ means the centroid (or context-tree weighted mixture of centroids) of the symbolic neighbors to this point in the shift space, just as usual for symbolic neighbors. Hence we must be able to locate symbolic neighbors of perturbations to the true UPO's codes, though the UPO itself need not be observable as a neighbor in the symbol space. Note the importance differences: (i) the distance is amplified by a factor ϵN, (ii) when the partition approaches generating (small corresponding distances) which agrees with the fixed UPO codes, the penalty goes to zero. Otherwise, at the end of the minimization the error due to the fixed UPOs would have dominated the total cost function and a nonoptimal partition will be chosen. Nevertheless, a partition is occasionally estimated that does not agree with fixed UPO symbols; e.g., a periodic point may be encoded with a "1" but all of its spatial neighbors are given the "0" symbol. We can easily test if this has happened and only use partitions where the UPOs are correctly encoded.

For the Henon map, the three lowest period UPOs were fixed with different symbolic codings. Plots of some of the partitions found are shown in Fig. 10, and the symbolic codes of the fixed UPOs are given in Table I. The Henon map also has a second fixed point which was not included, as it lies outside the attractor for these parameter values. Although not shown in the figure, the trivial case of setting the symbol $0\rightarrow 1$ and $1\rightarrow 0$ in the fixed UPOs yields the same partition with the symbols interchanged.

As the partitions are found from a finite amount of data, some partitions can be found more easily than other partitions. More convoluted partitions with more boundaries are more difficult to find than simpler partitions. Thus if no UPOs are fixed, the algorithm usually will converge to the same partition. The other partitions typically will be more convoluted and have more points on the boundary than the

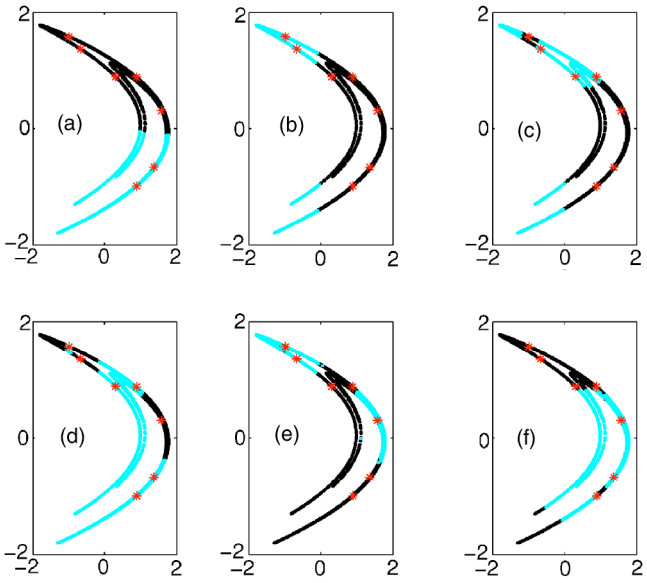

FIG. 10. (Color online) Several different partitions of the Henon map found by fixing the symbols of the three lowest period UPOs, which are shown as stars. The symbolic codes for the UPOs are given in Table I, and the values of the cost function H_{SFNN} and mismatch in topological entropy for these partitions is given in Table II.

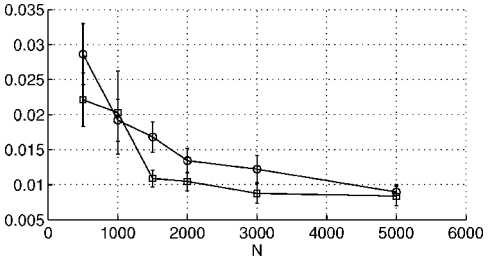

FIG. 11. The mismatch in topological entropy and average final energy from the Henon map as the number of points in the time series is varied. The lower order UPOs were given fixed symbolic codes as in Table I (f). x axis: length of data set (dimensionless); y axis: in circles the topological entropy mismatch δh_T and in squares the average distance to the nearest symbolic neighbors H_{SFNN} (both dimensionless). Plotted are averages over ensembles of 100 randomly chosen initial symbolic configurations using a constant data set.

"canonical" partition. This means that the mean distance from the centroid of each symbolic region and the point in that region increases for these partitions. Thus these partitions will give a higher H_{SFNN} statistic. Also, as we only have a finite amount of data, our methods cannot find the noncanonical partitions as well, and they will typically show a higher δh_T.

