AutoPlait: Automatic Mining of Co-evolving Time Sequences

Yasuko Matsubara
Kumamoto University
yasuko@cs.kumamoto-u.ac.jp

Yasushi Sakurai
Kumamoto University
yasushi@cs.kumamoto-u.ac.jp

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

ABSTRACT

Given a large collection of co-evolving multiple time-series, which contains an unknown number of patterns of different durations, how can we efficiently and effectively find typical patterns and the points of variation? How can we statistically summarize all the sequences, and achieve a meaningful segmentation?

In this paper we present AUTOPLAIT, a fully automatic mining algorithm for co-evolving time sequences. Our method has the following properties: (a) effectiveness: it operates on large collections of time-series, and finds similar segment groups that agree with human intuition; (b) scalability: it is linear with the input size, and thus scales up very well; and (c) AUTOPLAIT is parameter-free, and requires no user intervention, no prior training, and no parameter tuning.

Extensive experiments on *67GB* of real datasets demonstrate that AUTOPLAIT does indeed detect meaningful patterns correctly, and it outperforms state-of-the-art competitors as regards accuracy and speed: AUTOPLAIT achieves near-perfect, over *95%* precision and recall, and it is up to *472 times* faster than its competitors.

Categories and Subject Descriptors: H.2.8 [**Database management**]: Database applications–*Data mining*

General Terms: Algorithms, Experimentation, Theory

Keywords: Time-series data, Automatic mining

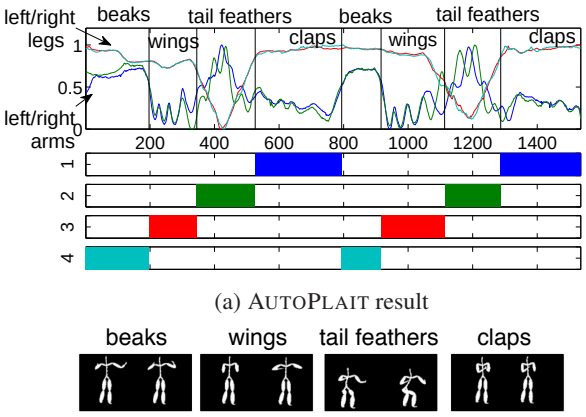
1. INTRODUCTION

Given a large collection of co-evolving time series, such as motion capture and web-click logs, how can we find the typical patterns or anomalies, and statistically summarize all the sequences? In this paper we focus on a challenging problem, namely fully-automatic mining, and more specifically, we tackle the four important time-series analysis tasks, namely, *CAPS* (**C**ompression / **A**nomaly-detection / **P**attern-discovery / **S**egmentation).

Our goal is to analyze a large collection of multiple time-series, (hereafter, a *"bundle"* of time-series), and summarize all the sequences into a compact yet powerful representation. So, what is a good representation of time-series? In practice, real-life data has distinct multiple trends, such as the weekday/weekend patterns of web-click sequences, and the normal/abnormal patterns seen in network traffic monitoring (hereafter we refer to such a pattern as a *"regime"*). How can we describe all these trends and distinct patterns (i.e., regimes) in large datasets? We need a good summarization of time-series in terms of a statistical framework.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2588556.



(a) AUTOPLAIT result

(b) Four basic steps of the "chicken dance"

Figure 1: AUTOPLAIT "automatically" identifies the dance steps of a motion capture clip, as well as the positions of the all cut points: original motion capture sequences and our result for the "chicken dance", which consists of four basic steps, "beaks", "wings", "tail feathers" and "claps".

In this paper, we present AUTOPLAIT, [1] which answers the above questions, and provides a good description of large collections of co-evolving sequences. Intuitively, the problem we wish to solve is as follows:

INFORMAL PROBLEM 1. *Given a large collection of co-evolving time series $\boldsymbol{X}$ (i.e., bundle), find a compact description of $\boldsymbol{X}$, i.e.,*

- *find a set of segments and their cut points,*
- *find a set of groups (namely, regimes) of similar segments*
- *automatically, and quickly*

Our algorithm automatically identifies all distinct patterns/regimes in a time-series, and spots the time-position of each variation.

Preview of our results. Figure 1 (a) shows the original time-series of the "chicken dance"[2] and the result we obtained with AUTOPLAIT. The motion consists of four co-evolving sequences: left/right arms, and left/right legs, composed of four basic steps in the following order: "beaks", "wings", "tail feathers" and "claps" (see

[1] Available at http://www.cs.kumamoto-u.ac.jp/~yasuko/software.html
[2] Popular in the 1980's; please see, e.g., http://www.youtube.com/watch?v=6UV3kRV46Zs&t=49s

Table 1: Capabilities of approaches. Only our approach meets all specifications.

	DWT	pHMM/ DynaMMo	HHMM	AUTOPLAIT
Compression	√	√	√	√
Segmentation		√	√	√
Regime identification			√	√
Anomaly detection				√
Parameter free				√

Figure 1 (b)). Figure 1 (a) also shows the regime labels (1,2,3,4) assigned to the frames along the vertical axis of the figure. The assignment of labels made it possible to successfully identify the four basic steps (i.e., "regimes") of the chicken dance, (i.e., the beak, wing, tail-feather, and clap steps are assigned to #4, #3, #2, and #1, respectively). Most importantly, these steps (i.e., "regimes") are unknown in advance, and thus the algorithm should recognize the groups, each of whose motions has similar characteristics. Also notice that our method automatically determines the discontinuity/transition points, by dividing the time interval into 8 segments (two for each regime).

1.1 Importance of fully-automatic mining

There are many, fascinating research problems for time-series analysis including pattern discovery [22, 25], summarization [24], clustering [19], segmentation [14, 18, 34] and sequence matching [28, 33, 26]. Although important, these methods basically require parameter settings and fine tuning. For example, the state-of-the-art methods [18, 34] require several user-specified inputs, such as the number of segments, and the reconstruction error thresholds (details are provided in section 6), and they are very sensitive to these parameters. The ideal method should look for arbitrary patterns and require no initial human intervention to guide it.

There is another important question: what if we have to handle very large datasets with millions, billions (or even trillions [26]) of time-series? In fact, faced with "big data", fully automatic mining is even more important: otherwise, the user would have to try several parameter tuning steps, each of which would take too long (e.g., hours, or days). Namely, as regards real big data analysis, we *cannot afford* human intervention.

1.2 Contrast with competitors

Table 1 compares AUTOPLAIT with existing methods. We illustrate their strengths and shortcomings: a check mark (√) indicates that a given method fulfills the corresponding requirement. Only our approach has checks against all entries, while,

- Wavelets and Fourier transforms (i.e., DWT, DFT) can compress the data into a fixed amount of memory, and detect bursts and typical patterns by keeping track of the top c largest wavelet/Fourier coefficients, but they cannot detect any segments or regimes.
- SWAB [14] and pHMM [34] have the ability to find linear piecewise segments from time-series, however, they are not intended to capture regimes (i.e., high-level distinct patterns in time-series).
- DynaMMo [18] is capable of compression and segmentation, but, it cannot find similar sequence patterns.
- HHMM [9] and BP-AR-HMM [11] are the stochastic models that can describe the high-level dynamics of sequences. However, they require human experts for parameter and model structure settings, *and*, they do not focus on scalability.

Most importantly, none of above are parameter-free methods.

1.3 Running examples

Here, we briefly describe application domains and provide some illustrative, intuitive examples of the usefulness of our approach.

Sensor data management. Many real-life applications such as inventory management systems [6], emergency medical care [30], and thermal management in data centers [17] rely on data captured from physical sensor devices. For example, consider motion capture sensors, which generate a collection of numerical attributes of kinetic energy values. In this setting, every motion can be represented as a bundle of d co-evolving sequences. Our approach can summarize all these sequences, and find similar motion groups without using annotations or other meta-data; e.g., *"Are there any distinct patterns?", "If yes, how many and what kind?"*

Web-click analysis. Consider a large number of clicks on web sites, where each click has a list of attributes, e.g., *URL, user, time*. Suppose that we record this information for all users, every minute. For this huge collection of web-click logs, we would like to find typical patterns of user behavior; e.g., *"Are there any daily/weekly periodicities or other distinct patterns in the clicking events?"*, *"How many visitors can we expect on weekends?"*

Social media marketing. Web content such as blogs, online news and SNS has been attracting considerable interest for business purposes as well as personal use. One of our motivating applications is monitoring online social activities. Assume that we are analyzing the online keyword search activities, such as Google, where each sequence represents search volumes related to the keyword over time. In this case, web-site owners and manufacturers could find "extreme" behavior and events (e.g., disease pandemics and new product releases), as well as typical daily activity patterns.

1.4 Contributions

The main contribution of this work is the design and analysis of AUTOPLAIT, which has the following desirable properties:

1. **Effective**: it correctly estimates the count of regimes (e.g., dance steps), the type of regimes, and the count and location of transition points (segments), in several diverse settings. Also, it guarantees to find the optimal location of each transition point.
2. **Sense-making**: thanks to our novel model proposed in section 4, it gives a high-level summary (i.e., set of regimes) that are easier for a human to understand and interpret (please see Figure 1, "beaks", "wings", etc).
3. **Parameter-free**: it is fully automatic, requiring no "magic numbers" (no user intervention), no estimate for the count of segments, no estimate for the count of regimes.
4. **Scalable**: it scales linearly with the input size, and thus is applicable to very large sequences.

Outline. The rest of the paper is organized in the conventional way. Next, we describe related work and background, followed by definitions, the proposed method, experiments and conclusions.

2. BACKGROUND

Here, we introduce some necessary background material.

2.1 Related work

We provide a survey of the related literature, which falls broadly into three categories: (1) probabilistic model estimation, (2) pattern discovery in time series, and (3) summarization and clustering.

