The Clustered Causal State Algorithm

Efficient Pattern Discovery for Lossy Data-Compression Applications

Pattern discovery is a potential boon for data compression, but current approaches are inefficient and produce cumbersome pattern descriptions. The clustered causal state algorithm is a new pattern-discovery algorithm that incorporates recent clustering technology.

Advances in sensor and communication technologies have increased the amount of information automated systems can easily acquire. However, this volume of available data is often cumbersome and expensive to transmit and interpret. Compression technologies can exploit lossy techniques to reduce bandwidth and storage costs, but our ability to comprehend compressed data is often impaired by information loss. To avoid this problem, we need methods that can compress data streams' relevant patterns by removing the stochastic noise and transient structure.

Pattern discovery is a new approach to extracting regularities in data. Whereas pattern recognition is based on an existing stable of known patterns, pattern discovery exploits general models of structure to identify even unanticipated regularities. These methods are ideal for unsupervised analysis and compression of real-world information. Real-time data streams, such as video and sensor data, often possess unanticipated or novel patterns. Pattern discovery can extract these patterns, giving us a succinct description we can exploit for compression and interpretation.

However, real-time pattern discovery has been difficult with existing algorithms, especially for compression applications. At the Applied Research Laboratory at Penn State University, our *clustered causal state algorithm* (CCSA) provides a solution to this difficulty. Our work has shown that CCSA performs unsupervised pattern discovery, producing pattern descriptions with computational efficiency and of small enough size for use in data compression, in exchange for a small loss in description fidelity.

1521-9615/06/$20.00 © 2006 IEEE
Copublished by the IEEE CS and the AIP

Mendel Schmiedekamp, Aparna Subbu, and Shashi Phoha
Applied Research Laboratory, Penn State University

Measuring Pattern Complexity

Discovering patterns requires distinguishing complex patterns from data. Several traditional methods exist for measuring this sort of pattern complexity in a data stream. Shannon entropy is the simplest way to measure a data stream's complexity.[1] Intuitively, a data stream's Shannon entropy is a measure of how much information could potentially be within the data. Highly structured data has a low entropy because redundancies eliminate much of the information. Stochastic fluctuations in a data stream increase its Shannon entropy, up to its theoretical maximum, at which point the stream is entirely random.

Another approach is to estimate the Kolmogorov complexity of one part of a data stream.[2] Intuitively, this measures the minimum description of the data

segment. For a random stream, the most succinct description is the stream itself, meaning a high Kolmogorov complexity. Rigidly patterned streams have much more compact descriptions, giving them a much lower complexity.

In both of these methods, random data streams are very complex because the models of patterns they use are based on deterministic data-encoding methods. Shannon entropy is based on the power of the data stream as an encoding strategy, and Kolmogorov complexity is based on the length of the program that generates the input string.

The computational-mechanics approach to measuring complexity is based on the intuition that an entirely random data stream has no patterns and shouldn't be treated as maximally complex.[3] To capture this, computational mechanics uses a minimum statistical model of a data stream, called an ε-machine. A data stream's ε-complexity is simply the size of the associated ε-machine. Both structured and random data tend toward simple ε-machines, whereas a mixture of structure and randomness produces a larger, more complex ε-machine. Estimating ε-complexity distinguishes random data from chaotic data, and, in either case, the ε-machine stores the underlying pattern.

Causal States and ε-Machines

Formally, we define ε-machines via infinite streams of discrete data—namely, symbols. At each point in the data stream, a history exists, composed of all symbols occurring prior, and a future exists, consisting of all symbols occurring afterward. An ε-machine is constructed from causal states, which are themselves constructed from these histories, in the following way:

- A causal state is a collection of all histories with statistically identical futures.
- An ε-machine is a probabilistic finite-state automata whose states are the data stream's causal states and whose transitions are the probability-weighted change in causal state caused by adding a symbol to the head of the history.

By this definition, causal states are the minimal Markovian states for the data stream, meaning that knowledge of the current causal state alone gives the most predictive power possible; by construction, the ε-machine will encode this information into an automata with the least number of states.

Unfortunately, causal states can only be precise with infinite data streams. To make ε-machines useful for physical and engineering applications, researchers have had to develop computationally tenable methods to approximate causal states, and in doing so, create ε-machines for finite segments of discrete data.