As we only use a finite amount of data, some partitions may not be true generating partitions; however, they still have the property that short symbolic words will localize points in phase space. As the number of points used increases, the partition can be found better. We fixed the UPOs in the configuration in Table I(f) and found the partition for several data sizes. The results are seen in Fig. 11. Table II shows H_{SFNN} and δh_T for the partitions found in Fig. 10. We can see the partitions in Figs. 10(a) and 10(b) are both found quite well. The partition in Fig. 10(b) is actually a pre-iterate of the partition in Fig. 10(a), and when no UPOs are fixed, one of these two partitions will be found. The partitions in Figs. 10(c)–10(f) have more topological defects in them, as seen by their larger δh_T; however, they still are minima of H_{SFNN} and localize symbolic regions well. Figure 12 shows how well partition (f) localizes points. Shown in lighter (cyan color online) points are all points which have a given symbol sequence. We can see that even with only four symbols, the points are localized to a very small region of state space. As an example from a measured time series, a partition from data measured from a bubble experiment was found. A constant flow of air was released at the bottom of a fluid tank and the time between bubbles was measured [23]. This time series was embedded in two dimensions, $\boldsymbol{x}_i = (\Delta t_i\ \Delta t_{i+1})$. UPOs were found from the time series using the method described by Schmelcher and Diakonos [10]. In that work, a local linear model of the dynamics is estimated, and then an additional linear transformation is applied which transforms unstable fixed points to stable ones. Then when the transformed model is iterated, the state will converge to the fixed points of compositions of the map, i.e., periodic orbits. For these data, three unstable periodic orbits were found. One orbit was of period 1; one was of period 2, and one was period 4. The symbolic codes of each of these orbits was fixed, and the resulting partitions are shown in Fig. 13.

TABLE I. The symbolic codes of the fixed UPOs in the partitions in Fig. 10.

	Partition					
Periodic point	(a)	(b)	(c)	(d)	(e)	(f)
(0.8839, 0.8839)	0	0	1	1	1	0
(−0.6661, 1.3661)	0	1	1	0	1	0
(1.3661, −0.6661)	1	0	0	1	0	1
(−0.9895, 1.5751)	0	1	0	1	0	0
(0.8935, −0.9895)	1	0	0	1	0	0
(0.3049, 0.8935)	0	0	1	1	0	0
(1.5751, 0.3049)	0	0	0	0	1	1

TABLE II. Values of the cost function H_{SFNN} and mismatch in topological entropy δh_T for the partitions of the Henon attractor shown in Fig. 10. Recall that the particular magnitude of H_{SFNN} need have no relation to that of δh_T, and that H_{SFNN} is not intended to be an estimator of δh_T. The message here is that trends in H_{SFNN}, which is observable, tends to match those of δh_T, and so observed values of H_{SFNN} can be used to rank the "generatingness" of empirically obtained partitions.

Partition	H_{SFNN}	δh_T
(a)	0.0026	0.0041
(b)	0.0025	0.0012
(c)	0.0243	0.0134
(d)	0.0255	0.0316
(e)	0.0201	0.0344
(f)	0.0141	0.0190

V. CONCLUSION

We have described an algorithm to find good partitions for symbolic dynamics from a measured time series. We find a partition by requiring that points which have close symbol sequences should be neighbors in state space. Given this criterion, we define an energy functional in a similar spirit to spin glass problems and find the ground state of this energy. The symbol configuration which minimizes this is the estimated partition. Multiple generating partitions exist simultaneously for any given dynamical system. We partially resolved this degeneracy by including a few low period UPOs with fixed symbolic codes in the energy functional and relax to partitions which agree with these externally inserted codings for the UPOs. Doing this we can find an assortment of alternative partitions to the unconstrained one, but many of these partitions have a significantly higher H_{SFNN} statistic. One reason for this is that as the symbolic regions are now more disjoint, so we can no longer expect the centroids to be close to each point anymore, especially for short symbol sequences. It may be preferable in cases where the symbolic regions are so disjoint to choose symbolic neighbors differently. Perhaps instead of considering the distance to the centroid, the distance to the symbolic neighbor nearest in state space could be used. Our method is related to a recent approach developed by Y. Hirata and co-workers [24]. Their approach uses the centroid locations themselves as the estimated parameters, each with its own symbolic code, and iteratively reassigns symbols to the nearest centroid. In that sense the partition is estimated relative to radial basis centers, more like Ref. [11], though the centers do not move. Centroids are then re-estimated until there is no further change at that level of refinement. Then, the symbolic words would be further extended (i.e., splitting centroids) and the procedure repeated until it has continued to a sufficient depth. Our method, especially with the weighted context tree which is sensitive to all depths, is similar in concept if not detail. Differences are that we permit every point's symbol to vary independently and have a single function to minimize for all time, instead of starting anew at successive depths, and with the tree weighting procedure, a more objective and flexible way to define neighborhoods than terminating the tree splitting at a fixed depth.