Probabilistic model estimation. Recently significant progress has been made on understanding the theoretical issues surrounding learning probabilistic models. Hidden Markov models (HMMs) have

been used in various research areas including speech recognition [35], and biological data analysis [8]. As regards HMM-based approaches for large time-evolving sequences, [16] presented a system for executing event queries over Markovian streams, generated from RFID sensors, while [12] proposed a fast search algorithm for large HMM datasets.Very recently, Wang et al. [34] improved [14], and presented a pattern-based hidden Markov model (pHMM). The pHMM is a new dynamical model for time-series segmentation and clustering, and provides a piecewise linear representation. It converts a time-series into a sequence of line segments, and learns a Markov model from the sequence. Regarding the hierarchical modeling, Fine et al. [9] presented a hierarchical HMM (HHMM), which is a recursive hierarchical generalization of the HMMs, while Fox et al. proposed the beta process autoregressive HMM [11] and variations [10], which are based on a Bayesian approach to learning Markov switching processes. These methods are the stochastic models that can describe the high-level dynamics of sequences, however, they require expert users for the parameter-tuning and the model structure setting, and also, they do not focus on scalability.

Pattern discovery in time series. In recent years, there has been an explosion of interest in mining time series [4, 7, 29, 25]. Traditional approaches applied to data mining include auto-regression (AR) or linear dynamical systems (LDS), Kalman filters (KF) and variants [13, 19, 31]. Li et al. [18] developed DynaMMo, which is a scalable algorithm for co-evolving time sequences with missing values. DynaMMo is based on a linear dynamical system, and the algorithm has the ability to segment a data sequence. The work described in [31] presents an approach for recursively predicting motion sequence patterns. For web-click and social media analysis, Agarwal et al. [1] exploit the Gamma-Poisson model to estimate "click-through rates" in the context of content recommendation, while [21] studies the rise and fall patterns in the information diffusion process through online social media. TriMine [20] is a scalable method for forecasting co-evolving multiple (thousands of) sequences. Mueen et al. studied time series motifs [22], while Rakthanmanon et al. [26] proposed a novel algorithm to accelerate a similarity search for "trillions of time-series" under the DTW distance.

Summarization and clustering. Work on summarization and clustering is remotely related to this work. A huge number of clustering techniques have been proposed including CLARANS [23], BIRCH [36], and TRACLUS [15]. As regards parameter-free mining, the work reported in [3] proposes OCI, which is a parameter-free clustering approach based on exponential power distributions (EPD). Related work [5, 32], focuses on summarization and clustering based on the MDL principle.

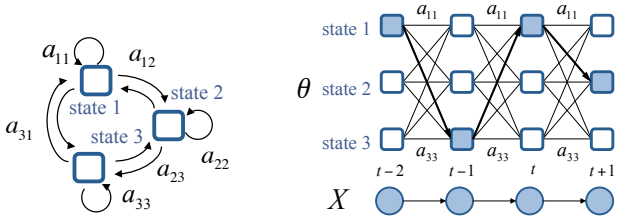
In summary, none of the above methods focus on (1) finding distinct high-level patterns from a large set of co-evolving sequences, (2) without any parameters, (3) in an efficient way.

2.2 Review of HMM

Although our proposed framework allows any model to take advantage of the compact representation of time-series data, we use the hidden Markov model (HMM) throughout this paper for simplicity. An HMM is a statistical model in which the system being modeled is assumed to be a Markov process with hidden states. It is widely used in many applications such as speaker recognition and the analysis of biological sequences.

An HMM is composed of the following probabilities: initial state probabilities $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{k}$, state transition probabilities $\mathbf{A} = \{a_{ij}\}_{i,j=1}^{k}$, and output probabilities $\mathbf{B} = \{b_i(\boldsymbol{x})\}_{i=1}^{k}$.

Likelihood function. Given a model $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$ and an input sequence $\boldsymbol{X}$, the likelihood value $P(\boldsymbol{X}|\boldsymbol{\theta})$ is computed as follows:



(a) State transition diagram (b) State transition over time

Figure 2: Illustration of the HMM structure. Given a bundle $\boldsymbol{X}$ and a model $\boldsymbol{\theta}$ (here, hidden states $k = 3$, sequence length $n = 4$), the colored states in (b) denote the Viterbi path.

$$P(\boldsymbol{X}|\boldsymbol{\theta}) = \max_{1 \le i \le k} \{p_i(n)\}$$

$$p_i(t) = \begin{cases} \pi_i b_i(\boldsymbol{x}_1) & (t = 1) \\ \max_{1 \le j \le k} \{p_j(t-1) a_{ji}\} b_i(\boldsymbol{x}_t) & (2 \le t \le n) \end{cases} \quad (1)$$

where $p_i(t)$ is the maximum probability of state i at time t, and n is the length of $\boldsymbol{X}$. The likelihood is computed based on the "*trellis diagram*" shown in Figure 2, where states lie on the vertical axis, and sequences are aligned along the horizontal axis. The likelihood is computed using a dynamic programming approach, called the Viterbi algorithm, which maximizes the probabilities from previous states (i.e., each state probability is computed using all previous state probabilities, associating transition probabilities, and output probabilities). The state sequence, which gives the likelihood, is called the Viterbi path. Note that the Viterbi algorithm requires $O(ndk^2)$ time to compute the likelihood value, where n and d are the length and dimension of $\boldsymbol{X}$, and k is the number of states.

Learning the parameters. Given a set of time series $\boldsymbol{X}$, the Baum-Welch algorithm is an efficient way to find the maximum likelihood model parameter set: $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$. The complexity of the algorithm is $O(ndk^2)$. For more details, please see, e.g., [2].

3. PROBLEM FORMULATION

In this paper we tackle a challenging problem, namely fully-automatic mining, and specifically, *CAPS* (**C**ompression / **A**nomaly-detection / **P**attern-discovery / **S**egmentation) for large co-evolving time series. Let us begin by defining a few key concepts.

DEFINITION 1 (BUNDLE). *Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a set of d co-evolving time sequences, where $\boldsymbol{x}_t$ is a d-dimensional vector at time-tick t. We refer to $\boldsymbol{X}$ as a bundle.*

Given a bundle $\boldsymbol{X}$, we want to convert $\boldsymbol{X}$ into a set of m non-overlapping segments $\mathcal{S} = \{s_1, \ldots, s_m\}$ where s_i consists of starting and end positions of the i-th segment (i.e., $s_i = \{t_s, t_e\}$). We also want to find a set of distinct patterns of segments by assigning each segment to a segment group, namely, the regime of segments.

DEFINITION 2 (REGIME). *Let r denote the desired number of segment groups. Each segment s is assigned to one of these groups. We define each such segment group as a regime, which is represented by a statistical model $\boldsymbol{\theta}_i$ ($i = 1, \ldots, r$).*

For example, in Figure 1, the motion consists of $m = 8$ segments, each of which belongs to one of the $r = 4$ regimes, (i.e., "beaks", "winds", "tail feathers" and "claps").

DEFINITION 3 (SEGMENT-MEMBERSHIP). *Let $\mathcal{F}$ be a list of m integers, $\mathcal{F} = \{f_1, \ldots, f_m\}$, where f_i is the regime that the i-th segment belongs to (i.e., $1 \le f_i \le r$).*

In Figure 1, the first segment belongs to regime #4 ("beaks"), and the second segment belongs to #3 ("wings"), and so on. These steps are repeated twice. That is, the segment-membership is, $\mathcal{F} = \{4, 3, 2, 1, 4, 3, 2, 1\}$.

Our goal is to determine the number of segment cuts and their positions automatically, in a scalable way, and describe each segment compactly. Thus, we want to provide a coding scheme with an appropriate cost function, and also achieve the optimal segmentation with probabilistic model learning to minimize the cost, without any knowledge of the characteristics of the sequences. We formally define the problem (i.e., *CAPS*: **C**ompression / **A**nomaly-detection / **P**attern-discovery / **S**egmentation) as follows:

PROBLEM 1 (FULLY-AUTOMATIC CAPS). **Given** *a set of co-evolving sequences (bundle)* $\boldsymbol{X}$, **find** *a compact description that best summarizes* $\boldsymbol{X}$, *that is,*

1. *determine the count* m *of segments and their positions, i.e.,* $\mathcal{S} = \{s_1, \ldots, s_m\}$,
2. *determine the count* r *of regimes and their segment-membership, i.e.,* $\mathcal{F} = \{f_1, \ldots, f_m\}$,
3. *estimate the model parameters of the* r *distinct regimes, i.e.,* $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r\times r}\}$,

to optimize the cost function of Equation 6, that is, the total lossless description length.

Here, we should note that the model parameter set $\boldsymbol{\Theta}$ consists of the HMM parameters of r regimes, i.e., $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r\}$, *and*, one additional matrix, namely, the *regime transition matrix* $\boldsymbol{\Delta}_{r\times r}$. We will describe the details about the regime transition matrix (Definition 5) and the cost function (Equation 6) in section 4.

As described in Problem 1, our final goal is to find a good representation of $\boldsymbol{X}$, which we refer to as a *candidate solution*.

DEFINITION 4 (CANDIDATE SOLUTION). *Let* $\mathcal{C}$ *be a complete set of parameters (namely* $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$*) that describe a segmentation, the regimes each segment belongs to, the parameters of each regime.*

We want to provide a good solution $\mathcal{C}$ to the problem. The essential questions are: how should we decide the number of segments and regimes, i.e., m and r, respectively? How can we generate 'informative' regimes and assign segments to their proper regimes? We address these questions without any parameter fine tuning.