Computing Patterns

To approximate a causal state with fixed computing resources and limited data, we must fix both the history and futures we're examining. To reduce the number of possible histories, we limit them to some fixed distance before the current state. Likewise, we constrain futures to include only the immediate future symbol. With these basic limitations, a variety of approaches exists to approximate ε-machines.

Due to ε-machines' analytic usefulness, the first avenue is to algorithmically optimize for accuracy. The *causal state splitting and reconstruction* (CSSR) algorithm is provably optimal for a given history length and data segment.[4,5] This algorithm computes a series of ε-machines, with incrementally increasing history lengths, iteratively modifying the approximate causal states and recomputing their likelihood of correctness at each step.

For some applications, completeness is more important than approaching the ideal ε-machine. This is especially true for critical fault analysis. To address this issue, we can use a variant of ε-machines called *D-Markov models*, for which we treat each unique history of length d as a separate causal state.[6] This ensures that we don't lose any statistical information due to insufficient history length.

Unfortunately, both of these approaches are ill-suited to real-time applications, such as lossy compression. CSSR has an exponentially increasing computational cost, whereas D-Markov models require exponentially increasing space for larger history lengths. To alleviate these two problems, we've developed a new algorithm, CCSA, as part of our work in lossy video compression to approximate ε-machines, for use in real-time and resource-limited applications.

The CSSR Algorithm

The CSSR algorithm[4] approximates ε-machines for a particular history length, L. CSSR iterates through history lengths from 0 up to L, first assuming that the data is best described as a single state machine, and then adding and removing states as needed to accommodate each increase in history length. The program uses the following steps:

- *Initialize*. Consider only the trivial history, of length 0. Thus, the approximation is a single state ε-machine with transitions weighted by the proportion of the data stream composed of each symbol.
- *Homogenize*. For each history length from 1 to L,

first construct a table of the histories of that length, assigning to each a weighted probability vector of future symbols, called a *morph*. Second, calculate the full morph for each state by a weighted average of the morphs of its histories. Finally, going through each history, determine whether the history's morph matches its current state's morph. If so, do nothing. If the history's morph matches another state's morph, move the history to the other state and return to the second part of this step. If the history's morph matches no other state's morph, create a new state and move the history to that state and return to the second part of this step.

- *Determinize*. For each state *s*, examine each symbol *a*, and if two or more states exist with histories reached by adding *a* to a history in *s*, split *s* into that many states, each containing only histories reaching a single state via *a*. Once this is done for all *s* and *a*, output the transitions between the states based on the state morphs and the future state for each *a*.

Figure 1 shows CCSR's output in discovering a particular pattern from stochastically generated data. The *even process* generates either a 0 or 11, thus it always has an even number of 1's. In the ε-machine that the CCSR algorithm generates, the latter two states are similar to the even-process model; states A and B are partially transient, in the sense that once either of them transitions to state C, they can't be reached again.

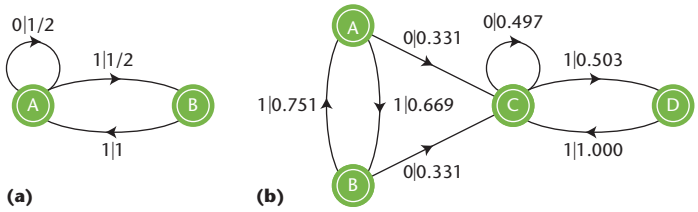


Figure 1. The causal state splitting and reconstruction (CSSR) algorithm in use. (a) The even process: a statistical generator which generates either a 0 or 11, ensuring it always has an even number of 1's. (b) The model that the CSSR produces from even process output.

D-Markov Models

The D-Markov model[6] is an ε-machine-like construction in which each history of length *D* is assigned a separate state (*D* here is the same as *L* in the CSSR and CCSA). Given a fixed *D*, all that's required is to generate a table of histories and their morphs, analogous to the second part of the homogenize step for CSSR. Then, the algorithm calculates the transitions between them directly, producing the D-Markov model. This approach, though clearly suboptimal, provides a machine structure in which each state's physical meaning is invariant under changes in the process dynamics. This makes the method very suitable for unambiguous detection of possible slow-time-scale occurring anomalies. We can then implement anomaly detection by choosing a suitable anomaly measure for the specific process.

