KNOWLEDGE MEDIA INSTITUTE

Bayesian Clustering by Dynamics

Marco Ramoni[1] **Paola Sebastiani**[2] **Paul Cohen**[3] **John Warwick**[3] **James Davis**[3]

[1] Knowledge Media Institute, The Open University, Milton Keynes MK7 6AA, United Kingdom

[2] Statistics Department, The Open University, Milton Keynes MK7 6AA, United Kingdom

[3] Department of Computer Science, University of Massachusetts, Amherst 01003, MA, USA

KMi-TR-78

February 1999

Bayesian Clustering by Dynamics

Marco Ramoni[1] **Paola Sebastiani**[2] **Paul Cohen**[3] **John Warwick**[3] **James Davis**[3]

[1] Knowledge Media Institute, The Open University, Milton Keynes MK7 6AA, United Kingdom

[2] Statistics Department, The Open University, Milton Keynes MK7 6AA, United Kingdom

[3] Department of Computer Science, University of Massachusetts, Amherst 01003, MA, USA

Abstract

This paper introduces a Bayesian method for clustering dynamic processes. The method models dynamics as Markov chains and then applies an agglomerative clustering procedure to discover the most probable set of clusters capturing different dynamics. To increase efficiency, the method uses an entropy-based heuristic search strategy. An experiment suggests that the method is very accurate when applied to artificial time series in a broad range of conditions. When the method is applied to clustering simulated military engagements and sensor data from mobile robots, it produces clusters that are meaningful in the domains of application.

Keywords: Bayesian Learning, Clustering, Time series, Markov Chains, Heuristic search, Entropy.

Reference: KMi Technical Report KMi-TR-78, Knowledge Media Institute, The Open University, Milton Keynes, United Kingdom, February 1999.

Address: Marco Ramoni, Knowledge Media Institute, The Open University, Milton Keynes, United Kingdom MK7 6AA. PHONE: +44 (1908) 655721, FAX: +44 (1908) 653169, EMAIL: m.ramoni@open.ac.uk, URL: http://kmi.open.ac.uk/people/marco.

Contents

1. Introduction

Suppose one has a set of univariate time series generated by one or more unknown processes that have characteristic dynamics. *Clustering by dynamics* is the problem of grouping time series into clusters so that the elements of each cluster have similar dynamics. For example, if a batch of time series represents sensory experiences of a mobile robot, clustering by dynamics might find clusters corresponding to abstractions of sensory inputs [15]. If a batch contains a time series of sistolic and diastolic phases, clustering by dynamics might find clusters corresponding to the pathologies of the heart. Sound patterns can be clustered by dynamics, also, and this is one way to discover patterns corresponding to words in the speech signal [8]. An open problem in unsupervised learning is to automatically construct explanations of data [17]. Clustering by dynamics can reduce a large batch of time series to a small number of clusters, where each cluster contains time series that have similar dynamics, thus simplifying the task of explaining the data.

This paper presents a Bayesian algorithm for clustering by dynamics. Our algorithm learns Markov Chain (MC) representations of the dynamics in the time series and then clusters similar time series to learn prototype dynamics. A MC represents a dynamic process as a transition probability matrix. For each time series observed on a variable X, we construct one such matrix. Each row in the matrix represents a state of the variable X, and the columns represent the probabilities of transition from that state to each other state of the variable on the next time step. The result is a set of conditional probability distributions, one for each state of the variable X, that can be learned from a time series. A transition matrix is learned for each time series in a training batch of time series. Next, a Bayesian clustering algorithm groups time series that produce similar transition probability matrices. The task of the clustering algorithm is two-fold: to find the set of clusters that gives the best partition according to some measure, and to assign each MC to one cluster. A partition is an assignment of MCs to clusters such that each time series belongs to exactly one cluster. The algorithm uses the posterior probability of a partition — i.e. the probability of a partition given the sample time series — as scoring metric and, to increase search efficiency, the method uses an entropy-based heuristic search strategy.

Bayesian methods for clustering were pioneered by Cheeseman; an overview is in [2]. However, as far as we know, this article presents the first exact Bayesian methods for clustering dynamic processes by modeling the dynamics via MCs. A Bayesian approach is particularly well suited to clustering by dynamics because it provides a principled way to integrate prior and current evidence. Furthermore, because the posterior probability of a partition is our scoring metric, we avoid the problem of increasing the overall probability of errors that plagues classical statistical methods [3].

While there are similarities between clustering by dynamics and learning Hidden Markov Models (HMMs), the former problem is different and somewhat simpler. An HMM has one probability distribution for the symbols emitted by each state, and also a matrix of proba-

bilities of transitions between states [11]. In our problem we fit a fully observable Markov model to each episode and then we search for the partition of these models into clusters that has maximum probability. Thus our algorithm is more closely related to other approaches to clustering by dynamics, such as [16, 13], than it is to HMMs. Oates has developed a powerful method for clustering by dynamics based on Dynamic Time Warping [8]. In this work, the "stretch" required to warp one multivariate time series into another is used as a similarity metric for agglomerative clustering. Because Dynamic Time Warping works on an entire time series, it is a good metric for comparing the *shape* of two series. The algorithm we discuss in this paper assumes the series are Markov chains, so clustering is based on the similarity of transition probability matrices, and some information about the shape of the series is lost. While there are undoubtedly applications where the shape of a time series is its most important feature, we find that Bayesian clustering of MCs produces meaningful groupings of time series, even in applications where the Markov assumption is not known to hold (see Section 5). Our algorithm is also very efficient and accurate, and it provides a way to include prior information about clusters and a heuristic search for the maximum likelihood clustering.

The reminder of this paper is organized as follows. We describe the Bayesian clustering algorithm in Section 2. Section 3 gives details of the algorithm, and an evaluation of properties of the algorithm on several sets of synthetic data is in Section 4. We apply the algorithm to two real data sets in Section 5. The goal of the first application is the identification of dynamics in a military scenario. The second applications creates prototype dynamics of sensory inputs in a mobile robot.

2. Clustering Markov Chains

Suppose we have a batch of m time series that record the values $1, 2, ..., s$ of a variable X. The goal is to identify time series that exhibit similar dynamics. Consider, for example, the plot of three time series in Figure 1. Each records the values of a variable with five states — labeled 1 to 5 — in 50 time steps. It is not obvious that the three time series are observations of the same process. However, when we explore the underlying dynamics of the three series more closely, we find, for example, that state 2 is frequently followed by state 1, and state 3 is followed disproportionately often by state 1. We are interested in extracting these types of similarities among time series. (The problem of finding dependencies between states in time series has been studied by [10, 9, 6, 7]; the current work is unique in its approach to *clustering* time series by the dependency structure that holds among states, and in particular, its ability to differentiate time series that have different dependency structures.) To cluster time series by their dynamics, we model the dynamics of time series' as Markov chains (MCs). For each time series, we estimate a transition matrix from data and then we cluster transition matrices with similar dynamics.