4. DATA COMPRESSION AND SUMMARIZATION

In this section we present our model for dealing with Problem 1. There are two main ideas behind our model:

1. **Multi-level chain model (MLCM)**: We group states of the HMMs into regimes, and regimes into super-regimes, etc. Figure 3 illustrates our approach: The blue states are grouped into regime 1 (which could correspond, say, to the "beaks" phase of the dance of Figure 1), and similarly for the red states (regime 2, e.g., "wings"). Here, we stay with two levels only, 'states' and 'regimes'. Within each regime, we have a transition matrix $\mathbf{A}$; our novelty is that we allow across-regime transitions, with *regime transition probabilities* $\boldsymbol{\Delta}_{r\times r}$, which is an $r \times r$ transition matrix, where r is the count of regimes ($r = 2$, in Figure 3).
2. **Model description cost**: We use the MDL (minimum description length) principle, to choose among alternative segmentation and regime-descriptions. Although important, the MDL principle is a model selection criterion based on lossless compression principles, which does not directly address our problem. Thus, we define a new coding scheme for the segmentation and summarization of a given bundle $\boldsymbol{X}$. Specifically, we propose (a) a novel cost function (i.e., Equation 6) to estimate the description cost of a given candidate and (b) an efficient algorithm to find an optimal segmentation.

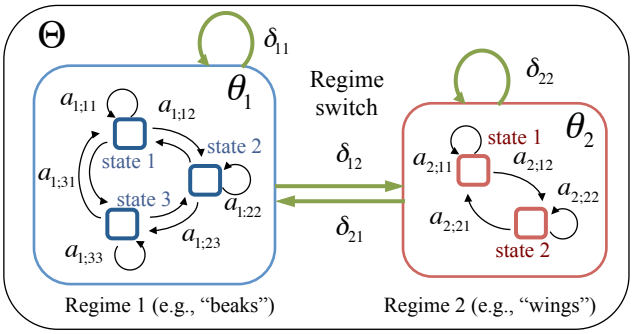


Figure 3: Multi-level transition diagram for the $r = 2$ regime transition model Θ; blue regime (regime 1, say, "beaks") and red regime (say, "wings"), each with three and two internal states. The black and green arrows indicate "within"-regime and "across"-regime transitions, respectively.

Overall, our model consists of two parts: (1) MLCM (**M**ulti-**L**evel **C**hain **M**odel), a statistical model for dealing with *multi-level transitions*, and (2) a cost model, which enables our algorithm to estimate probabilistic parameters *automatically*. These two parts are described next.

4.1 MLCM: multi-level chain model

One of the novelties of this paper is the multi-level chain model, where we propose grouping states into 'regimes' as shown in Figure 3. Here we stay with two levels only, but it is possible to employ higher-level models. Our idea is best described with an example, as in Figure 3. There, we have five states total, but instead of having a 5×5 transition matrix, we propose grouping states into super-states, that we refer to as 'regimes'. In Figure 3, suppose we group the three blue states together (say, they describe the "beaks" motion of the dance example of Figure 1), with the remaining two states grouped into the red regime (say, "wings" of Figure 1). Each regime has its own transition matrix ($a_{1;ji} \in \mathbf{A}_1$ and $a_{2;ji} \in \mathbf{A}_2$, in our running example, shown as the black arrows), and there is a second-level transition matrix ($\delta_{vu} \in \boldsymbol{\Delta}_{2\times 2}$) which governs the across-regime transitions (shown as the green arrows).

DEFINITION 5 (REGIME TRANSITION MATRIX). *Let* $\boldsymbol{\Delta}_{r\times r}$ *denote a transition probability matrix of* r *regimes, where each element* $\delta_{ij} \in \boldsymbol{\Delta}$ *is the regime transition probability from the* i*-th regime to the* j*-th regime.*

Matrix $\boldsymbol{\Delta}$ has all the properties of a transition matrix: all entries $\delta_{i,j}$ are probabilities, and thus in the range (0,1), and the rows sum up to 1, i.e., $0 \le \delta_{ij} \le 1$, with $\sum_j \delta_{ij} = 1$.

Consequently, we define a set of model parameters of r regimes as $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r\times r}\}$, where $\boldsymbol{\theta}_i$ are the parameters of the i-th regime, i.e., $\boldsymbol{\theta}_i = \{\boldsymbol{\pi}_i, \mathbf{A}_i, \mathbf{B}_i\}$. [3]

[3] In our setting, we assume a Gaussian distribution for the output probability, which is able to handle multi-dimensional vectors at each time-tick (i.e., $\mathbf{B} = \{\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)\}_{i=1}^k$).

4.2 What is a good description?

We introduce a new, intuitive coding scheme, which is based on lossless compression principles. In short, the goodness of the model can be roughly described as follows: $Cost_T = Cost(\mathcal{M}) + Cost(\boldsymbol{X}|\mathcal{M})$, where $Cost(\mathcal{M})$ shows the cost of describing the model $\mathcal{M}$, and $Cost(\boldsymbol{X}|\mathcal{M})$ represents the cost of describing the data $\boldsymbol{X}$ given the model $\mathcal{M}$.

4.2.1 Model description cost

The description complexity of our model consists of the following terms:

- The number of time-ticks n and the number of dimensions d require $\log^*(n) + \log^*(d)$ bits. [4]
- The number of segments m and the number of regimes r require $\log^*(m) + \log^*(r)$ bits.
- The assignments of the segments to regimes (i.e., segment-membership of these regimes) require $m \log(r)$.
- The length of each segment s, needs $\sum_{i=1}^{m-1} \log^* |s_i|$ bits.
- The model parameters of r regimes need $Cost_M(\boldsymbol{\Theta})$, i.e.,

$$Cost_M(\boldsymbol{\Theta}) = \sum_{i=1}^{r} Cost_M(\boldsymbol{\theta}_i) + Cost_M(\boldsymbol{\Delta}). \quad (2)$$

Here, a single model $\boldsymbol{\theta}$ requires $\log^*(k)$ for the number of hidden states k, and the model cost (i.e., $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$). That is,

$$Cost_M(\boldsymbol{\theta}) = \log^*(k) + c_F \cdot (k + k^2 + 2kd), \quad (3)$$

where c_F is the floating point cost[5]. Similarly, the regime transition requires the cost of $Cost_M(\boldsymbol{\Delta}) = c_F \cdot r^2$.

4.2.2 Coding cost of whole bundle

After estimating a representation of the bundle $\boldsymbol{X}$, we need a reliable approach to judge the accuracy of the fit. Data compression using Huffman coding assigns a number of bits to each value in $\boldsymbol{X}$, which is the logarithm of the inverse of probability (i.e., the negative log-likelihood) of the values. The coding cost of $\boldsymbol{X}$, Huffman-coded with a given model $\boldsymbol{\theta}$ is:

$$Cost_C(\boldsymbol{X}|\boldsymbol{\theta}) = \log_2 \frac{1}{P(\boldsymbol{X}|\boldsymbol{\theta})} = -\ln P(\boldsymbol{X}|\boldsymbol{\theta}), \quad (4)$$

where $P(\boldsymbol{X}|\boldsymbol{\theta})$ is the likelihood of $\boldsymbol{X}$, described in Equation 1. Given a whole bundle $\boldsymbol{X}$, and the model parameters of r regimes, $\boldsymbol{\Theta}$, the total cost of data compression is

$$\begin{aligned} Cost_C(\boldsymbol{X}|\boldsymbol{\Theta}) &= \sum_{i=1}^{m} Cost_C(\boldsymbol{X}[s_i]|\boldsymbol{\Theta}) \\ &= \sum_{i=1}^{m} -\ln(\delta_{vu} \cdot (\delta_{uu})^{|s_i|-1} \cdot P(\boldsymbol{X}[s_i]|\boldsymbol{\theta}_u)), \end{aligned} \quad (5)$$

where the i-th and $(i-1)$-th segments are governed by the u-th and v-th regimes, respectively (i.e., $f_i = u$ and $f_{i-1} = v$), and $f_0 = f_1$. Also, $\boldsymbol{X}[s_i]$ is the sub-bundle of segment s_i, and $P(\boldsymbol{X}[s_i]|\boldsymbol{\theta}_u)$ is the likelihood of s_i. Notice that $\boldsymbol{\theta}_u$ is the regime that the segment s_i belongs to (i.e., the regime that describes it best).

[4]Here, $\log^*$ is the universal code length for integers, defined as $\log^*(x) \approx \log_2(x) + \log_2 \log_2(x) + \ldots$, where only the positive terms are included in the sum [27].

[5]We used 4×8 bits in our setting.

Table 2: Symbols and definitions.

Symbol	Definition
for bundle	
n	Number of time ticks
d	Number of dimensions
$\boldsymbol{X}$	Bundle: d co-evolving time sequences
for segments	
m	Number of segments in $\boldsymbol{X}$
$\mathcal{S}$	Segment set in $\boldsymbol{X}$, i.e., $\mathcal{S} = \{s_1, \ldots, s_m\}$
$\mathcal{F}$	Segment-membership i.e., $\mathcal{F} = \{f_1, \ldots, f_m\}$
for regimes	
r	Number of regimes in $\boldsymbol{X}$
$\boldsymbol{\Theta}$	Model parameters of r regimes, i.e., $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r\times r}\}$
$\boldsymbol{\theta}_i$	Model parameters governing i-th regime
k_i	Number of hidden states in $\boldsymbol{\theta}_i$
$\boldsymbol{\Delta}_{r\times r}$	Regime transitions, i.e., $\boldsymbol{\Delta} = \{\delta_{ij}\}_{i,j=1}^{r}$
for coding scheme	
$\mathcal{C}$	Candidate solution i.e., $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$
$Cost_M(\boldsymbol{\Theta})$	Model description cost of $\boldsymbol{\Theta}$
$Cost_C(\boldsymbol{X}\|\boldsymbol{\Theta})$	Coding cost of $\boldsymbol{X}$ given $\boldsymbol{\Theta}$
$Cost_T(\boldsymbol{X};\mathcal{C})$	Total cost of $\boldsymbol{X}$ given $\mathcal{C}$

4.2.3 Putting it all together

The total code length for bundle $\boldsymbol{X}$ with respect to a given candidate solution: $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$ can be described as follows:

$$\begin{aligned} Cost_T(\boldsymbol{X};\mathcal{C}) &= Cost_T(\boldsymbol{X}; m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}) \\ &= \log^*(n) + \log^*(d) + \log^*(m) + \log^*(r) + m\log(r) \\ &\quad + \sum_{i=1}^{m-1} \log^* |s_i| + Cost_M(\boldsymbol{\Theta}) + Cost_C(\boldsymbol{X}|\boldsymbol{\Theta}) \end{aligned} \quad (6)$$

Thus, our goal is to find the best combination of segments and regimes, to minimize the above function. This is exactly the focus of section 5.