Resource-Limited Applications

The major limitation facing CSSR in real-time ε-machine generation is that its accuracy comes at a high computational cost. Its computational complexity is $O(k^{2L+1})+ O(N)$,[4] where k is the alphabet size, L is the maximum length, and N is the total length of the input symbol stream. Intuitively, much of this cost arises from the morph recomputation that occurs when a history moves between states or to an entirely new state.

An additional cost is also incurred due to determinization—the likelihood of splitting states becomes much higher as the number of symbols increases, and, as such, the states can also increase exponentially, approaching k^L. Determinism, while analytically useful, isn't required for reconstruction purposes. Furthermore, although D-Markov models are reasonably time efficient, they produce the worst possible ε-machines from the perspective of compression, reaching the k^L limit by design.

Resource-limited applications, especially those involving compression of real-time data, require an improvement in the state of the art of ε-machine algorithms. For this purpose, we leveraged the existing field of data clustering to produce a time-efficient and compressive ε-machine algorithm.

Clustered Causal State Algorithm

CCSA is based on the hierarchical agglomerative clustering method. Like CSSR, it attempts to describe patterns intrinsic to a process, but it achieves this at a lower computational cost.

Clustering

Clustering involves dividing a data set into groups of similar objects. Similarity, in this case, means that the distance between any two similar objects is less than the distance between any two dissimilar ones. Representing the data by fewer clusters necessarily loses fine details but achieves simplification in representation. The two primary methodologies of unsupervised clustering are *hierarchical* and *partitional*.[7]

Hierarchical clustering follows one of two approaches: *agglomerative methods* start with each object as a cluster and, with each step, combine

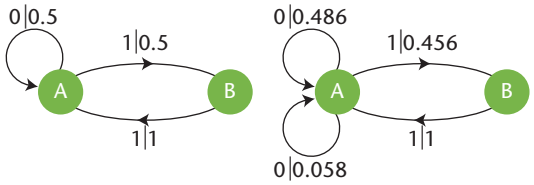


Figure 2. Clustered causal state algorithm results (CCSA). (a) The even process generates either a 0 or 11, so it always has an even number of 1's. (b) The even-process model that the CCSA generates. The extraneous transition looping on state A is an artifact of the finite history length, *L* = 8, indicating the cases in which eight symbols aren't enough to determine if the current sequence of 1's is odd or even.

adjacent clusters to form bigger clusters into only one large cluster. *Divisive methods* begin with one large cluster and proceed to split it into smaller clusters by determining which objects are most dissimilar. Hierarchical clustering algorithms are computationally simple to implement, which is an important factor for compression schemes.

Partitional clustering methods generate a single partition of the data set to try to recover natural groups in the data—that is, given n objects in a d-dimensional space, the problem is dividing a partition of the objects into K clusters, where K might or might not be specified. Partitional clustering methods provide clusters that are approximately optimal for K initial points by revisiting and revising the partitions at the end of each iteration. This ensures that the clusters aren't rigid, but it increases the algorithm's computational complexity as well.

Description of CCSA

The inputs to the CCSA program are the symbol stream, the length L, and the cluster radius r. The algorithm executes in the following steps:

- *Initialization*. A table is constructed of the histories of length L, assigning to each a morph.
- *Clustering*. For each history, compare the Euclidean distance between the history's morph and each state's archetype morph. If there is no state within r of the history, then introduce a new state and use the history's morph as the archetype for that state. This occurs automatically for the first history. If the history is within r of a state, then add the history to the first such state.
- *Finalization*. Compute the states' transitions by assigning a probability to each symbol and next state pair based on the distribution of histories within the state. This explicitly permits nondeterminism, in which the same symbol can lead to two or more next states from a single prior state.

Radius and Maximum States

Because CCSA uses an agglomerative clustering approach, the maximum number of states for the final machine is bounded based on the choice of r and the application's number of symbols, k. In particular, the maximum number of states will be precisely the maximum number of times that a new state can be added to the space of possible morphs, a k-dimensional space.

Given that morphs are probability vectors, the space of possible morphs is a k-simplex with unit sides, a k-dimensional analog of a unit triangle. No state can be closer than r to any other state, meaning that each state's center point can at best be distributed as vertices of a compact tesselation in the simplex. The most compact of these tessellations is tiling with similar simplexes.