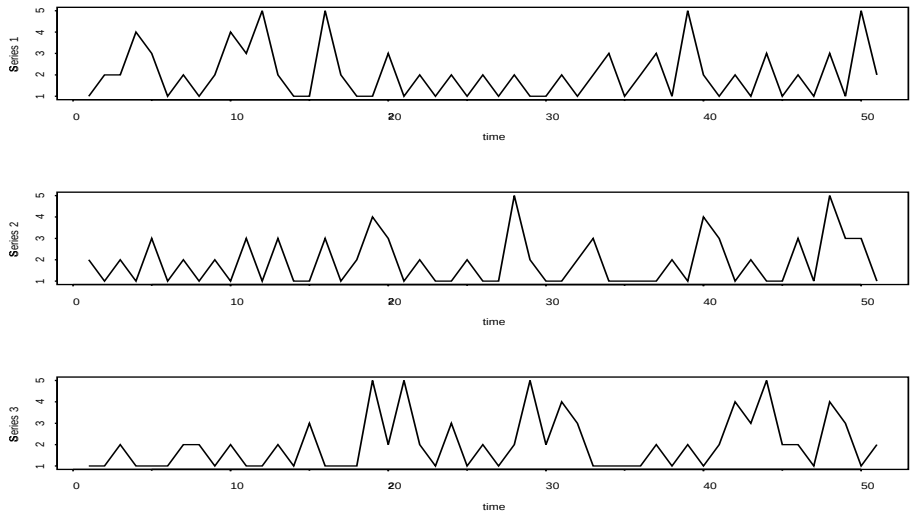


Figure 1: Plot of three time series

2.1 Learning Markov Chains

Suppose we observe a time series $x = (x_0, x_1, x_2, ..., x_{i-1}, x_i, ..)$, where each x_i is one of the states $1, ..., s$ of a variable X. The process generating the sequence x is a MC if the conditional probability that the variable visits state j at time t, given the sequence $(x_0, x_1, x_2, ..., x_{t-1})$, is only a function of the state visited at time $t-1$ [14]. Hence, we write $p(x_t = j|(x_0, x_1, x_2, ..., x_{t-1})) = p(x_t = j|x_{t-1})$ for any x_t in x. In other words, the probability distribution of the variable X at time t, say X_t, is *conditional independent* of the values $(x_0, x_1, x_2, ..., x_{t-2}))$, once we know x_{t-1}. This conditional independence assumption allows us to represent a MC as a vector of probabilities $p_0 = (p_{01}, p_{02}, ..., p_{0s})$, denoting the distribution of X_0 (the initial state of the chain) and a matrix P of transition probabilities, where $p_{ij} = p(X_t = j|X_{t-1} = i)$.

$$P = (p_{ij}) = \begin{array}{c|cccc} & X_t & & & \\ X_{t-1} & 1 & 2 & \cdots & s \\ \hline 1 & p_{11} & p_{12} & \cdots & p_{1s} \\ 2 & p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & & \cdots & & \\ s & p_{s1} & p_{s2} & \cdots & p_{ss} \end{array}$$

Given a time series generated from a MC, we can estimate the probabilities p_{ij} from the data and store them in the matrix $\hat{P}$. The assumption that the generating process is a MC implies that only pairs of transitions $X_{t-1} = i \to X_t = j$ are informative, where a transition $X_{t-1} = i \to X_t = j$ occurs when we observe the pair $X_{t-1} = i, X_t = j$ in the time series. Hence, the time series can be summarized into an $s \times s$ contingency table containing the frequencies of transitions $n_{ij} = n(i \to j)$ where, for simplicity, we denote the transition $X_{t-1} = i \to X_t = j$ by $i \to j$. The frequencies n_{ij} are used to estimate the transition probabilities p_{ij} characterizing the dynamics of the process that generated the data.

However, the observed transition frequencies n_{ij} may not be the only source of information about the process dynamics. We may also have some background knowledge that can be represented in terms of a hypothetical time series of length $\alpha + 1$ in which the α transitions are divided into α_{ij} transitions of type $i \to j$. This background knowledge gives rise to a $s \times s$ contingency table, homologous to the frequency table, containing these hypothetical transitions α_{ij} that we call *hyper-parameters*.

A Bayesian estimation of the probabilities p_{ij} takes into account this prior information by augmenting the observed frequencies n_{ij} by the hyper-parameters α_{ij} so that the *Bayesian estimate* of p_{ij} is

$$\hat{p}_{ij} = \frac{\alpha_{ij} + n_{ij}}{\alpha_i + n_i} \tag{1}$$

where $\alpha_i = \sum_j \alpha_{ij}$ and $n_i = \sum_j n_{ij}$. Thus, α_i and n_i are the numbers of times the variable X visits state i in a process consisting of α and n transitions, respectively. By writing Equation 1 as

$$\hat{p}_{ij} = \frac{\alpha_{ij}}{\alpha_i}\frac{\alpha_i}{\alpha_i + n_i} + \frac{n_{ij}}{n_i}\frac{n_i}{\alpha_i + n_i} \tag{2}$$

we see that $\hat{p}_{ij}$ is an average of the classical estimate n_{ij}/n_i and of the quantity α_{ij}/α_i, with weights depending on α_i and n_i. Rewriting of Equation 1 as 2 shows that α_{ij}/α_i is the estimate of p_{ij} when the data set does not contain transitions from the state i — and hence $n_{ij} = 0$ for all j — and it is therefore called the *prior* estimate of p_{ij}, while $\hat{p}_{ij}$ is called the *posterior estimate*. The variance of the prior estimate α_{ij}/α_i is given by $(\alpha_{ij}/\alpha_i)(1 - \alpha_{ij}/\alpha_i)/(\alpha_i + 1)$ and, for fixed α_{ij}/α_i, the variance is a decreasing function of α_i. Since small variance implies a large precision about the estimate, α_i is called the *local precision* about the conditional distribution $X_t|X_{t-1} = i$ and it indicates the level of confidence about the prior specification. The quantity $\alpha = \sum_i \alpha_i$ is the *global* precision, as it accounts for the level of precision of all the s conditional distributions. Further details may be found in [12].

When n_i is large relative to α_i, so that the ratio $n_i/(\alpha_i + n_i)$ is approximately 1, the Bayesian estimate reduces to the classical estimate given by the ratio between the number n_{ij} of times the transition has been observed and the number n_i of times the variable has visited

N=

	1	2	3	4	5
1	3	12	3	0	3
2	11	1	2	2	0
3	6	0	0	0	1
4	0	0	2	0	0
5	0	4	0	0	0

$\Rightarrow$ $\hat{P}$=

	1	2	3	4	5
1	0.15	0.55	0.15	0.01	0.15
2	0.66	0.07	0.13	0.13	0.01
3	0.78	0.03	0.03	0.03	0.15
4	0.07	0.07	0.73	0.07	0.07
5	0.04	0.84	0.04	0.04	0.04

Table 1: Observed and learned transition matrices for the first time series in Figure 1.

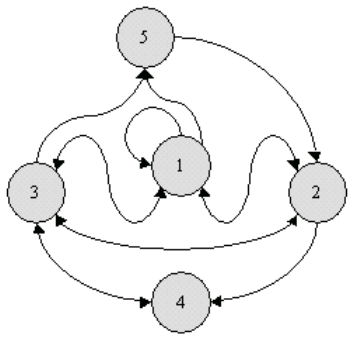


Figure 2: Markov Chain induced from data.

state i. In this way, the estimate of the transition probability p_{ij} is approximately 0 when $n_{ij} = 0$ and n_i is large. The variance of the posterior estimate p_{ij} is $\hat{p}_{ij}(1-\hat{p}_{ij})/(\alpha_i+n_i+1)$ and, for fixed $\hat{p}_{ij}$, it is a decreasing function of α_i+n_i, the local precision augmented by the sample size n_i. Hence, the quantity $\alpha_i + n_i$ can be regarded as a measure of the *confidence* in the estimates: the larger the sample size, the stronger the confidence in the estimate.

Example 1 Table 1 reports the frequencies of transition n_{ij} $i,j = 1,...,5$ observed in the first time series in Figure 1 and the learned transition matrix when the prior global precision is $\alpha = 5$ and $\alpha_{ij} = 1/5$. The matrix $\hat{P}$ describes a dynamic process characterized by transitions among states 1, 2 and 3 while states 4 and 5 are visited rarely. Note that although the observed frequency table is sparse, as 14 transitions are never observed, null frequencies of some transitions do not induce null probabilities. The small number of transitions observed from state 3 ($n_3 = 7$), state 4 ($n_4 = 2$) and state 5 ($n_5 = 4$) do not rule out, for instance, the possibility of transitions from 3 to either 2, 3 or 4. A summary of the essential dynamics is in Figure 2 in which double headed paths represent mutual transitions. Transitions with probability smaller than 0.05 are not represented.

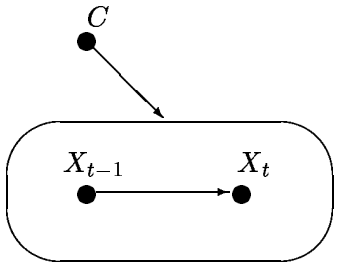


Figure 3: Graphical representation of a clustering model M_c.