We illustrate our ideas with a simple example, with a "bundle" consisting of a single sequence (d=1), with categorical values (although our upcoming algorithms are mainly for numerical values.

EXAMPLE 1. *Let us assume that we are given the following bundle $\boldsymbol{X} = (a, a, a, a, b, c, b, c, b, a, a, a, a, a, a)$, where $n = 15$, $d = 1$. We illustrate how to find the best summarization to achieve the minimum cost.*

Given a regime $\boldsymbol{X}$, the most straightforward solution is to consider the whole bundle $\boldsymbol{X}$ as a single segment, and create a single regime $\boldsymbol{\theta}$ that describes all the sequences (we refer to it as "single-transition (ST)" model), i.e.,

$$\mathcal{S}_{ST} = \{s_1\} = \{\boldsymbol{X}\}, \boldsymbol{\Theta}_{ST} = \{\boldsymbol{\theta}, \boldsymbol{\Delta} = [1.0]\}, \mathcal{F}_{ST} = \{f_1 = 1\},$$
$$\boldsymbol{\theta} = \{\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}\},$$

where, the model $\boldsymbol{\theta}$ starts with 'aa…', and then follows transition matrix $\mathbf{A}$, e.g., 'bcbc…'.

Our model, MLCM*, on the other hand, is much more intuitive. We create three segments and two regimes, i.e.,*

$$\mathcal{S}_{\text{MLCM}} = \{s_1, s_2, s_3\} = \{\{a, a, a, a\}, \{b, c, b, c, b\}, \{a, a, \cdots, a\}\},$$
$$\boldsymbol{\Theta}_{\text{MLCM}} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta} = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}\}, \mathcal{F}_{\text{MLCM}} = \{1, 2, 1\},$$

$$\boldsymbol{\theta}_1 = \{\pi = [1.0], \mathbf{A} = [1.0], \mathbf{B} = [a]\},$$
$$\boldsymbol{\theta}_2 = \{\pi = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b \\ c \end{bmatrix}\},$$

where, regime $\boldsymbol{\theta}_1$ is the infinite Markov chain that gives 'a' forever, and regime $\boldsymbol{\theta}_2$ strictly alternates between emitting 'b' and 'c'.

We now compare the costs of two candidate segmentations and regime-memberships. From Equation 6, we have

(a) $Cost_M(\boldsymbol{\Theta}_{ST}) = 612$, $Cost_C(\boldsymbol{X}|\boldsymbol{\Theta}_{ST}) = 8$, $Cost_T(\boldsymbol{X}; m = 1, r = 1, \mathcal{S}_{ST}, \boldsymbol{\Theta}_{ST}, \mathcal{F}_{ST}) = \underline{632}$.

(b) $Cost_M(\boldsymbol{\Theta}_{\text{MLCM}}) = 580$, $Cost_C(\boldsymbol{X}|\boldsymbol{\Theta}_{\text{MLCM}}) = 8$, $Cost_T(\boldsymbol{X}; m = 3, r = 2, \mathcal{S}_{\text{MLCM}}, \boldsymbol{\Theta}_{\text{MLCM}}, \mathcal{F}_{\text{MLCM}}) = \underline{619}$.

Consequently, even in this small, toy example, the second alternative (i.e., MLCM*) gives better cost.*

5. OPTIMIZATION ALGORITHM

In the previous section, we have seen how we can estimate the goodness of the segmentation and summarization, if we are given a candidate solution $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$. Now, the question is *how to find* an optimal solution? This is exactly the focus of this section.

5.1 Overview

We employ our model to automatically select the number of segments and regimes. It follows the assumption that the more we can compress the data, the more we can learn about its underlying patterns. The optimal solution corresponds to the number of segments m, the number of regimes r, a set of segments $\mathcal{S}$, model parameters $\boldsymbol{\Theta}$, and their membership $\mathcal{F}$, such that the total resulting coding cost, namely, $Cost_T(\boldsymbol{X}; m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F})$ is minimized. Equation 6 provides a strong theoretical foundation, and it can point out the best candidate among many for segmentation.

So how can we find good candidates? For simplicity, let's focus on a simple step first, where we assume that (1) we are given two regimes with fixed model parameters. We then (2) estimate the regime parameters, and even (3) the number of regimes. We divide the question into three steps:

1. **CutPointSearch (inner-most loop)**: Find good cut points to create two sets of segments, $\mathcal{S}_1$ and $\mathcal{S}_2$, when given the number of regimes ($r = 2$) and regime model parameters.
2. **RegimeSplit (inner loop)**: Estimate good regime parameters ($\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, and $\boldsymbol{\Delta}$), for a fixed number of regimes $r = 2$.
3. **AutoPlait (outer loop)**: Search for the best number of regimes ($r = 2, 3, 4, \ldots$).

Among all possible r values, and all possible cut point locations, our algorithm picks the arrangement with the smallest total compression cost, as suggested by our cost model.

Figure 4 provides an overview of AUTOPLAIT involving the segmentation and segment-menbership of regimes. Starting with $r = 2$ (at iteration 1), our method successively increases r and m, and finds progressively better representations of the bundle $\boldsymbol{X}$. During the splitting of regimes and segments, the cost is greatly reduced (see the cost curve, Figure 4 (d)).

5.2 Cut-point search (inner-most loop)

Let's begin with the simplest case. We assume that we are given a bundle $\boldsymbol{X}$, and *also*, the model parameters of two regimes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$. Our first goal is to detect the switching positions (i.e., cut points) of regimes according to the model parameters, efficiently with only a single scan. Figure 5 shows the single cut-point case, where we switch from the blue regime $\boldsymbol{\theta}_1$ to the red regime $\boldsymbol{\theta}_2$. More specifically, we want to find multiple cut points, that divide $\boldsymbol{X}$ into two

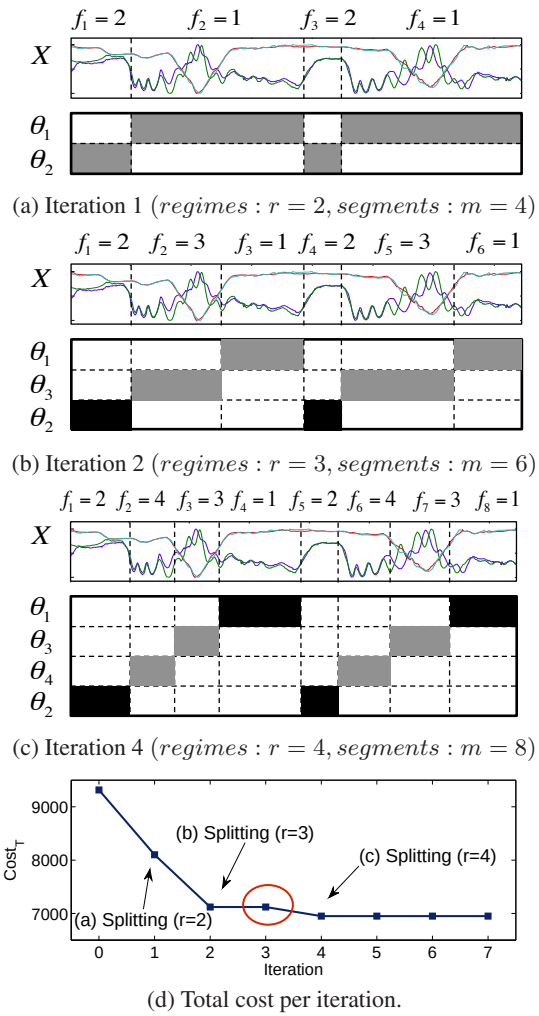


Figure 4: Overview of the workflow of AUTOPLAIT: (a)-(c) starting with bundle $\boldsymbol{X}$, it iteratively finds the segment groups (i.e., regimes), and their segments (grey boxes). (d) At each iteration, new regimes are created by splitting an existing regime to improve the coding cost. Note that at iteration 3 (red circle in (d)), the cost stays the same as at iteration 2, because of the unsuccessful attempt (θ_1 is not split at iteration 3).

sets of segments (i.e., $\mathcal{S}_1$ and $\mathcal{S}_2$) so that we minimize the sum of the cost (i.e., in Equation 5). Clearly the two sets of segments alternate: the odd ones belong to one regime, and the even ones to the other. For example, at iteration #1 in Figure 4, $\mathcal{S}_1 = \{s_2, s_4\}$ belong to $\boldsymbol{\theta}_1$, $\mathcal{S}_2 = \{s_1, s_3\}$ to $\boldsymbol{\theta}_2$.

How do we go about finding multiple cut points? An elementary concept that we need to introduce is a multi-level transition diagram, specifically a two-level transition diagram over the time domain. We are given two regimes $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. We then connect their transition diagrams (see Figure 5), and compare these regimes in terms of coding cost. To compute the coding cost $Cost_C(\boldsymbol{X}|\boldsymbol{\Theta}) = -\ln P(\boldsymbol{X}|\boldsymbol{\Theta})$, we present a dynamic programming approach and show how to exploit the two-level transition diagram.