The largest number of vertices the k-simplex contains from this tessellation is simply the figurate number, a likewise higher-dimensional analog of triangular numbers for the dimension and side length equal to $n = [1/r] + 1$.[8] The figurate number in this case is $n(n + 1) \ldots (n + k)/k!$. For many applications, we use this formula to find a desired upper bound for the ε-machine. In addition, as k increases, a little algebra shows that this formula will follow: the maximum number of states equals $O(k^n)$, meaning valid choices of r continue to exist even for very large values of k.

Example of CCSA

We applied CCSA to the even process we discussed previously. Figure 2 shows the results. We used a history of length 8, with the radius $R = 0.4661$. The extra transition on state 0 is generated because, unlike CSSR, CCSA doesn't iterate up to L.

CCSA Time Complexity Analysis

In CCSA's initialization stage, it creates a hash table of input histories and constructs look-up tables of symbol encodings with complexities of $O(N)$ and $O(k)$, respectively. The clustering stage builds the causal model's states, which is a function of the number of histories in the hash table and the number of states (the program compares each history against existing state archetypes to assign it to a state). If s is the number of final states derived, then this stage's time complexity is $O(\min(k^L, N) \cdot s)$. As just shown, $s = O(k^n)$, where $n = [1/r] + 1$. CCSA's last remaining stage

is the finalization process in which it determines the transitions between states. This stage's time complexity is $O(s^2 \cdot k)$. Putting it all together, we get

$$T(N, k, L) = O(N) + O(k) + O(\min(k^L, N) \cdot k^n) + O(k^{2n+1})$$

$$T(N, k, L) = O(N + k + k^{2n+1} + \min(k^L, N) \cdot k^n)$$

$$T(N, k, L) = O(N + k^{2n+1} + \min(k^{L+n}, N \cdot k^n)).$$

We can further simplify this for $k^L > N$, $T(N, k) = O(k^n \cdot N + k^{2n+1})$, and for $k^L \leq N$, $T(N, k) = O(k^n \cdot N + k^{2n+1}) = O(N)$ (for $N >> k$ and $k^L >> k^n$).

Note, that the requirement $k^L >> k^n$ stems from the compressive nature of the ε-machines this algorithm will produce. From this analysis, we note that the dependence of the algorithm's asymptotic time complexity on the max history length L has been removed.

CCSA Advantages

CCSA has the distinct advantage of polynomial computational complexity versus the CSSR algorithm's exponential complexity in L. In a large symbol-space application, such as the image-compression application we discuss later, this improvement is marked. Using the CCSR algorithm to compress the image required three to five minutes, depending on the image. Using CCSA, compression took a few seconds. The models generated also demonstrated significant improvements: CSSR models, which are necessarily smaller than D-Markov models, often produced no compression at all, and at best reduced the image by a factor of three. With CCSA, the models could reliably generate 10- to 20-fold compressions.

Comparing Discovered Patterns

CCSA efficiently produces compact ε-machine approximations, but these improvements come with a cost in accuracy. This trade-off is worth making only if CCSA produces comparable results to CSSR (we need consider only CSSR because its optimality is equal to or better than that of D-Markov models). To answer this question, we must introduce a metric to compare causal models' reconstruction quality, the primary concern in data-compression applications.

Statistical Language Measure

Both CSSR and CCSA generate approximations of the ε-machine that describes a data stream. Although these ε-machines describe the same data, their states and internal structure can be quite different. Unlike D-Markov models, which always use the same states for a given history length, CSSR- and CCSA-generated ε-machines don't lend themselves to automata methods of comparison. Instead, we must perform comparisons entirely on the language level, by comparing the statistics of the strings these machines generate.

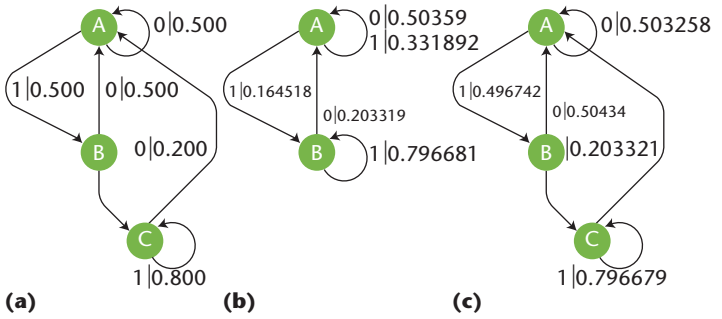


Figure 3. A three-state deterministic hidden Markov model. (a) The original process machine is deterministic because no transitions in any state with the same symbol travel to different states. (b) The clustered causal state algorithm (CCSA) derived machine is nondeterministic, as opposed to (c) the causal state splitting and reconstruction (CSSR) algorithm-derived machine.