2.2 Clustering

The second step of the learning process is an unsupervised agglomerative clustering of MCs on the basis of their dynamics. The available data is a set $S = \{S_i\}$ of m time series. The task of the clustering algorithm is two-fold: find the set of clusters that gives the best partition according to some measure, and assign each MC to one cluster. A partition is an assignment of MCs to clusters such that each time series belongs to exactly one cluster.

We regard the task of clustering MCs as a Bayesian model selection problem. In this framework, the model we are looking for is the most probable way of partitioning MCs according to their similarity, *given* the data. We use the probability of a partition given the data —- i.e. the *posterior probability* of the partition — as a scoring metric and we select the model with maximum posterior probability. Formally, this is done by regarding a partition as a hidden discrete variable C. Each state C_k of C represents a cluster of time series, and hence determines a transition matrix. Graphically, the partition of the transition matrices, that we can learn from S, can be represented as in Figure 3. The directed link from the node C and the node containing the MC represents the dependence of the transition matrix $X_t|X_{t-1}$ on C. The number c of states of C is unknown, but the number m of available MCs imposes an upper bound, as $c \leq m$. Each partition identifies a model M_c, and we denote by $p(M_c)$ its prior probability. By Bayes' Theorem, the posterior probability of M_c, given the sample S, is

$$p(M_c|S) = \frac{p(M_c)p(S|M_c)}{p(S)}.$$

The quantity $p(S)$ is the marginal probability of the data. Since we are comparing all the models over the same data, $p(S)$ is constant and, for the purpose of maximizing $p(M_c|S)$, it is sufficient to consider $p(M_c)p(S|M_c)$. Furthermore, if all models are *a priori* equally likely, the comparison can be based on the *marginal likelihood* $p(S|M_c)$, which is a measure of how likely the data are if the model M_c is true.

The quantity $p(S|M_c)$ can be computed from the marginal distribution (p_k) of C and the conditional distribution (p_{kij}) of $X_t|X_{t-1} = i, C_k$ — C_k is the cluster membership — using a well-known Bayesian method [4]. Let n_{kij} be the observed frequencies of transitions $i \to j$ in cluster C_k, and let $n_{ki} = \sum_j n_{kij}$ be the number of transitions observed from state i in cluster C_k. We define m_k to be the number of time series that are assigned to cluster C_k. The observed frequencies (n_{kij}) and (m_k) are the data required to learn the probabilities (p_{kij}) and (p_k) respectively and, together with the prior hyper-parameters α_{kij}, they are all that is needed to compute the probability $p(S|M_c)$, computed as

$$p(S|M_c) = f(S, C) f(S, X_{t-1}, X_t, C) \tag{3}$$

Intuitively, the first quantity is the likelihood of the data, if we assume that we can partition the m MCs into c clusters, and it is computed as

$$f(S, C) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + m)} \prod_{k=1}^{c} \frac{\Gamma(\alpha_k + m_k)}{\Gamma(\alpha_k)}.$$

The second quantity measures the likelihood of the data when, conditional on having c clusters, we uniquely assign each time series to a particular cluster. This quantity is given by

$$f(S, X_{t-1}, X_t, C) = \prod_{k=1}^{c} \prod_{i=1}^{s} \frac{\Gamma(\alpha_{ki})}{\Gamma(\alpha_{ki} + n_{ki})} \prod_{j=1}^{s} \frac{\Gamma(\alpha_{kij} + n_{kij})}{\Gamma(\alpha_{kij})}$$

where $\Gamma(\cdot)$ denotes the Gamma function. Equation 3 is derived directly from the results in [4] by treating a MC as *child* variable of C, using the popular terminology of Bayesian Network. Once created, the transition probability matrix of a cluster C_k — obtained by merging m_k time series — can be estimated as $\hat{p}_{kij} = (\alpha_{kij} + n_{kij})/(\alpha_{ki} + n_{ki})$.

We conclude this section by suggesting a choice of the hyper-parameters α_{kij}. We use uniform prior distributions for all the transition probability matrices considered at the beginning of the search process. The initial $m \times s \times s$ hyper-parameters α_{kij} are set equal to $\alpha/(ms^2)$ and, when two MCs are similar and the corresponding observed frequencies of transitions are merged, their hyper-parameters are summed up. Thus, the hyper-parameters of a cluster corresponding to the merging of m_k initial MCs will be $m_k\alpha/(ms^2)$. In this way, the specification of the prior hyper-parameters requires only the prior global precision α, which measures the confidence in the prior model. An analogous procedure can be applied to the hyper-parameters α_k associated with the prior estimates of p_k. We note that, since $\Gamma(x)$ is defined only for values greater than zero, the hyper-parameters α_{kij} must be non-negative. Note further that a choice of uniform hyper-parameters biases the algorithm toward the hypothesis that the initial m time series are generated from the same MC.

3. A Heuristic Search

To implement the clustering method described in the previous section, we might search all possible partitions and return the one with the highest posterior probability. Unfortunately, the number of possible partitions grows exponentially with the number of MCs so a heuristic method is required to make the search feasible. We use a measure of similarity between estimated transition probability matrices to guide the search process. The resulting algorithm is called *Bayesian Clustering by Dynamics* (BCD).

3.1 The Algorithm

The algorithm performs a bottom-up search by recursively merging the closest MCs (representing either a cluster or a single time series) and evaluating whether the resulting model is more probable than the model where these MCs are kept separate. The similarity measure that guides the process can be any distance between probability distributions. Let P_1 and P_2 be matrices of transition probabilities of two MCs. Because each is a collections of s probability distributions, and rows with the same index are probability distributions conditional on the same event, the measure of similarity that BCD uses is an average of the Kulback-Liebler distance between corresponding rows. Let p_{1ij} and p_{2ij} be the probabilities of the transition $i \rightarrow j$ in P_1 and P_2. The Kullback-Liebler distance of these two probability distributions is

$$D(p_{1i}, p_{2i}) = \sum_{j=1}^{s} p_{1ij} \log \frac{p_{1ij}}{p_{2ij}} \tag{4}$$

The average distance between P_1 and P_2 is then $D(P_1, P_2) = \sum_i D(p_{1i}, p_{2i})/s$. Note that this distance becomes 0 when $P_1 = P_2$ and it is otherwise greater than zero. The current implementation of the BCD algorithm is based on Equation 4 but other measures of distance, such as the *mean square error*, could be used instead. The rationale behind the distance measure is that merging more similar MCs earlier should find sooner more probable models and increase the marginal likelihood 3 used as scoring metric by the algorithm.

Algorithm 1 *The algorithm takes as input a set S of m time series and returns a partition B of times series S.*

procedure BCD(S)
  $MCs \leftarrow \emptyset$; $Distances \leftarrow \emptyset$;
  while $i \leq |S|$ **do** $MCs \leftarrow MCs \bigcup \{\text{MC}(S_i)\}$;
  while $i \leq |MCs|$ **do**
    while $j \leq |MCs|$ **do**
      if $i \neq j$ **then** $Distances \leftarrow Distances \bigcup \{\text{DISTANCE}(MCs_i, MCs_j)\}$;

```
    end
  end
  Distances←SORT(Distances);
  P_new←ML(MCs); Best←MCs;
  while P_new > P_old do
    P_old←P_new;
    while i ≤ |Distances| do
      MC←MERGE(Distances_i);
      Current←{MCs\{Distances_i}} ∪ {MC};
      when P_new > ML(Current) do
        Best←Current; P_new←ML(Current);
        while j ≤ |Distances| do
          if Distances_i ⋂ Distances_j ≠ ∅ then Distances←Distances\{Distances_j};
        end
        while j ≤ |Best| do
          Distances←Distances ⋃ DISTANCE(MC, Best_j);
        end
        Distances←SORT(Distances);
        return
      end
    end
  end
  return Best
end
```

Initially, the algorithm transforms each of m time series in a set S into a MC. This is accomplished by the function MC$(\cdot)$, which follows the procedure described in Section 2. Then, the algorithm sorts the set *Distances* of pairwise distances between the m generated MCs into an acscending order with the function DISTANCE$(\cdot,\cdot)$. This function returns a pair of MCs indexed by a measure of their mutual distance (which for the current work is defined in Equation 4). The last step of the initialization phase is the computation of the the marginal likelihood $p(S|M_s)$, where M_s represents the model in which each time series is generated by a different MC, and M_s is taken as current best model. The marginal likelihood 3 is computed by the function ML$(\cdot)$, taking as its argument a set of MCs.