5.2.1 Algorithm

Formally, given a bundle $\boldsymbol{X}$, two regimes $\boldsymbol{\theta}_1 = \{\boldsymbol{\pi}_1, \mathbf{A}_1, \mathbf{B}_1\}$, $\boldsymbol{\theta}_2 = \{\boldsymbol{\pi}_2, \mathbf{A}_2, \mathbf{B}_2\}$, and a regime transition matrix $\boldsymbol{\Delta} = \{\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}\}$, the likelihood value $P(\boldsymbol{X}|\boldsymbol{\Theta})$ is computed as follows:

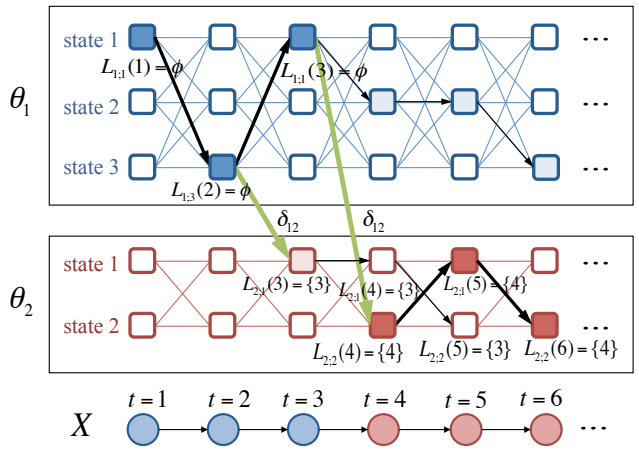


Figure 5: Illustration of CutPointSearch. Given a bundle $\boldsymbol{X}$ and two models $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ (here, duration $n = 6$), our algorithm requires only a single scan to detect the regime cut point (i.e., at time-tick $t = 4$). Note that the colored cells indicate the optimal states (i.e., Viterbi path). At time-tick $t = 3$ and 4 in the red regime, the algorithm finds $\mathcal{L}_{2;1}(3) = \{3\}$ and $\mathcal{L}_{2;2}(4) = \{4\}$ as the first and second cut-point candidates. and at time-tick $t = 6$, $\mathcal{L}_{2;2}(6) = \{4\}$ is chosen as the optimal cut point.

$$P(\boldsymbol{X}|\boldsymbol{\Theta}) = \max \begin{cases} \max_{1\le i\le k_1}\{p_{1;i}(n)\} & // \text{ regime } \boldsymbol{\theta}_1 \\ \max_{1\le u\le k_2}\{p_{2;u}(n)\} & // \text{ regime } \boldsymbol{\theta}_2 \end{cases} \quad (7)$$

$$p_{1;i}(t) = \max \begin{cases} \delta_{21} \cdot \max_v\{p_{2;v}(t-1)\} \cdot \pi_{1;i} \cdot b_{1;i}(\boldsymbol{x}_t) \\ \qquad // \text{ regime switch from } \boldsymbol{\theta}_2 \text{ to } \boldsymbol{\theta}_1 \\ \delta_{11} \cdot \max_j\{p_{1;j}(t-1) \cdot a_{1;ji}\} \cdot b_{1;i}(\boldsymbol{x}_t) \\ \qquad // \text{ staying at regime } \boldsymbol{\theta}_1 \end{cases} \quad (8)$$

$$p_{2;u}(t) = \max \begin{cases} \delta_{12} \cdot \max_j\{p_{1;j}(t-1)\} \cdot \pi_{2;u} \cdot b_{2;u}(\boldsymbol{x}_t) \\ \qquad // \text{ regime switch from } \boldsymbol{\theta}_1 \text{ to } \boldsymbol{\theta}_2 \\ \delta_{22} \cdot \max_v\{p_{2;v}(t-1) \cdot a_{2;vu}\} \cdot b_{2;u}(\boldsymbol{x}_t) \\ \qquad // \text{ staying at regime } \boldsymbol{\theta}_2 \end{cases} \quad (9)$$

where $p_{1;i}(t)$ is the maximum probability of state i of regime $\boldsymbol{\theta}_1$ at time t, and $p_{2;u}(t)$ is that of state u of $\boldsymbol{\theta}_2$ at time t. As an initial setting, at time $t = 1$, the probability for each regime is set at:

$$\begin{aligned} p_{1;i}(1) &= \delta_{11} \cdot \pi_{1;i} \cdot b_{1;i}(\boldsymbol{x}_1) \\ p_{2;u}(1) &= \delta_{22} \cdot \pi_{2;u} \cdot b_{2;u}(\boldsymbol{x}_1) \end{aligned} \quad (10)$$

In Equation 8, the first row shows the probability for the case of regime switch (i.e., from regime $\boldsymbol{\theta}_2$ to regime $\boldsymbol{\theta}_1$), and the second row shows the case of state transition in regime $\boldsymbol{\theta}_1$, where:

- δ_{21}: regime transition probability from $\boldsymbol{\theta}_2$ to $\boldsymbol{\theta}_1$,
- $\max_v\{p_{2;v}(t-1)\}$: probability of the best state of $\boldsymbol{\theta}_2$ at time $t-1$,
- $\pi_{1;i}$: initial probability of state i of $\boldsymbol{\theta}_1$,
- $b_{1;i}(\boldsymbol{x}_t)$: output probability of $\boldsymbol{x}_t$ for state i of $\boldsymbol{\theta}_1$,
- $a_{1;ji}$: transition probability from state j to state i in $\boldsymbol{\theta}_1$.

Equation 9 also shows the maximum probability of regime $\boldsymbol{\theta}_2$, which can be computed similarly.

The above approach is a good first step that can tell us the coding cost with respect to the two regimes. Our next question is how to identify the cut point of each segment. We thus propose keeping track of candidate cut points during the likelihood computation.

Let $\mathcal{L} = \{l_1, l_2, \ldots, l_{m-1}\}$ be the set of cut point locations, where m is the number of segments, and l_i is the i-th cut point, i.e., $1 \le l_i \le n$. Our idea is that we retain a candidate cut-point set for every state of the two regimes. Specifically,

$$\mathcal{L}_{1;i}(t) = \begin{cases} \mathcal{L}_{2;v}(t-1) \cup \{t\} & // \text{ switch from } \boldsymbol{\theta}_2 \text{ to } \boldsymbol{\theta}_1 \\ \mathcal{L}_{1;j}(t-1) & // \text{ staying at regime } \boldsymbol{\theta}_1 \end{cases} \quad (11)$$

$$\mathcal{L}_{2;u}(t) = \begin{cases} \mathcal{L}_{1;j}(t-1) \cup \{t\} & // \text{ switch from } \boldsymbol{\theta}_1 \text{ to } \boldsymbol{\theta}_2 \\ \mathcal{L}_{2;v}(t-1) & // \text{ staying at regime } \boldsymbol{\theta}_2 \end{cases} \quad (12)$$

where $\mathcal{L}_{1;i}(t)$ shows the candidate cut point location(s) for state i of $\boldsymbol{\theta}_1$ at time-tick t, and $\mathcal{L}_{2;u}(t)$ shows the candidate(s) for state u of $\boldsymbol{\theta}_2$, which are updated according to the likelihood computation in Equations 8 and 9. We add time t as a candidate to the cut-point set if the regime is switched to the other. Algorithm 1 shows the overall procedure for the cut-point search. The cut-point set $\mathcal{L}$ is transmitted as a message through the optimal path of the diagram. At time $t = n$, we choose the best cut-point set $\mathcal{L}_{best}$, from $\mathcal{L}_{1;i}(n)$ and $\mathcal{L}_{2;u}(n)$ of all states i and u, so as to maximize $P(\boldsymbol{X}|\boldsymbol{\Theta})$.

Algorithm 1 CutPointSearch ($\boldsymbol{X}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}$)

```
Input: Bundle X, model parameters of two regimes {θ1, θ2, Δ}
Output: (a) Number of segments assigned to each regime, m1, m2
        (b) Segment sets of two regimes S1, S2
/* Compute p1;i(t) and p2;u(t) */
for t = 1 : n do
    Compute p1;i(t) for state i = 1, ..., k1;   /* Equations 8 and 10 */
    Compute p2;u(t) for state u = 1, ..., k2;   /* Equations 9 and 10 */
    Update L1;i(t) for state i = 1, ..., k1;    /* Equation 11 */
    Update L2;u(t) for state u = 1, ..., k2;    /* Equation 12 */
end for
/* Divide into two sets of segments S1, S2 */
Choose the best cut-point set L_best;
ts = 1; /* Starting position of first segment */
for each cut point li in L_best do
    Create segment si = {ts, li};
    if i is odd then
        Add si into S1;  m1 = m1 + 1;
    else
        Add si into S2;  m2 = m2 + 1;
    end if
    ts = li;
end for
return {m1, m2, S1, S2};
```

EXAMPLE 2. *We illustrate how this is done using Figure 5. At time $t = 1$ and $t = 2$, the algorithm shows that $p_{1;1}(1)$ and $p_{1;3}(2)$ in $\boldsymbol{\theta}_1$ provide the highest probabilities. At $t = 3$, it finds a candidate cut point from $p_{1;3}(2)$ to $p_{2;1}(3)$ (i.e., the first green arrow from $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_2$) and then retains the position $\mathcal{L}_{2;1}(3) = \{3\}$ as a candidate cut point. Similarly, it finds the second candidate point, and retains $\mathcal{L}_{2;2}(4) = \{4\}$. At $t = 6$, the algorithm shows that $p_{2;2}(6)$ provides the maximum probability, thus determines $\mathcal{L}_{2;2}(6) = \{4\}$ should be the best position of the regime switch.*

5.2.2 Theoretical analysis

LEMMA 1. *The CutPointSearch algorithm takes $O(ndk^2)$ time.*

PROOF. The CutPointSearch algorithm needs to compute $O(dk^2)$ numbers per time tick. The algorithm requires only a single scan to find cut points. Thus, the time complexity is $O(ndk^2)$. □

LEMMA 2. *For the given model $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}\}$, CutPointSearch guarantees the output of the optimal cut points.*

PROOF. We are now given r regimes each with k states. Let us assume that each state is linked to all $k \times r$ states of the r regimes, and $\delta'_{u,i;v,j}$ is the transition probability from state i of regime u to state j of regime v. This is equivalent to the case that we have $k \times r$ states of a single regime. The optimal path with the minimum cost can be found by using a dynamic programming approach. In the two-level transition of our model, regime u is linked to regime v,

thus transition probability can be considered as $\delta_{uv} = \delta'_{u,i;v,j}$ for any possible combinations of states i and j.