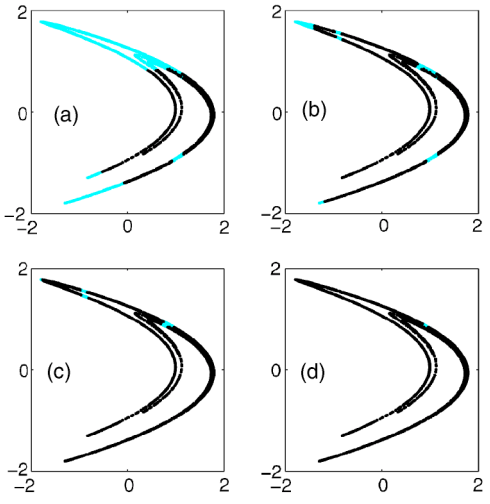

FIG. 12. (Color online) Successive localization of the Henon attractor using the partition found in Fig. 10(f). The lighter (cyan color online) points are those points in the partition whose central symbolic code is (a) 0, (b) 00, (c) 000, (d) 0000. We can See that even with only four symbols the fixed point, whose symbolic code is $\overline{0}$, is well localized.

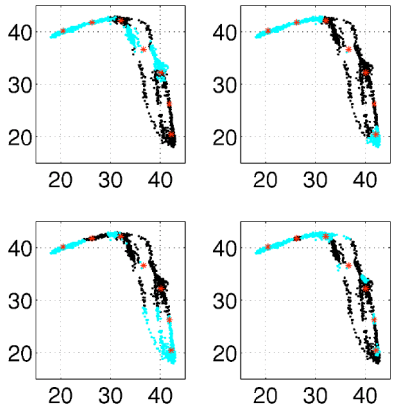

FIG. 13. (Color online) Several different partitions found from a measured time series from a bubble dynamics experiment. The interval between bubble-rise events in a viscous fluid was measured. All axes are time between bubble events: x axis is $T(i)$ (arbitrary units) and y axis is $T(i+1)$ (arbitrary units). Low order UPOs were found from the time series and their symbolic codes were fixed, thus defining different partitions. The UPOs are again shown as stars.

Software and documentation in Fortran 95 is available on the on-line EPAPS archive associated with this work, and at author M.K.'s FTP site, ftp://lyapunov.ucsd.edu/pub/nonlinear/partitions (see Ref. [25]).

ACKNOWLEDGMENTS

We are indebted to discussions with Y. Hirata (University of Western Australia) about generating partition algorithms

and symbolic dynamics in general. This material is based upon work supported by the National Science Foundation under Grant No. 0081636.

APPENDIX: DRAWING RANDOM VARIATES FROM A POWER-LAW DISTRIBUTION

It is surprisingly nontrivial to directly draw random variates from a power-law distribution. Central to τ-EO optimization is an integer K which must be chosen such that

$$P(K) \propto K^{-\tau}. \tag{A1}$$

To compute this we need the cumulative distribution

$$C(l) = A\sum_{K=1}^{l} K^{-\tau}, \tag{A2}$$

where A is the normalization

$$A = \left(\sum_{K=1}^{N} K^{-\tau}\right)^{-1}. \tag{A3}$$

As we have a finite number of points, the above sum only goes to N. An exact method is to tabulate $C(l)$ for $1 \leqslant l \leqslant N$. Then when a random integer is needed, a uniform variate $[0, 1)$ is drawn, and $C(l)$ searched to find the closest match. This is rather slow, and the optimization requires many power-law distributed numbers.

Instead, we use an analytic approximation to $C(l)$, replacing the above sum with the integral

$$C(l) = A\int_{\delta}^{l+\delta} K^{-\tau} dK \tag{A4}$$

with a similar integral defining the normalization.