The iterative core of the algorithm loops on the ordered set of MC pairs and tries to merge the two closest MCs into a single MC. The function MERGE$(\cdot)$ performs this merger as described in Section 2. This function also estimates the marginal likelihood $p(S|M_c)$, where M_c is the model in which the two merged MCs are replaced by the MC resulting from their merger. If the marginal likelihood of this model is higher than the marginal likelihood of the current *Best* model, the model M_c is taken as current *Best*, all the ordered

pairs involving one of the merged MCs are removed from the set *Distances*, the distances between the new MC and the other MCs in the *Best* model are sorted into the set *Distances*, and the procedure is iterated on the new *Best* model. Otherwise, the procedure tries to merge the remaining pairs of MCs in the ordered set *Distances*. If no merging will result in a model with higher marginal likelihood than the current *Best* model, the procedure stops and returns the *Best* model found so far.

3.2 Computational Complexity

We shall assume that the quantities required to compute Equation 3 in the function ML$(\cdot)$ and Equation 4 in the function DISTANCE$(\cdot,\cdot)$ have been pre-computed and stored in some array. We do not consider the time required by the function SORT$(\cdot)$.

Recall that s is the number of states of the variable generating the m time series in the set S. We assume that the longest time series contains n data points. The execution of function MC$(\cdot)$ requires at most ns^2 applications of Equations 1. Therefore, the first *while* statement requires $O(mns^2)$ time. The following nested *while* statements call the DISTANCE$(\cdot$,f)unction $(m\frac{m-1}{2})$ times, in case of symmetric distances, and $(m(m-1))$ times otherwise. As the function DISTANCE$(\cdot,\cdot)$ applies Equation 4 s^2 times, the cost of this second *while* statement is bounded by $O(s^2m^2)$. The third *while* statement is called, in the worse case, m times on the $(m(m-1))$ elements of *Distances*. As the function ML$(\cdot)$ sums over a table of ms^2 elements, the cost of the third *while* statement is bounded by $O(m^4s^2)$. When a new model is successfully created, we must also update the current set of *Distances* in order to sort the new generated MC into the other elements of the sorted list *Best*. As the number of new models is bounded by $(m-1)$, this updating requires, in the worse case, an additional $m^2 + s^2m^2(m-1)$ steps. Putting these results together, we obtain $O(nms^2) + O(m^2s^2) + O(m^4s^2) + O(m^2) + O(m^3s^2)$ and we can bound the cost of the algorithm by $O(m^4s^2)$.

4. An Experimental Evaluation

This section presents the results of an experiment to evaluate the accuracy of the algorithm when it is applied to a batch of time series generated from different MCs. The results show an overall accuracy that appears to suffer only when the time series in the batch are both very short and have very similar dynamics.

4.1 Procedure

To assess the accuracy of the algorithm, we varied four factors:

Factor 1 The length of each time series: 25, 50, 125 and 250 time steps.

Factor 2 The number of different MCs generating the batch of data: either 4 or 8.

Factor 3 The number of time series generated from each MC: either equal or different.

Factor 4 Global prior precision: $\alpha = 4, 8, 80$.

The first three factors yield sixteen experimental conditions. In each we generated eighty time series. Each time series included at most five distinct values; that is, each was generated by a MC with five states. We then tested the algorithm with these time series for three values of the global prior precision α.

The choice of values of Factor 1 follows from the fact that our time series are generate by MCs with five states. For a given number of states, we require a minimum sample size to ensure that states are visited often enough to yield good estimates of transition probabilities. A time series of $n+1$ steps generated from a MC on s states with uniform probabilities yields n/s^2 expected transitions $i \to j$, for any i and j. When the time series is not generated from a MC with uniform transition probabilities, some observed transition frequencies will be smaller than n/s^2, others will be larger. Hence, if n is not too large relative to s, we expect to have several null transition frequencies. We decided to fix the number of states of each MC to five and to generate, from each MC, time series of length 25="very short", 50="short", 125="medium" and 250="long". Very short and short time series give an expected number of one and two transitions in the uniform case, so that we expect to have several zeros when the generating MCs are not uniform. The expected number of transitions are five and ten in the other two cases.

The time series in an experimental batch are generated by either four or eight MCs. We constructed the MCs as follows: Each of five rows of a transition matrix is a probability distribution with masses on five states. When we think of probability distributions, we imagine symmetric, uniform or skew-symmetric distributions. Uniform distributions aside, the other patterns are characterized by changing the concentration of probability masses — without changing the shape of the distributions — to have different degrees of variability. We might construct an infinite number of transition matrices whose rows mix these different distributions. To avoid selection bias, we generated a set of sixteen probability distributions given by a uniform distribution and symmetric and skew-symmetric distributions that are shown in Figure 4. We then generated eight transition probability matrices $P_1, ..., P_8$ by sampling, in each case, five distributions from this set.

The Kullback-Liebler distance between pairs of transition probability matrices so generated ranges between 0.7 — in which the two transition probability matrices have three identical rows — and 1.95. In the latter case the two transition probability matrices represent very different dynamics, as in one case the chain is expected to visit all states 1–5 while, in the other case, transitions are mainly limited to states 1 and 4 with the other states having a very small probability of being visited. For the cases in which we generated the time series from four MCs, we sampled the four matrices P_1, P_4, P_5 and P_7.

We suspected that having generating processes equally represented in the batch of time series can help the algorithm to cluster MCs correctly. Hence, for each of the eight combi-

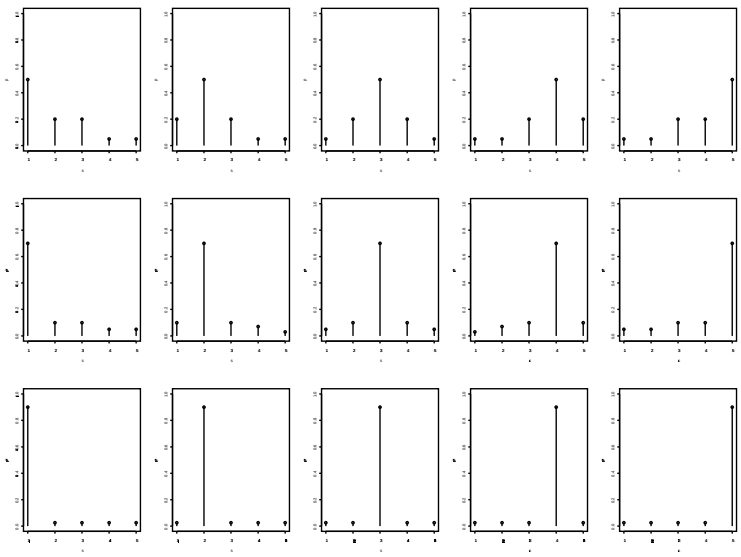


Figure 4: Probability distributions generating the transition matrices in the experiment.

nations of length and number of generating MCs, we created two batches of time series in which the number of time series generated from each MC is either constant or variable. In the constant conditions, each of four (or eight) MCs generated 20 (or 10) time series. In the unequal conditions, each of four MCs generates 18, 10, 33 and 19 time series, respectively; and each of eight MCs generated 12, 23, 2, 4, 5, 12, 16, 6, series. (These unequal distributions were generated randomly.) We also decided to start each simulated time series from the same initial point. However, since some initial point can be more advantageous for some time series than others, we choose the common initial point at random. Thus we generated eighty time series for each of the sixteen combinations of levels of the factors. We first generated four sets of eighty time series of length 250 time steps, and then we extracted four sets given by the first 25 steps, four sets given by the first 50 time steps, and four sets given by the first 125 time steps. In this way, the comparisons of the results for time series of varying lengths are not confounded with the fact that we observe different processes, although generated from the same MC.