Therefore, our algorithm provides the optimal path and optimal cut points with the minimum cost. □

5.3 Regime parameter estimation (inner loop)

Until now, we have assumed that the model parameters of two regimes (i.e., $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$) were given. Next, we tackle an important and challenging question, namely, how can we estimate model parameters that describe these regime patterns? We would like to (a) estimate the model parameters of two regimes and (b) find the locations of all cut points, *simultaneously*.

The goal is to obtain parameters that minimize the cost of modeling the whole bundle $\boldsymbol{X}$, as described in Equation 6. However, it is difficult to minimize the modeling cost directly. We thus propose an iterative optimization algorithm (see Algorithm 2). Our algorithm searches for the optimal solution using a two-phase approach, i.e.,

- *Phase 1:* Find the cut points of segments according to the coding cost, (using Algorithm 1), and split the segments into two segment groups $\{\mathcal{S}_1, \mathcal{S}_2\}$.
- *Phase 2:* Estimate the model parameters of two regimes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$ based on the new segmentation. Here, we use the Baum-Welch algorithm to infer the HMM parameters.

Algorithm 2 RegimeSplit ($\boldsymbol{X}$)

```
Input: Bundle X
Output: (a) Number of segments assigned to each regime, m1, m2
        (b) Segment sets of two regimes, S1, S2
        (c) Model parameters of two regimes {θ1, θ2, Δ}
Initialize models θ1, θ2;   /* Equation 13 */
while improving the cost do
  /* Find segments (phase 1) */
  {m1, m2, S1, S2} = CutPointSearch (X, θ1, θ2, Δ);
  /* Update model parameters (phase 2) */
  θ1 = BaumWelch (X[S1]);
  θ2 = BaumWelch (X[S2]);
  Update regime transitions Δ;   /* Equation 14 */
end while
return {m1, m2, S1, S2, θ1, θ2, Δ};
```

Model initialization. Before we start the iterations, we need to initialize the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$ to some random values. Simplest solution would be to estimate the initial models using randomly-selected subsequences of $\boldsymbol{X}$ (i.e., use randomly-chosen starting/ending points). However, this approach might converge to a local minimum of the cost function, depending on the initial values. Thus, we propose a sampling-based approach to avoid this issue. We uniformly take several sample segments from the original data $\boldsymbol{X}$. For each sample segment s, we estimate the model parameters $\boldsymbol{\theta}_s$. We then compute the coding cost of all possible pairs of $\{\boldsymbol{\theta}_{s_1}, \boldsymbol{\theta}_{s_2}\}$, and choose the most appropriate pair $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$.

$$\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\} = \underset{\boldsymbol{\theta}_{s_1}, \boldsymbol{\theta}_{s_2} | s_1, s_2 \in \mathcal{X}}{\arg\min} Cost_C(\boldsymbol{X} | \boldsymbol{\theta}_{s_1}, \boldsymbol{\theta}_{s_2}), \tag{13}$$

where $\mathcal{X} = \{s_1, s_2, \dots\}$ is a set of samples taken from $\boldsymbol{X}$.

Model estimation. Note that the Baum-Welch algorithm requires the number of hidden states k for each model $\boldsymbol{\theta}$, and it is hard to set a reasonable k by hand. When we use a very small number for k, the model provides a poor fit to the data, and the algorithm might fail to find optimal segments. By contrast, when we use a much larger number for k, the models would give a very poor representation, known as over-fitting. So how can we determine the optimal number for k? We thus vary $k = 1, 2, 3, \dots$, and determine appropriate models so as to minimize the cost function, i.e., $Cost_M(\boldsymbol{\theta}) + Cost_C(\boldsymbol{X}[\mathcal{S}] | \boldsymbol{\theta})$.

We also need to minimize the coding cost with respect to the regime transition probabilities $\boldsymbol{\Delta}$. We treat the cut points of segments $\{\mathcal{S}_1, \mathcal{S}_2\}$ and the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$ as constant, and then compute $\boldsymbol{\Delta} = \{\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}\}$ using appropriate Lagrange multipliers with the results as follows:

$$\delta_{11} = \frac{\sum_{s \in \mathcal{S}_1} |s| - N_{12}}{\sum_{s \in \mathcal{S}_1} |s|}, \quad \delta_{12} = \frac{N_{12}}{\sum_{s \in \mathcal{S}_1} |s|}, \tag{14}$$

where $\sum_{s \in \mathcal{S}_1} |s|$ shows the total length of segments that belongs to regime $\boldsymbol{\theta}_1$, and N_{12} indicates the regime-switch count from $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_2$. We compute δ_{21}, δ_{22} similarly, but we omit the details.

5.4 AutoPlait (outer loop)

Thus far, we have discussed how to find segments and their cut points given two models/regimes (i.e., $r = 2$). Our final goal is to find multiple patterns in a large bundle without any user intervention (described in Problem 1). So how should we decide the number of segments m and regimes r? How can we assign the segments to their proper regimes?

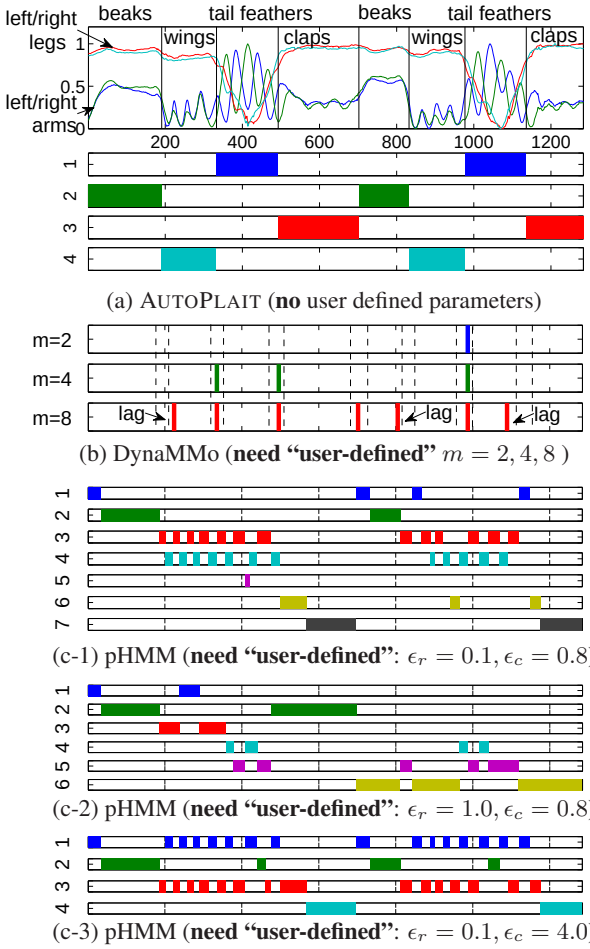
5.4.1 Algorithm

We introduce a stack-based algorithm, namely AUTOPLAIT, for fully-automatic mining. The idea is to use a greedy approach, splitting a bundle into segments, and introducing new regimes, as long as the coding cost (Equation 6) keeps decreasing. The detailed AUTOPLAIT algorithm is shown in Algorithm 3.

Algorithm 3 AUTOPLAIT ($\boldsymbol{X}$)

```
Input: Bundle X
Output: Complete set of parameters C, i.e.,
        (a) Number of segments, m
        (b) Number of regimes, r
        (c) Segment set, S = {s1, ..., sm}
        (d) Model parameters of regimes, Θ = {θ1, ..., θr; Δ}
        (e) Segment membership, F = {f1, ..., fm}
Q = ∅;  /* Q: stack for number of segments, segment set, regime */
S = ∅;   m = 0;   r = 0;   S0 = {1, n};   m0 = 1;
θ0 = BaumWelch (X[S0]);   /* Estimate model θ0 of S0 */
Push an entry {m0, S0, θ0} into Q;
while stack Q ≠ ∅ do
  Pop an entry {m0, S0, θ0} from Q;
  /* Try to refine a regime */
  {m1, m2, S1, S2, θ1, θ2, Δ} = RegimeSplit (X[S0]);
  /* Compare single regime θ0 v.s. regime pair θ1 and θ2 */
  if Cost_T(X; S0, θ0) > Cost_T(X; S1, S2, θ1, θ2) then
    /* Regime pair win - split regime */
    Push entries {m1, S1, θ1}, {m2, S2, θ2} into Q;
  else
    /* Single regime win - no more split, leave it out of the stack */
    S = S ∪ S0;   Θ = Θ ∪ θ0;   r = r + 1;
    Update Δ_{r×r};   /* Equation 14 */
    f_i = r   (i = m + 1, ..., m0);   m = m + m0;
  end if
end while
return C = {m, r, S, Θ, F};
```

At each step, the algorithm first pops an entry $\{\boldsymbol{\theta}_0, m_0, \mathcal{S}_0\}$ from the stack $\mathcal{Q}$. It then tries to refine the current regime $\boldsymbol{\theta}_0$, that is, it finds new candidate regime pair $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$, and their segment sets $\{\mathcal{S}_1, \mathcal{S}_2\}$ for a given segment set $\mathcal{S}_0$. If the coding cost of the new candidate regimes is less than the cost of the current regime, (i.e., the candidate regime pair wins), the algorithm pushes the candidate pair into the stack $\mathcal{Q}$. Otherwise, it leaves the regime out of the stack, and report $\{\boldsymbol{\theta}_0, m_0, \mathcal{S}_0\}$ as the output. This process is repeated until the holding stack is empty.