The *statistical language measure* generates a metric (called a *stat metric*) that describes the distance between the patterns the two ε-machines generated. If the machine that CCSA generates differs significantly from the original process model, or from the CSSR machine, it won't be able to capture all the patterns, and the stat metric will be high.

The stat metric algorithm takes as input the two machines to be compared, the depth up to which the machines are to be compared, and a start state for each machine. It then iterates through the machines, generating a sequence of morphs to that depth for both. These morphs are the probability vectors of a particular symbol being output at that step of the machines, with the machine having started in the given start state. At each step, the algorithm accumulates the normalized Euclidean distance between the two morphs, and the metric's final output is the normalized average of these distances.

The stat metric thus measures the probability of symbol streams generated by the two machines being different. The lower the metric, the more accurate the CCSA-generated machine is to the original process model and the CSSR machine.

Comparison Using the Stat Metric

Consider the three-state deterministic hidden Markov model. Figure 3a shows the original pro-

Table 1. Stat metric for different process models.

Process model	Original and CCSA*	Original and CSSR*	CCSA and CSSR
Three-state process model	0.00565899	0.00568011	3.283E-05
Misiurerwicz	0.0054871	0.0054648	9.33E-05
Even process	0.00282952	0.00202206	0.0008225
Noisy 2 period	0.00303	0.00303	2.61E-05
Nondeterministic	0.0022647	0.00190953	0.0003756
m 4	0.007556	0.048069	0.0476938
m 8	0.009185	0.00937152	0.0006746

**CCSA: clustered causal state algorithm; CSSR: causal state splitting and reconstruction algorithm*

Table 2. Comparison of the number of states generated.

Process	Alphabet size	Original states	CCSA* states	CSSR* states
Misiurerwicz	2	4	2	8
Even process	2	2	2	2
Noisy 2 period	2	2	2	2
Nondeterministic	2	2	2	5
m 4	4	4	3	11
m 8	8	4	6	17

**CCSA: clustered causal state algorithm; CSSR: causal state splitting and reconstruction algorithm*

cess machine, and Figures 3b and 3c depict the machine structure generated by CCSA and CSSR, respectively. Note the nondeterministic-equivalent machine CCSA returned as opposed to the deterministic machine that CSSR returned.

The top row of Table 1 shows the stat metric values for these machines, to a depth of 10,000. We see from the table that CCSA derives a machine very close to the CSSR machine up to a fifth order of magnitude but at a much lower computational time. Thus, CCSA is better suited for real-time purposes because it achieves nearly the same results as CSSR but much faster and with more compact output.

We applied this same comparison to a variety of models. For each process model, we computed the stat metric between the original model and CCSA, the original model and CSSR, and the CCSA and CSSR models. In nearly all cases, we found that CCSA performed similar to CSSR up to a third order of magnitude. Table 1 summarizes the results, and Figure 4 shows the process models' structure.

Nondeterministic Machine Generation

Of special interest to compression applications is the fact that CCSA can generate compact nondeterministic machines whereas CCSR can't. To compare both algorithms, Table 2 tabulates the number of states each algorithm returns for each of the processes in Table 1. For a binary alphabet, simple deterministic machines capture the model accurately; for nondeterministic machines with large alphabets, however, the number of states that CCSR generates grows rapidly.

Applications

Pattern-discovery algorithms such as CSSR and CCSA have a wealth of applications for data compression. As unsupervised methods for extracting underlying regularities from data streams, ε-machines can directly encode a large quantity of data or help guide the decisions made in lossy compression.

Image Representation

As a comparison of the actual viability of using CCSA instead of CCSR, we used both as part of an image-compression scheme. We achieved image compression by applying a 2D multiresolution Daubechies wavelet transform[9,10] to the image. We symbolized each scale of the wavelet transform separately, and then used these symbol streams to derive descriptive causal models. We used the alphabet size $K = 15$, which makes the CCSR program's computational complexity quite high.