We then applied the BCD algorithm to the sixteen data sets for three different values of the global prior precision. Values of $\alpha = 4$ or 8 give a small adjustment to the observed frequency, as the initial values of the prior hyper-parameters are $\alpha_{kij} = 0.002$ and $\alpha_{kij} = 0.004$ and they become at most $\alpha_{kij} = 0.16$ in the worst case in which the algorithm merges

	No of generated time series per MC							
	Equal				Different			
Length	25	50	125	250	25	50	125	250
4 MCs	4	4	4	4	4	4	4	4
8 MCs	8	8	8	8	4	7	7	8

Table 2: Number of clusters created in the sixteen data sets.

the eighty time series into one cluster. We note that, since several transition frequencies are zeros, small values of α_{kij} are supposed to affect the precision of the posterior probability. A choice of $\alpha = 80$ corresponds to setting $\alpha_{kij} = 0.20$ initially and, when the algorithm begins to assign time series to clusters, these values increase. For instance, when 10 time series are assigned to the same cluster, the hyper-parameter values rise to 2.

Recall that the task of the clustering algorithm is two-fold: Find the set of clusters that gives the best partition of time series, and assign the time series to clusters. We evaluate the algorithm using two performance measures: the number of clusters found in each data set and the number of time series correctly assigned to clusters. For each cluster C_j found by the algorithm, we find the generating MC with transition matrix P_i that has minimum Kullback-Liebler distance from the transition probability matrix of the cluster. We then compute the number of time series generated from the MC with transition matrix P_i that are assigned to the cluster C_j. The cumulative sum of time series assigned, correctly, to all clusters is the value of the second performance measure. The first performance measure ranges between 1, when the algorithm creates only one cluster, and 80, if no time series are clustered. The correct value is 4 when the time series are generated from 4 MCs and 8 in the other case. The second performance measure ranges between 0, when the algorithm does not assign correctly any time series, to 80, when all time series are assigned correctly to their generating MCs.

4.2 Results and Discussion

The number of clusters created by the BCD algorithm in each of the sixteen experimental conditions is shown in Table 2 (global prior precision is $\alpha = 8$. The results were not sensitive to changes of the global prior precision). Table 3 represents the accuracy with which the algorithm assigns time series to clusters. For conditions in which the batch of 80 time series was generated by four MCs, exactly four clusters were always found, and in all but one condition, all time series were correctly assigned to their generating MCs. In one condition, when the time series are only 25 steps long, the algorithm wrongly assigns a single time series generated from matrix P_7 to a cluster of series generated from matrix P_5. (P_7 and P_5 are the two most similar matrices.) Thus, in the conditions in which we used

	No of generated time series per MC							
	Equal				Different			
Length	25	50	125	250	25	50	125	250
4 MCs	79	80	80	80	80	80	80	80
8 MCs	77	80	80	80	63	76	77	80

Table 3: Number of time series correctly assigned to clusters in the sixteen data sets.

four generating MCs neither the lengths of the time series nor the fact that the generating MCs are represented unevenly in the data set has any effect.

In those conditions in which eight MCs are used to generate the 80 time series, performance is understandably a little less accurate, because fewer time series are available to represent each MC. Consider first the four conditions in which each MC generates exactly ten time series. Table 2 shows that, in these cases, the algorithm generated the correct number of clusters, one for each generating MC. Table 3 shows that in three of these four conditions, the assignment of time series to clusters was perfect. Only when the time series were very short was there a problem: One cluster in this condition contains just eight time series, albeit generated from the same MC; two clusters contain ten time series generated from the same MC and one series wrongly assigned. One cluster contains nine time series generated from the same MC and one wrongly assigned. Globally, three time series are mis-assigned so that the second performance measure is 77.

The results in the last group of conditions, in which we generated an uneven number of time series from eight MCs is indicative of a possible weaknesses of the BCD algorithm. This is the only situation in which the algorithm fails to create the correct number of clusters until the time series are 250 steps long. The assignment of time series to clusters also fails to be optimal until the time series are long enough. The data set contains eighty time series generated in numbers 12, 23, 2, 4, 5, 12, 16, 6 from eight MC transition matrices $P_1, ..., P_8$. The algorithm creates 4 clusters when the time series are 25 steps long, 7 clusters when the time series are either 50 or 125 steps long and identifies the correct number of clusters when the time series are 250 steps long.

Figure 4 shows the details of the clusters and of the assignment of time series to clusters. When the sequences are very short, the algorithm merges time series generated from matrices P_3, P_7 and P_8 in one cluster. Time series generated from P_1 and P_5 are also assigned to one cluster, as are P_2 and P_4. Only the series generated from P_6 are correctly identified. Although the amount of information about the generating process is not sufficient to let the algorithm identify the correct generating MCs, the clusters generated in these conditions have meaningful features. For instance, the matrices P_3 and P_8 have the smallest Kullback-Liebler distance of any pair of matrices (0.70), while the matrices P_7 and P_8 have the second smallest distance (0.76). The matrix P_7 is frequently represented in the data set, sixteen

		Time Series Length			
MC	Time Series	25	50	125	250
P_6	47–58	•	•	•	•
P_1	1–12	}	•+ 46	•	•
P_5	42–46	}	• – 46	•	•
P_2	13–35	}	•	•	•
P_4	38–41	}	•	•	•
P_3	36–37	}	} + 67	} + 67	•
P_8	75–80	}	} + 67	} + 67	•
P_7	59–70	}	• – 67	• – 67	•
Total Time Series:		80	80	80	80
Classification Errors:		17	4	3	0

Table 4: Cluster identification and assignments in the experiments. A dot represents a correct cluster identification. Curly brackets put together time series generated from different MCs and assigned to one single cluster by the algorithm. A $+n$ denotes that the time series n is assigned to the wrong cluster, a $-n$ denotes that the times series n is not included in the right cluster.

time series generated from it, whereas just two and six time series were generated from P_3 and P_8, respectively. Apparently, the cluster containing the sixteen series generated from P_7 absorbs the series generated from P_3 and P_8. Similarly, the cluster containing the twelve time series generated from P_1 absorbs the five time series generated from P_5, just as the cluster containing the twenty-three time series generated from P_2 absorbs the four series generated from P_4. We note that the Kullback-Liebler distance $D(P_1, P_4)$ is greater than $D(P_1, P_5)$, so that the assignment of time series to clusters reflects features of the generating processes. The only cluster that is correctly identified contains twelve time series generated from P_6. This matrix is maximally different from the seven other matrices.

As the time series become longer, the algorithm accuracy increases. With time series of 50 steps, the algorithm is able to distinguish series generated from P_2 and P_4, as well as P_1 and P_5 — although series 46 is still assigned to the wrong cluster. Similarly, the time series generated from P_3, P_7 and P_8 are assigned to two clusters. One of these contains all but one series generated from P_7, and this series is assigned to the cluster merging the series generated from P_3 and P_8. This merger is the biggest error committed by the algorithm when the time series are 50 and 125 steps long. For series of length 250, the accuracy of the algorithm is 100%.

Clearly the algorithm can be extremely accurate when the batch of data contain roughly equal numbers of time series generated by different MCs. The fact that series are short does

not seem to jeopardize accuracy when only four generating MCs are involved. However, when time series are short and their generating MCs are quite similar, and some of these MCs contribute many more series to a batch of data than the others, then one cluster may be formed from time series from the heavily-represented MC as well as a few series from similar MCs. Our algorithm may not distinguish similar MCs when one contributes many more time series to a batch of data than another, very similar one, and the series are short. This potential problem may be exacerbated by the fact that uniform priors bias the clustering algorithm toward the hypothesis that the time series in a data set are generated from a common MC. If the sample does not provide enough information to change the prior knowledge about a model, then the model may not discriminate MCs. However, this danger is typical of statistical methods applied to small samples.