This approximation is similar to the midpoint method for numerical integration. The approximation is best when the curvature of the function is small—in this case for large l and small τ. As it is equivalent to the midpoint method, δ close to $1/2$ is optimal. Empirically we found $\delta = 0.6$ to give good results. For $l = 1$ and $1 < \tau < 2$ this approximation is accurate to a few percent, with the accuracy rapidly improving for larger l. As an example, for $l = 1, \tau = 1.5, N = 100$ the true cumulative probability is $C(l) \approx 0.4144$, while this approximation gives $C(l) \approx 0.4201$.

With the analytical formula for $C(l)$ the random variable K is found by equating a uniform random variate $\eta \in [0, 1)$ with the value of the cumulative distribution and solving for l. Using the above approximation, $C(l)$ is easily inverted and gives

$$l = [(\eta - 1)\delta^{1-\tau} - \eta(N + \delta)^{1-\tau}]^{1/(1-\tau)} - \delta. \tag{A5}$$

The desired random integer K is then the smallest integer such that $K > l$.

[1] S. Wiggins, *Introduction to Applied Nonlinear Dynamical Systems and Chaos* (Springer, New York, 1990).
[2] E. M. Bollt, T. Stanford, Y.-C. Lai, and K. Zyczkowski, Phys. Rev. Lett. **85**, 3524 (2000); Physica D **154**, 259 (2001).
[3] P. Grassberger and H. Kantz, Phys. Lett. **113A**, 235 (1985).
[4] L. Jaeger and H. Kantz, J. Phys. A **30**, L567 (1997).
[5] R. L. Davidchack, Y.-C. Lai, E. M. Bollt, and M. Dhamala, Phys. Rev. E **61**, 1353 (2000).
[6] J. Plumecoq and M. Lefranc, Physica D **144**, 231 (2000).
[7] J. Glover and A. Mees, J. Circuits Syst. Comput. **3**, 201 (1993).
[8] S. Allie and A. I. Mees, Phys. Rev. E **56**, 346 (1996).
[9] P. So *et al.*, Phys. Rev. E **55**, 5398 (1997).
[10] P. Schmelcher and F. K. Diakonos, Phys. Rev. Lett. **78**, 4733 (1997).
[11] M. B. Kennel and M. Buhl, Phys. Rev. Lett. **91**, 084102 (2003).
[12] D. Lind and B. Marcus, *An Introduction to Symbolic Dynamics and Coding* (Cambridge University Press, Cambridge, UK, 1995).
[13] R. Storn and K. Price, J. Global Optim. **11**, 341 (1997).
[14] S. Boettcher and A. G. Percus, Phys. Rev. Lett. **86**, 5211 (2001).
[15] W. Liebert, K. Pawelzik, and H. G. Schuster, Europhys. Lett. **14**, 521 (1991).
[16] M. B. Kennel and H. D. I. Abarbanel, Phys. Rev. E **66**, 026209 (2002); M. B. Kennel, R. Brown, and H. D. I. Abarbanel, Phys. Rev. A **45**, 3403 (1992).
[17] J. Rissanen, *Stochastic Complexity in Statistical Inquiry* (World Scientific, Singapore, 1989).
[18] M. B. Kennel and A. I. Mees, Phys. Rev. E **66**, 056209 (2002).
[19] B. W. Silverman, *Density Estimation for Statistics and Data Analysis* (Chapman and Hall, London, 1986).
[20] J. H. Friedman, J. L. Bentley, and R. A. Finkel, ACM Trans. Math. Softw. **3**, 209 (1977); R. F. Sproull, Algorithmica **6**, 579 (1991).
[21] M. B. Kennel, J. Shlens, H. D. I. Abarbanel, and E. J. Chichilnisky, Neural Comput. **17**, 1 (2005).
[22] M. Eisele, J. Phys. A **32**, 1533 (1999).
[23] K. Nguyen, C. S. Daw, P. Chakka, M. Cheng, D. D. Bruns, C. E. A. Finney, and M. B. Kennel, Chem. Eng. J. **64**, 191 (1996).
[24] Y. Hirata, K. Judd, and D. Kilminster, Phys. Rev. E **70**, 016215 (2004).
[25] M.K., see EPAPS Document No. E-PLEEE8-71-127503 for (software and documentation). A direct link to this document may be found in the online article's HTML reference section ftp://lyapunov.ucsd.edu/pub/nonlinear/partitions. The document may also be reached via the EPAPS homepage (http://www.aip.org/pubservs/epaps.html) or from ftp.aip.org in the directory/epaps/. See the EPAPS homepage for more information.