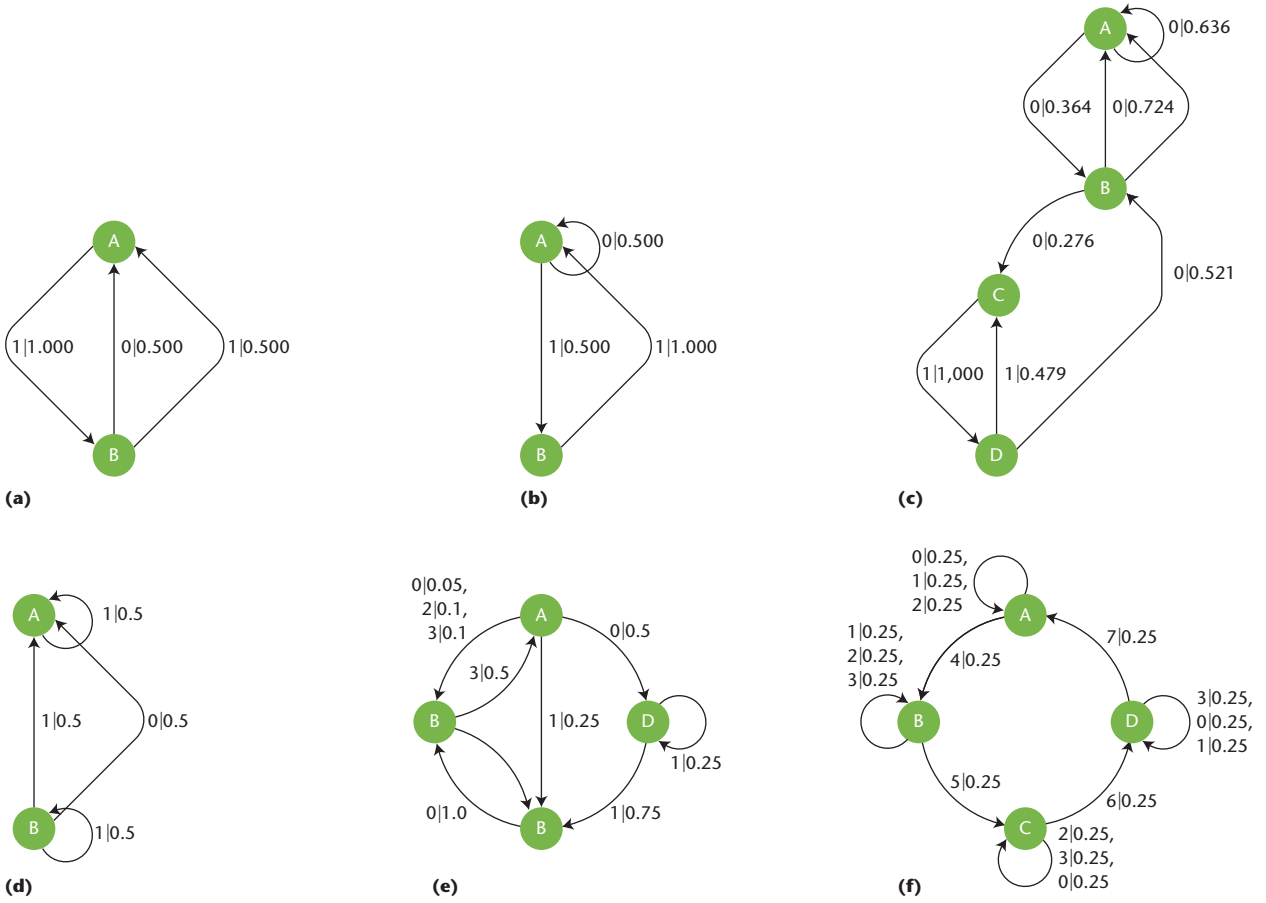


Figure 4. Process model structure. For various process models, including (a) noisy 2 period, (b) even process, (c) Misiurerwicz, (d) nondeterministic, (e) m 4, and (f) m 8, we computed the stat metric between the original model and the clustered causal state algorithm (CCSA), the original model and the causal state splitting and reconstruction (CSSR) algorithm, and CCSA and CCSR.