5. Case Studies

In this section, we describe two applications of Bayesian clustering by dynamics. The first is a data reduction problem in which time series from a simulator of military engagements are clustered. The second involves time series generated by the Pioneer 1 mobile robot, and an effort to have the robot learn to cluster and then classify its experiences.

5.1 Clusters of Military Engagements

In this section we report the result of our algorithm applied to the identification of dynamics in a military scenario.

5.1.1 The Problem

The domain of our application is a simulated military scenario. For this work, we employ the *Abstract Force Simulator* (AFS) [1], which has been under development at the University of Massachusetts for several years. AFS uses a set of abstract agents called *blobs* which are described by a small set of physical features, including mass and velocity. A blob is an abstract unit; it could be an army, a soldier, a planet, or a political entity. Every blob has a small set of primitive actions that it can perform, primarily *move* and *apply-force*, to which more advanced actions, such as tactics in the military domain, can be added. AFS operates by iterating over all the units in a simulation at each clock tick and updates their properties and locations based on the forces acting on them. The physics of the world specifies probabilistically the outcomes of unit interactions. By changing the physics of the simulator, a military domain was created for this work.

The time series that we want to analyze come from a simple 2-on-2 *Capture the Flag* scenario. In this scenario, the blue team, *Blue 1* and *Blue 2*, attempt to capture the objective *Red Flag*. Defending the objective is the red team, *Red 1* and *Red 2*. The red team must defend the objective for 125 time steps. If the objective has not been captured by the 125^{th}

	Blob	*Task*
Primary Effort	Red 2	**retain** objective **Red Flag**
	Blue 2	**attack** objective **Red Flag**
Supporting Effort	Red 1	**attack** blob Blue 1
	Blue 1	**escort** blob Blue 1

Table 5: The tasks given to each blob in the scenario.

time step, the trial is ended and the red team is awarded a victory. The choice of goals and the number of blobs on each team provide a simple scenario. Each blob is given a task (or tactic) to follow and it will attempt to fulfill the task until it is destroyed or the simulation ends (Table 5). In this domain, *retaining* requires the blob to maintain a position near the object of the retain — the Red Flag in this example — and protect it from the enemy team. When an enemy blob comes within a certain proximity of the object of the retain, the retaining blob will attack it. *Escorting* requires the blob to maintain a position close to the escorted blob and to attack any enemy blob that comes within a certain proximity of the escorted blob. *Attacking* requires the blob to engage the object of the attack without regard to its own state. These tactics remain constant over all trials, but vary in the way they are carried out based on environmental conditions such as mass, velocity and distance of friendly and enemy units. To add further variety to the trials, there are three initial mass values that a blob can be given. With four blobs, there are 81 combinations of these three mass values. At the end of each trial, one of three ending conditions is true:

A The trial ends in less than 125 time steps and the blue team captures the flag.

B The trials ends in less than 125 time steps and the blue team is destroyed.

C The trial is stopped at the 125^{th} time step and the blue team fails to achieve its goal.

To capture the dynamics of the trials, we chose to define our state space in terms of the number of units engaged and force ratios. There are three possible engagement states at each time step. Red has more blobs "free" or unengaged (*F1*), both blue and red have an equal number of unengaged blobs (*F2*), or blue has more unengaged blobs (*F3*). In each of these states, either the red team or the blue team has more unengaged mass (*FFR+* or *FFR-* respectively). In each of the six possible combinations of the above states, either red or blue has more cumulative mass (*CFR+* or *CFR-* respectively). Altogether there are 12

State #	*State Description*	*Notes*
0	(*F1, FFR+, CFR+*)	Strong Red
1	(*F1, FFR+, CFR-*)	
2	(*F1, FFR-, CFR+*)	
3	(*F1, FFR-, CFR-*)	
4	(*F2, FFR+, CFR+*)	Strong Red
5	(*F2, FFR+, CFR-*)	
6	(*F2, FFR-, CFR+*)	
7	(*F2, FFR-, CFR-*)	Strong Blue
8	(*F3, FFR+, CFR+*)	
9	(*F3, FFR+, CFR-*)	
10	(*F3, FFR-, CFR+*)	
11	(*F3, FFR-, CFR-*)	Strong Blue

Table 6: The state space for Capture the Flag military engagements.

possible world states, as shown in Table 6. The table shows states 0 and 4 to be especially advantageous for red and states 7 and 11 to be favorable to blue.

5.1.2 Clusters of Dynamics

The 81 time series generated with AFS for the Capture the Flag scenario consist of 42 trials in which the blue team captures the red flag (end state A), 17 trials in which the blue forces are defeated (end state B) and 22 which were stopped after 125 time steps (end state C).

We used our BCD algorithm to partition the time series. With a prior global precision $\alpha = 972$ — corresponding to the initial assignment $\alpha_{kij} = 1/12$ in the 81 transition probability matrices — the algorithm produced eight clusters. Table 7 gives the assignment of time series to each of these clusters. By analyzing the dynamics represented by each cluster, it is possible to reconstruct the course of events for each trial. We did this "by hand" to understand and evaluate the clusters, to see whether the algorithm divides the trials in a meaningful way. We found that, indeed, the clusters correspond not only to end states A, B and C, but to different prototypical ways in which the end states were reached.

Clusters C_2, C_4 and C_5 consist entirely of trials in which blue captured the flag or time expired (end state A and C). While this may at first be seen as the algorithm's inability to distinguish between the two events, a large majority (though it is not possible to judge how many) of the "time-outs" were caused by the blue team's inability to capitalize on a favorable circumstance. A good example is a situation in which the red team is eliminated, but the blue blobs overlap one another in their attempt to reach the flag. This causes them to slow to a speed at which they were unable to move to the flag before time expires. Only

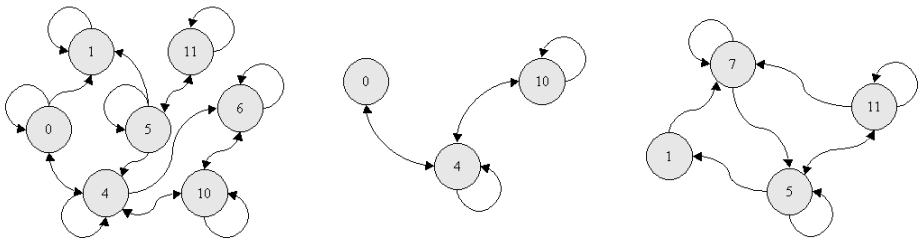
Cluster	*A*	*B*	*C*	*Total*
C_1	5	1	3	9
C_2	2	0	2	4
C_3	7	0	0	7
C_4	14	0	12	26
C_5	1	0	1	2
C_6	8	16	4	28
C_7	2	0	0	2
C_8	3	0	0	3
Total	42	17	22	81

Table 7: Summary of the clusters identified by the BCD algorithm.

a handful of "time-outs" represent an encounter in which the red team held the blue team away from the flag. Clusters C_2, C_4 and C_5 demonstrate that the clustering algorithm can identify subtleties in the dynamics of trials, as no information about the end state is provided, implicitly or explicitly, by the world state.

Clusters C_1 and C_6 merge trials of all types. C_1 is an interesting cluster of drawn out encounters in which the advantage changes sides, and blobs engage and disengage much more than in the other clusters. For example, C_1 is the only cluster in which the MC visits all states of Table 6, in particular, is the only cluster in which state 8 is visited. By looking at the transition probabilities, we see that state 8 is more likely to be reached from state 6, and to be followed by state 0. Thus, from a condition of equal free units (*F2*) we move to a situation in which blue disengages a unit and has a free unit advantage (*F3*), which is immediately followed by a situation in which red has a free units advantage (*F1*). The "time-outs" (end state C) in this cluster represent the red team holding off the blue team until time runs out. Cluster C_6, on the other hand, contains all but one of the trials in which the red team eliminated all of the blue units (end state B), as well as very similar trials where the red blobs appear dominant, but the blue team makes a quick move and grabs the flag. The cluster is characterized by having transitions among states 0, 4 and 10, with a large probability of staying in state 0 (in which the red forces are dominant) when reached. The large number of trials in which the blue team wins (especially large when we realize that C-endings are blue wins but for the fact that overlapping forces move very slowly) is a result of Blue 1 being tasked to escort Blue 2, a tactic which allows Blue 1 to adapt its actions to a changing environment more readily than other unit's tactics, and in many trials, gives blue a tactical advantage.