(a) AUTOPLAIT (**no** user defined parameters)

(b) DynaMMo (**need "user-defined"** $m = 2, 4, 8$)

(c-1) pHMM (**need "user-defined"**: $\epsilon_r = 0.1, \epsilon_c = 0.8$)

(c-2) pHMM (**need "user-defined"**: $\epsilon_r = 1.0, \epsilon_c = 0.8$)

(c-3) pHMM (**need "user-defined"**: $\epsilon_r = 0.1, \epsilon_c = 4.0$)

Figure 6: AUTOPLAIT is fully automatic: it finds the right cuts automatically: (a) result of AUTOPLAIT, (b) three sets of results $(r, m = 2, 4, 8)$ **for DynaMMo, (c) three sets of results** $(\epsilon_r = 0.1, 1.0, \epsilon_c = 0.8, 4.0)$ **for pHMM. Note that AUTOPLAIT finds the correct count and location of the cuts, while its competitors are very sensitive to user-defined parameters.**

5.4.2 *Theoretical analysis*

LEMMA 3. *The computation time of* AUTOPLAIT *is linear on the duration of the bundle,* n.

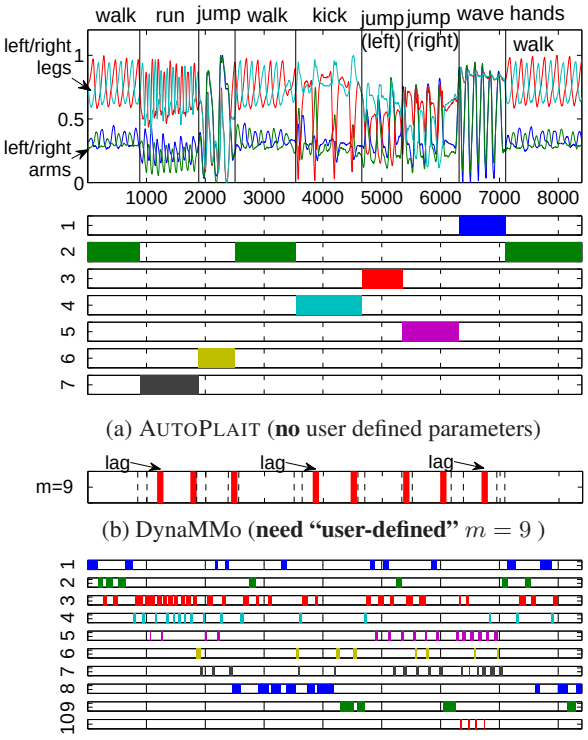
PROOF. For each iteration, CutPointSearch and RegimeSplit require $O(ndk^2)$ time to compute the coding cost and estimate the model parameters, where d is the number of dimensions, and k is the maximum number of hidden states in the regimes $\{\boldsymbol{\theta}_i\}_{i=1}^{r}$, (i.e., $k = \max\{k_1, k_2, \ldots, k_r\}$), Thus, the time complexity is $O(\#iter \cdot ndk^2)$. Note that the number of iterations $\#iter$, the number of hidden states k and dimensions d are small constant values that are negligible. Thus, the complexity is $O(n)$. □

6. EXPERIMENTS

In this section we demonstrate the effectiveness of AUTOPLAIT on real datasets. To ensure the repeatability of our results, we used publicly available datasets. These experiments were designed to answer the following questions:

Q1 *Sense-making*: Can our method help us understand the given input bundle?

Q2 *Accuracy*: How well does our method find cut-points and regimes?



(a) AUTOPLAIT (**no** user defined parameters)

(b) DynaMMo (**need "user-defined"** $m = 9$)

(c) pHMM (**need "user-defined"**: $\epsilon_r = 0.1, \epsilon_c = 0.8$)

Figure 7: AUTOPLAIT can detect un-repeated regimes: (a) It captures all distinct motions (walking, running, etc), while (b) DynaMMo was not completely successful in finding optimal cut points, (c) pHMM cannot find high-level motion patterns.

Q3 *Scalability*: How does our method scale in terms of computational time?

Our experiments were conducted on an Intel Core 2 Duo 3.50GHz with 32GB of memory, running Linux. We performed our experiments on the following real datasets. We normalized the values of each dataset so that they had the same mean and variance (i.e., z-normalization).

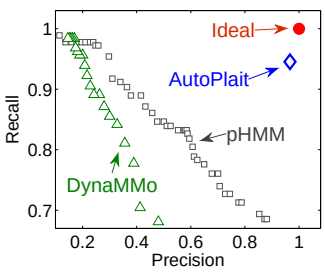
- *MoCap*: The dataset was obtained from the CMU motion capture database[6]. In this dataset, every motion is represented as a sequence of hundreds of frames. It consists of sequences of 64-dimensional vectors, and we chose four dimensions (left-right legs and arms).
- *WebClick*: This consists of the web-click records of 2,582,252 anonymous users, obtained over one month, (from 1st to 30th April 2007). It contains one billion records with 67 GB of storage, each of which has three attributes: URL, user and the time stamp of the click.
- *GoogleTrend*: This dataset consists of the volume of searches for various queries (i.e., words) on Google[7]. Each query represents the search volumes related to keywords over time (over nine years, on a weekly basis).

6.1 Sense-making

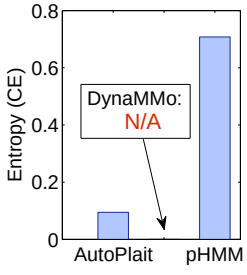
We carried out experiments on real motion capture datasets to demonstrate the effectiveness of AUTOPLAIT in finding optimal segments and regimes. We compared our method with scalable and state-of-the-art methods, namely, DynaMMo [18] and pHMM [34].

[6]http://mocap.cs.cmu.edu/

[7]http://www.google.com/trends/



(a) Precision and recall (higher is better)



(b) CE score (lower is better)

Figure 8: Accuracy of AUTOPLAIT: it consistently outperforms two alternative methods with respect to precision, recall and entropy. (a) Segmentation accuracy: it achieves 97% for precision and 95% for recall. (b) Clustering accuracy: it shows the conditional entropy (CE) of the confusion matrix between true labels and outputs. Note that DynaMMo cannot find clusters.

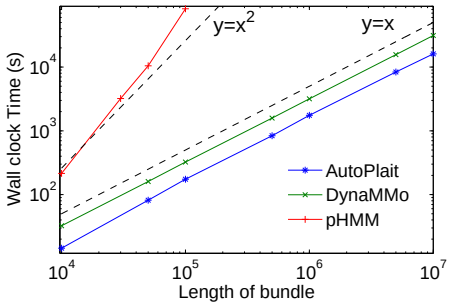


Figure 9: AUTOPLAIT scales linearly: wall clock time vs. datasize n. Our method and DynaMMo are linear (i.e., slope $= 1$ in log-log scale), while pHMM is at least quadratic (i.e., slope ≈ 2). AUTOPLAIT is 472 times faster than pHMM at $n = 10^5$.

One of our results was already presented in section 1 (see Figure 1). Figure 6 shows the results for another "chicken dance". Similar to the previous result, AUTOPLAIT successfully detected $m = 8$ segments, and $r = 4$ different steps of the dance. Figure 6 (b) shows the result obtained with DynaMMo, which requires a user-defined parameter: the number of segments m. We tried $m = 2, 4, 8$. In Figure 6 (b), the vertical dotted lines are the ground truth of the segmentation. DynaMMo was not completely successful in detecting the correct cut points, even though we provided the correct number of segments $m = 8$. Also note that DynaMMo is not capable of finding clusters/regimes. Figure 6 (c1)-(c3) shows the result obtained with pHMM. It required two parameters, ϵ_r, ϵ_c, which corresponded to the thresholds of the fitting errors. For example, in Figure 6 (c1), pHMM finds 36 segments, then each segment is assigned to one of 7 clusters. Since pHMM captures only line-like patterns, it cannot recognize high-level/long-term patterns. Also, as shown in Figure 6 (c1)-(c3), it is very sensitive to the user-defined parameters.

Figure 7 shows a more complicated case. This motion consists of multiple distinct motions, such as walking, jumping and kicking. Every motion appears only once, except for the walking motion. One of the strong points of AUTOPLAIT is that, unlike the typical clustering method, it can detect "un-repeated" trends. Actually, in Figure 7 (a), AUTOPLAIT successfully captures unrepeated motions such as running and jumping, as well as frequently-appearing motions (i.e., walking). Figure 7 (b), (c) show the results obtained with DynaMMo and pHMM. Similar to the results shown in Figure 6, they failed to capture distinct high-level patterns.

6.2 Accuracy

In this subsection, we discuss the quality of AUTOPLAIT in terms of segmentation and clustering accuracy.

Segmentation accuracy. We compared the segmentation accuracy of AUTOPLAIT and two competitors with respect to the precision/recall framework. Figure 8 (a) shows the precision and recall scores for the three methods on *MoCap* dataset. We used a set of 20 bundles, containing about $m = 20 \times 10$ segments. In total, bundle consists of approximately $n = 20 \times 10,000$ motion frames. Note that precision is defined as the ratio of reported correct cuts versus the total number of reported cuts. Recall is defined as the ratio of reported correct cuts versus the total number of correct cuts. The closer precision and recall are to 1, the more accurate the method is. In Figure 8 (a), AUTOPLAIT is described as a point, since our method does not need any parameters. Our method is very close to the ideal point (more than 95%), that is, there are very few false alarms and false dismissals. As mentioned in Figure 6, DynaMMo requires user-defined numbers m. We varied the number of segments $m = 2, 4, 6, \ldots, 30$. For pHMM, the accuracy threshold ϵ_r varied from 0.1 to 10.0. The result implies that there is a trade-off between precision and recall with both DynaMMo and pHMM.