CCSA's computational complexity, on the other hand, doesn't significantly increase.

As Figure 5 shows, the results didn't differ greatly. To get a quantitative measure of the images' reconstruction quality that CCSA and CSSR algorithm generated, we computed the mean difference with the original for each reconstructed image (see Table 3).

Distinguishing Patterns in Lossy Data Compression

With image representation, we use ε-machines solely to describe individual data streams—namely, the panels of the multiresolution wavelet decomposition. However, because ε-machines describe not only a data stream but also a statistical language, we can compare them and determine their distance using the stat metric discussed earlier. Although we developed this metric to validate the al-

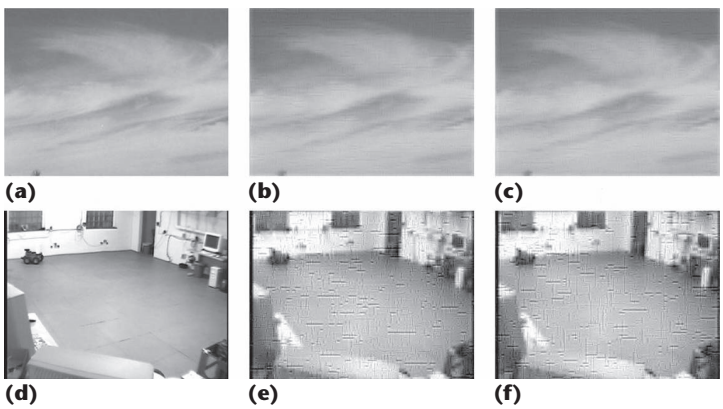
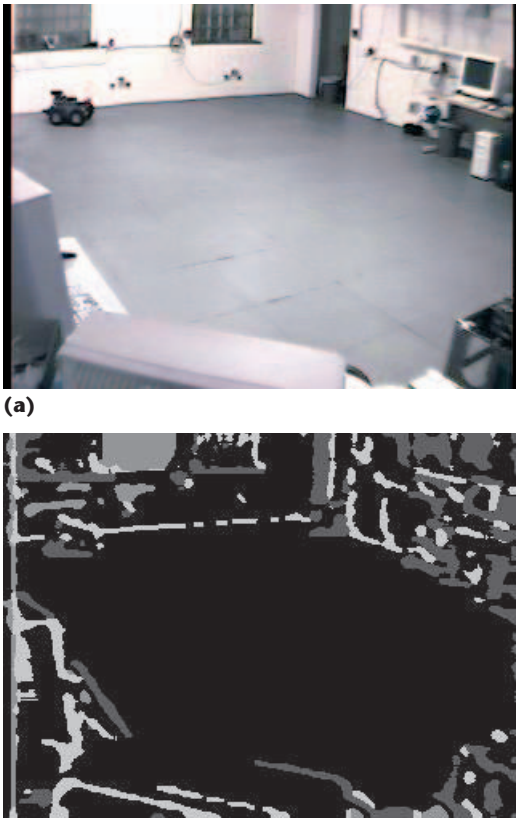


Figure 5. Image-compression results. We compressed (a) the original image and reconstructed it using using (b) the clustered causal state algorithm (CCSA) and (c) the causal state splitting and reconstruction (CSSR) algorithm. (d-f) The same reconstruction with a second image.

Table 3. Absolute mean difference of reconstructed images.

Image	Difference from CSSR* image	Difference from CCSA* image
Image 1 in Fig. 5	3.7170	3.8128
Image 2 in Fig. 5	16.3490	16.4221

** CSSR: causal state splitting and reconstruction algorithm; CCSA: clustered causal state algorithm*



(a) **(b)**

Figure 6. Image texture segmentation. We developed an ε-machines-based method to take (a) an original image and classify (b) patterns in the dynamics of an image's pixels around each one.

gorithms used to generate these machines, we can also use it to compare data streams or portions of data streams.

Within large data sets, the data's low-level patterns often aren't important; instead, the important information is the change from one low-level pattern to another. By breaking the data down into consecutive regions, called *windows*, we can compute an ε-machine for each window and then compare it with preceding and subsequent windows using the stat metric. Large differences indicate that the data set underwent a phase change in that window, entitling that window higher fidelity in compression.