Clusters C_3, C_7 and C_8 merge only time series of end state A, in which the blue team always captured the flag. Figure 5 displays the MC representing the three clusters (in which we have removed transitions with very low probability). Each cluster captures a different



CLUSTER 3 CLUSTER 7 CLUSTER 8

Figure 5: Markov Chains representing clusters C_3, C_7 and C_8.

dynamics of how a blue victory was reached. For example, cluster C_8 is characterized by transitions among states 1, 5, 7 and 11 in which the blue team maintains dominance, and transitions to states 4 and 8 — in which the red forces are dominant — have a very low probability. The trials in cluster C_7 visited states 0, 4, and 10 frequently and correspond to cases in which the blue team won despite a large mass deficit. In these cases, the objective was achieved by a break away of one of the blue blobs that outruns the red blobs to capture the flag. Cluster C_3 displays transitions among states 0, 1, 4, 5, 6, 10 and 11 and represents longer, more balanced encounters in which the blue team was able to succeed.

5.1.3 A Comparison with String-Edit Distance

In this section, we compare the clusters obtained with our Bayesian clustering method with those obtained using String-Edit Distance. String-Edit Distance is a measure of similarity between two strings that share a common discrete alphabet. We use dynamic programming to determine the string-edit distance of all pairs of time series generated by the AFS simulation. The algorithm finds the minimum number of deletions and insertions of elements from the alphabet necessary to transform one string into another. Since the String-Edit Distance is a function of the length of each string, the number of deletions and insertions is normalized by dividing by the average string length of the two strings being compared. The string-edit distance of a pair of time series serves as a measure of their similarity for the purposes of agglomerative clustering. At the outset of clustering, each time series is in a cluster by itself. During successive steps of the clustering algorithm, the two clusters with the minimum average string-edit distance between their members are joined. The algorithm halts when joining two clusters would increase the intra-cluster distance by more than the average distance from the original cluster to the new cluster. String-Edit Distance was chosen as a good similarity metric because trials with similar dynamics should, in theory, have similar string — sequence of states — compositions. Trials with similar transitions between states (corresponding to carrying out certain parts of tactics) should have smaller

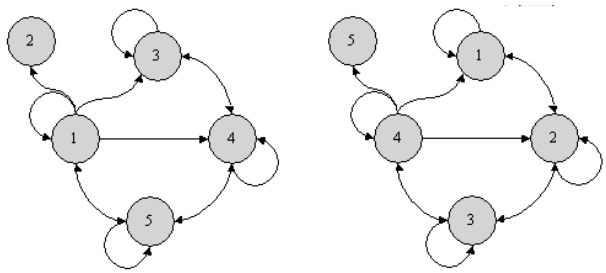
Cluster	*A*	*B*	*C*	*Total*
C'_1	18	11	2	31
C'_2	1	1	2	4
C'_3	15	0	8	23
C'_4	5	5	4	14
C'_5	2	0	6	8
C'_6	1	0	0	1
Total	42	17	22	81

Table 8: Summary of the clusters identified by String-Edit distance.

string-edit distances than trials that have different transitions. By normalizing the string length, the clusters are not explicitly affected by the length of the trials.

Table 8 shows the results. The String-Edit Distance produces 6 clusters, and the assignment of time series of end state A, B and C to clusters shows several similarities with the clusters produced by our algorithm. Cluster C'_2 is a strict subset of cluster C_1, while cluster C_8 is a strict subset of C'_3 and cluster C_5 is a strict subset of string-edit cluster C'_1. String-edit cluster C'_4 is very closely related to cluster C_6, showing that string-edit distance also had problems clustering by the very complex dynamics exhibited by time series in this cluster. Note, however, that while cluster C_6 is dominated by series of end state B, cluster $C'4$ contains a nearly even number of series of end states A, B and C.

For a more formal comparison of the clusters produced by our algorithm and String-Edit Distance, we measured the amount of information that the two clustering algorithms lose with respect to the composition of the original 81 time series into end states A, B and C. Each cluster — either C_k or C'_k — carries an amount of information about time series of end states A, B, C in terms of their proportions. When a cluster contains time series with one end state only, it is most informative. It is least informative when the proportions of time series with end states A, B and C are equal (in this case the cluster does not contain any information to discriminate end states). These proportions, say p_{Ak}, p_{Bk} and p_{Ck}, can be regarded as a probability distribution over the end states in clusters C_k and C'_k. The loss of information of each cluster is then given by the entropy $H_k = -p_{Ak} \log p_{Ak} + p_{Bk} \log p_{Bk} + p_{Bk} \log p_{Bk}$ and the mean entropy $H = \sum_k H_k/c$ — c is the number of clusters — measures the average loss of information per cluster. We find that the mean entropy of our clustering method is 0.5, versus 0.7 of String-Edit Distance. Since the maximum loss of information is achieved when the proportions $p_{Ak} = p_{Bk} = p_{Ck}$ are equal to $1/3$ and $H_k = \log(3)$, the relative difference of entropy between the two clustering methods is $(0.5 - 0.7)/\log(3) = -0.19$. Hence, the String-Edit Distance loses 19% information more than the BCD algorithm.



CLUSTER 1 CLUSTER 2

Figure 6: MC representing the first and the second cluster.

5.2 Clusters of Sensory Inputs

One goal of our robotics work is to enable mobile robots to learn classes of activities without supervision. For example, our robot has learned clusters of time series of sensor data that correspond to passing an object and moving toward an object. We present briefly an example of Bayesian clustering by dynamics of robot sensor data. The main point of this section, however, is to introduce another evaluation method for Markov chain models and clusters, which penalizes these abstractions for losing information about time series.

5.2.1 The Robot Platform

The Pioneer 1 robot is a small platform with two drive wheels and a trailing caster, and a two degree of freedom paddle gripper. Our configuration of the Pioneer 1 has roughly forty sensors including sonars, gripper sensors, and a primitive vision system. The values returned by some sensors are derived from others.

During its interaction with the world, the robot records the values of about 40 sensors every 1/10 of a second. In an extended period of wandering around the laboratory, the robot will engage in several different activities — moving toward an object, losing sight of an object, bumping into something — and these activities will have different sensory signatures to be clustered.

5.2.2 Prototypical Dynamics for the Robot

In an experimental trial lasting about 30 minutes, the robot's activities were divided into 42 episodes, without human intervention, by the following criterion: An episode ends when three or more sensors' values change simultaneously. The data include 11,118 values for each sensor. Our prior hyper-parameters are computed by uniformly distributing the global prior precision, where $\alpha = 5$ and $\alpha' = 42$.

Figure 6 depicts the MCs representing the two clusters C_1 and C_2 learned for the sensor

`vis-a-x`, the horizontal location of an object in the visual field. The first cluster captures the sensor dynamics when the robot is not close to an object. The transitions are limited to states 3, 4 and 5 that correspond to the range -28, 140. The initial state 1 can be reached from state 5, which represents the fact that the object appears and disappears from the visual field. However, since the estimate of the probability of transitions $5 \rightarrow 1$ and $1 \rightarrow 5$ are derived from only two cases observed in all the episodes merged into cluster C_1, the confidence in these estimate is very low. The second cluster, on the other hand, represents the sensor dynamics for an object not far from the robot, since transitions are essentially limited among the first 4 states. The prior specification does not rule out the possibility that either state 1 or 5 be reached from state 4. However, in the 12 episodes merged to create cluster C_2, the transitions $4 \rightarrow 5$ and $4 \rightarrow 1$ were never observed, while state 4 was reached only once, from state 3.

5.2.3 Evaluating Information Loss in Bayesian Clustering by Dynamics

Clustering by dynamics involves two levels of abstraction. First, the transitions in an episode are summarized in a transition matrix, or MC; second, episodes are grouped into clusters, and the MCs for clusters are averages of the constituent episode MCs. Both operations lose information. The *log-score* of a transition helps us evaluate these losses [5]. Let $s_{e,kij} = -\log \hat{p}_{kij}$ be the score of the transition $i \rightarrow j$ *observed* in an episode e. This score penalizes an episode MC by assigning large values to observed transitions when the MC assigns them small probabilities. We sum this score over all transitions in an episode, and sum again over all episodes, to get a score for the loss incurred by summarizing the time series of episode transitions into episode MCs. Now, instead of computing losses for episode e based on the episode MC, we can compute them based on the MC for the *cluster* to which e belongs. We let $s_{c,kij} = -\log \hat{p}_{kij}$ be the score assigned to the transition $i \rightarrow j$ observed in an episode that belongs to cluster C_k. As before, we sum $s_{c,kij}$ over the transitions within an episode, and sum again over episodes.

Finally, if episode e belongs to cluster C_k, we can ask how much predictive accuracy would be lost by using a *randomly selected* cluster to predict the episode transitions. This amounts to a test of whether clusters retain any predictive power at all: if not, then a randomly selected cluster will incur the same losses as cluster C_k for episode e. The score $s_{r,kij} = -\log \hat{p}_{kij}$ is given to transition $i \rightarrow j$ *observed* in episode k but predicted by a random cluster. Once again, these scores are summed over transitions in an episode and over episodes. We have now described three cumulative scores, s_e, s_c and s_r, which, for the data described earlier, have values $s_e = 120.5$, $s_c = 212.2$ and $s_r = 449.5$. The loss is least for episode MCs, intermediate for cluster MCs and highest for randomly-selected cluster MCs. Apparently, the MC for episode e does a better job of predicting transitions in e than does the MC for cluster C_k to which e belongs, and both are better than using a randomly selected cluster MC to make predictions.

Sign tests can tell us whether $s_e = 120.5$, $s_c = 212.2$ and $s_r = 449.5$ are significantly different. Let $s_{\delta,kij} = s_{e,kij} - s_{c,kij}$ be the difference in scores assigned to episode and cluster MCs for the transition $i \rightarrow j$ in episode k. Under the null hypothesis that the episode and cluster MCs make equally lossy predictions, half these differences should have a positive sign, half negative. In fact, 1567 differences are positive, 9950 are negative. We can compare cluster predictions with randomly-selected cluster predictions in the same way; 2799 differences are positive, 4445 are negative, and 3993 are zero. The sampling distribution for the number of positive differences is binomial and is well approximated by the normal distribution for this sample size. We find that episode MCs are significantly less lossy than cluster MCs, which are themselves significantly less lossy than randomly-selected cluster MCs.

6. Conclusions

The world is dynamic and learning about the world often means learning about dynamics. Our overriding goal has been to develop algorithms sufficient for a robot to learn about its activities by clustering time series of activities by their dynamics. This work was described briefly in Section 5 and in more depth in [15]. Related methods for clustering robot data by their dynamics are presented in [8]. This paper focused on the description and analysis of Bayesian clustering by dynamics BCD. The method starts by modeling dynamics as MCs then applies a Bayesian clustering procedure to merge these MCs. On artificial data, the algorithm achieves very high accuracy in most conditions. We also applied the algorithm to the problem of summarizing simulation data. It's one thing to run dozens of simulations of military engagements, quite another to offer a concise critique of a military plan. How can we extract critiques from megabytes of time series data generated by the simulations? One can see immediately that this problem is not limited to simulations of military engagements. All simulators produce vast amounts of data in need of summary. Many natural processes and engineered artifacts similarly generate masses of time series data. BCD reduces data to a few prototypical time series; for example, it reduces one hundred military engagements to eight clusters that correspond to different ways to win and lose battles. Explaining half a dozen clusters is much easier than explaining the original time series. So whether one's task is to cluster robot experiences into an ontology of activities, or to reduce data preparatory to explanation, BCD has a role.

The principal limitation of the BCD algorithm is that it is limited to univariate time series, whereas the dynamics of many phenomena are played out in several variables; for example, the Pioneer 1 robot has roughly 40 sensors, many of which contribute information about activities. We are extending BCD to the multivariate case. A straightforward way to do it is to form a state space from the power set of state variables. If i variables each have j states then the number of "superstates" is j^i, and the transition matrices contain $(j^i)^2$ transitions. The data requirement for good estimates of probabilities in such matrices

can be prodigious. Instead we are exploring techniques that first find the combinations of state variable values that actually occur in time series, then clustering on the dynamics of transitions within this smaller set of superstates.

Acknowledgments

We wish to thank Tim Oates for help with string-edit distance and for fruitful discussions about approaches to clustering by dynamics. This research is supported by DARPA/AFOSR under contract(s) No(s) F49620-97-1-0485. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of DARPA/AFOSR or the U.S. Government. This research was developed while Paola Sebastiani and Marco Ramoni were visiting fellows at the Department of Computer Science, University of Massachusetts, Amherst.

References

[1] M. Atkin, D. L. Westbrook, P. R. Cohen, and G D. Jorstad. AFS and HAC: The last agent development toolkit you'll ever need. In *Proceedings of the AAAI Workshop on Software Tools for Developing Agents*. American Association for Artificial Intelligence, 1998.

[2] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180, Cambridge, MA, 1996. MIT Press.

[3] P.R. Cohen and D. Jensen. Overfitting explained. In *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics*, pages 115–122, 1997.

[4] G.F. Cooper and E. Herskovitz. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[5] D.J. Hand. *Construction and Assessment of Classification Rules*. Wiley, New York, 1997.

[6] A.E. Howe. Analyzing failure recovery to improve planner design. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 387–392. AAAI Press/The MIT Press, 1992. Also available as University of Massachusetts Computer Science Department Technical Report 92-42.

[7] A.E. Howe and G. Somlo. Modeling discrete event sequences as state transition diagrams. In *Advances in Intelligent Data Analysis: Reasoning About Data. Proceedings of IDA-97.* SpringerVerlag, 1997.

[8] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. In *Proceedings of the Sixteenth International Conference on Machine Learning.* Morgan Kaufmann Publishers, Inc., 1999. Submitted.

[9] T. Oates and P.R. Cohen. Searching for structure in multiple streams of data. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 346–354. Morgan Kaufmann Publishers, Inc., 1996.

[10] T. Oates, M.D. Schmill, D. Jensen, and P.R. Cohen. A family of algorithms for finding temporal structure in data. In *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics*, pages 371–378, 1997.

[11] L. R. Rabiner. A tutorial on Hidden Markov models and selected applications in speach recognition. *Proceedings of the IEEE*, 77:257–285, 1989.

[12] M. Ramoni and P. Sebastiani. Bayesian methods for intelligent data analysis. In M. Berthold and D.J. Hand, editors, *An Introduction to Intelligent Data Analysis*, New York, 1999. Springer. Also available as KMi Technical Report 67, Knowledge Media Institute, The Open University, United Kingdom at `http://kmi.open.ac.uk/techreports/KMI-TR-67`.

[13] M. Rosenstein and P. R. Cohen. Concepts from time series. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* AAAI Press, 1998.

[14] S.M. Ross. *Stochastic Processes.* Wiley, New York, 1996.

[15] P. Sebastiani, M. Ramoni, and P. Cohen. Bayesian clustering of sensory inputs by dynamics. Technical report, Experimental Knowledge Systems Laboratory, University of Massachusetts, 1999.

[16] P. Smyth. Clustering sequences with hidden Markov models. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing*, volume 9, Cambridge, MA, 1997. MIT Press.

[17] R. St. Amant and P. R. Cohen. Evaluation of a semi-autonomous assistant for exploratory data analysis. In *Proceedings of the First International Conference on Autonomous Agents*, pages 355–362, 1997.