In addition, unlike pattern recognition, pattern-discovery methods allow us to identify these phase changes without a prior model or human supervision. Thus, we can reduce a large quantity of data to a series of critical moments, while using individual ε-machines to succinctly describe the remainder.

As an example of this application, we developed an ε-machine-based method of image texture segmentation. This algorithm identifies textures as underlying patterns in the dynamics of an image's pixels within a window. Figure 6 shows an output from this method.

Pattern discovery is a class of problems that are highly relevant for lossy compression. Extracting novel patterns without human supervision enables compression to retain relevant information that might otherwise be lost by conventional lossy compression. CCSA is an efficient pattern-discovery algorithm designed for real-time compression applications, using the principles of data clustering. Although less accurate than offline algorithms such as CSSR and D-Markov models, CCSA generates automata models of patterns with only a small comparative loss.

Future improvements to CCSA include incorporating alternate clustering technology, such as *k*-means[11] or divisive hierarchical clustering.[12] Motivated by real-time windowing applications, a windowed CCSA could dynamically build models of a particular time span and could also compare them with earlier observations, as well as those taken over broader or narrower time spans. This opens a wide area of potential applications for the real-time identification and classification of unanticipated patterns.

Acknowledgments

This material is based on work supported and administered by the US Office of Naval Research under the Semantics Source Coding project award no. N00014-03-1-0231. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

References

1. J.R. Pierce, *An Introduction to Information Theory*, 2nd ed., Dover Books, 1980.

2. M. Li and P.M.B. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, 1993.
3. C.R. Shalizi and J.P. Crutchfield, "Computational Mechanics: Pattern and Prediction, Structure and Simplicity," *J. Statistical Physics*, vol. 104, nos. 3–4, 2001, pp. 819–881.
4. C.R. Shalizi, K.L. Shalizi, and J.P. Crutcheld, *An Algorithm for Pattern Discovery in Time Series*, tech. report 02-10-060, Santa Fe Inst., 2002; www.arxiv.org/abs/cs.LG/0210025.
5. C.R. Shalizi and K.L. Shalizi, "Blind Construction of Optimal Nonlinear Recursive Predictors for Discrete Sequences," *Proc. 20th Conf. Uncertainty in Artificial Intelligence*, ACM Press, 2004, pp. 504–511.
6. A. Ray, "Symbolic Dynamic Analysis of Complex Systems for Anomaly Detection," *Signal Processing*, vol. 84, no. 7, 2004, pp. 1115–1130.
7. L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, 1990.
8. L.E. Pickson, *History of the Theory of Numbers Vol II: Diophantine Analysis*, Chelsea Publishing, 1966.
9. S.G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, 1998, pp. 674–693.
10. I. Daubechies, "Ten Lectures on Wavelets," *Proc. Conf. Board of Mathematical Sciences-National Science Foundation Regional Conf. Series in Applied Math.*, SIAM, vol. 61, 1992.
11. J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. 5th Berkeley Symp. Mathematical Statistics*, Univ. of California Press, vol 1, 1967, pp. 281–297.
12. R. Ng, and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining," *Proc. Int'l Conf. Very Large Data Bases* (VLDB 94), VLDB Endowment, 1994, pp.144–155.

Mendel Schmiedekamp *is technical lead for the Intelligent Control Department of the Information Science and Technology Division at the Applied Research Laboratory at Penn State University. His research interests include efficient pattern-discovery algorithms, mission-oriented compression, and formal and pragmatic mechanisms for autonomous network collaboration. Schmiedekamp has an MS in computer science and engineering from Penn State University. He is codeveloper of the Common Control and Communication Language. Contact him at mxs485@psu.edu.*

Aparna Subbu *is a PhD student in mechanical engineering at Penn State University. Her research interests include pattern discovery and lossy video image compression. Subbu has an MS in electrical engineering from Penn State University. Contact her at aparnasubbu@psu.edu.*

Shashi Phoha *is the head of the Information Science and Technology Division at the Applied Research Laboratory at Penn State University and director of the Information Technology Laboratory, US National Institute of Standards and Technology. Her research interests include information sciences that bring together ideas from logic, operations research, formal languages, automata theory, dynamic systems, and control theory in diverse applications. Phoha has a PhD in Nonlinear Prediction from Michigan State University, She is a member of the IEEE. Contact her at sxp26@psu.edu.*