Clustering accuracy. Next, we show how accurately AUTOPLAIT can find regimes. Since we know the true labels of each motion, we evaluate our method in terms of a clustering problem. Specifically, we adopt a standard measure of conditional entropy (CE) from the confusion matrix (CM) of prediction regime labels against true regime labels to evaluate the clustering quality. The CE score shows the difference between two clusters using the following equation: $CE = -\sum_{i,j} \frac{CM_{ij}}{\sum_{ij} CM_{ij}} \log \frac{CM_{ij}}{\sum_j CM_{ij}}$. Note that the ideal confusion matrix will be diagonal, in which case $CE = 0$. Figure 8 (b) shows the average CE scores of AUTOPLAIT and pHMM. Our method identified almost all regimes correctly, while pHMM failed to find groups. Note that DynaMMo is not capable of finding clusters.

6.3 Scalability

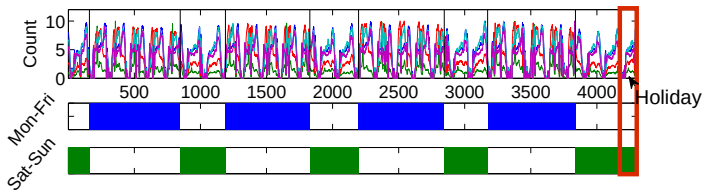
We performed experiments to evaluate the efficiency and to verify the complexity of AUTOPLAIT, which we discussed in section 5. Figure 9 compares AUTOPLAIT and the two comparison methods in terms of computation time for varying durations n. The plots were generated using *MoCap*. We used $k = 4$ hidden variables for DynaMMo, and $\epsilon_r = 0.1, \epsilon_c = 0.8$ for pHMM. AUTOPLAIT and DynaMMo are linear with respect to data size (i.e., $slope = 1.0$ in log-log scale). Unlike AUTOPLAIT and DynaMMo, which require $O(n)$, pHMM needs $O(n^2)$, to find optimal segments (i.e., $slope \approx 2.0$ in the figure). In fact, AUTOPLAIT is up to 472 times faster than pHMM at $n = 100,000$.

7. AUTOPLAIT - AT WORK

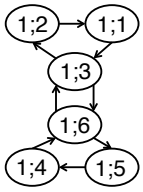
Our proposed method is capable of various applications. Here, we provide some examples of the usefulness of our approach.

7.1 Model analysis

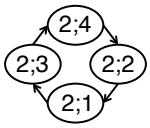
We demonstrate how effectively AUTOPLAIT can learn regimes using *WebClick* data. Specifically, Figure 10 (a) shows the access count of five major URLs (blog, news, etc.) every 10 minutes for one month. AUTOPLAIT detects two regimes, weekday and weekend. There is one anomaly point at the end of the month, and this is because of a national holiday (see the red rectangle in the figure). Figure 10 (b)-(e) provide the details for two estimated models



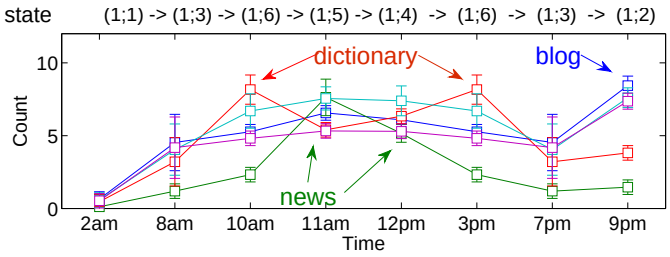
(a) One month pattern of five URLs (blog, news, dictionary, Q&A, and mail)



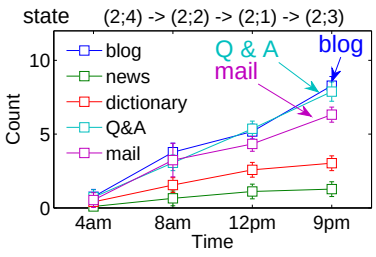
(b) State transition $\mathbf{A}_1$ (weekday, $k_1 = 6$)



(c) State transition $\mathbf{A}_2$ (weekend, $k_2 = 4$)



(d) Output probability $\mathbf{B}_1$ (weekday regime)



(e) Output probability $\mathbf{B}_2$ (weekend regime)

Figure 10: Sense-making - AUTOPLAIT results match intuition: (a) five time-sequences (blog, news, dictionary, etc.) over a month, every 10 minutes on *WebClick* data (top), and the result of AUTOPLAIT (bottom). It consists of two regimes (the blue regime, which perfectly maps weekdays, and the green regime, mapping weekends (and holidays - see last day); (b, d) details of the weekday regime: state $(1;1)$ is low volume (and it occurs around 2am - sleeping times), state $(1;6)$ accrues at 10am and 3pm - students use the online-dictionary; (c, e) details of the weekend regime: all states include weekend activities (blog, mail, Q&A) in varying magnitudes.

that describe the weekday and weekend regimes. More specifically, figure (b), (c) describe the Markov chains of the models (i.e., transition of hidden states), and (d), (e) show the output probability of five URLs, in each time range. For example, neither regime shows much activity during sleeping time, then the access count increases toward a peak at 9pm. Here we report some observations regarding these models.

- *Working hard every weekday:* For the weekday regime, states $(1;5)$, $(1;6)$ appear at 10am-3pm. From this observation, we can recognize that (1) most users visit the news site during their lunch break; (2) college students (or business professionals) frequently use the online-dictionary in the daytime.
- *No more work on weekends:* Most users visit social media sites on the weekend for non-business purposes. By contrast, news and dictionary sites are accessed less often on weekends. We also observed that on weekdays users visit blog and mail sites at night.

7.2 Event discovery

Since AUTOPLAIT has the ability to detect unknown patterns and regimes without any user interaction, the most natural and important application would be automatic event detection. Here, we introduce some of our discoveries as regards *GoogleTrend* data.

Anomaly detection. Figure 11 (a) shows a $d = 4$ dimensional bundle consisting of four flu-related keywords (e.g., "flu fever", "flu symptom") and covering nine years. There is a clear yearly periodicity. Starting in October, it slowly increases toward its peak point (in February), and then decays over time. The only exception is in 2009, (regime #1, blue box in the figure). In regime #1, there are two spikes in April and October, since that was when the swine flu pandemic spread around the world.

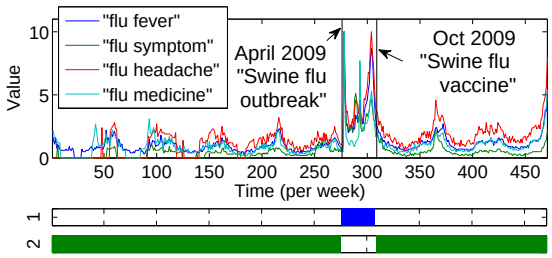
Turning-point detection. Figure 11 (b) shows another observation. The figure shows a bundle related to a seasonal-sweets topic (e.g., "ice cream", "hot cocoa"). Each keyword has a yearly cycle with a different phase; there are peaks in July for "ice cream" and "milk shakes", while there are peaks in December for "hot cocoa" and "gingerbread". However, the trend suddenly changed in Dec 2010. This was caused by the release of the android OS, called "Gingerbread", "Ice Cream Sandwich".

Trend discovery. Figure 11 (c) shows co-evolving sequences, related to the game industry, (e.g., "xbox", "wii"). There is a yearly periodicity, with a peak every December. Recently, the video-game industry has been facing increasing competition. AUTOPLAIT detects $r = 3$ regimes in the game console war over the last nine years; (1) Xbox and Playstation were sold worldwide, and then (2) Wii outsold its competitors after its 2006 launch. However, (3) Wii and other console sales fell considerably in 2010, probably because of the increased popularity of mobile and social games.

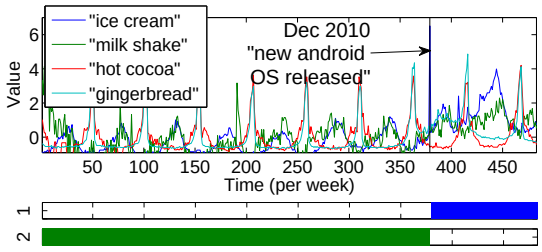
8. CONCLUSIONS

We focused on the problem of the automatic mining of co-evolving time sequences, with the aim of automatically determining the number and type of "regimes" (= patterns), as well as the transition (= discontinuity) points. Our proposed AUTOPLAIT indeed exhibits all the desirable properties we listed in the introduction:

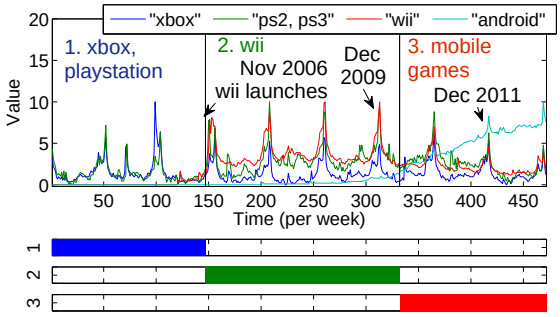
- It is **effective**: our experiments with diverse datasets (mocap data, web-hit counts) show that AUTOPLAIT discovers regimes that agree with human intuition, and correctly spots the transition points. More importantly, our algorithm guarantees to find the optimal location of each regime transition.
- It is **sense-making**: thanks to our modeling framework, it can help with sense-making. It provides a high-level summary (e.g., "beaks" and "wings" in Figure 1) as opposes to a large number of low-level states that a human cannot interpret easily.
- It is **fully automatic**: AUTOPLAIT needs no training set, no domain expertise, and no hint regarding the number of regimes or transition points. Thanks to our novel coding scheme, it determines all of the above automatically.
- It is **scalable**: both the computation time and the memory requirements are linear for the duration n of the sequences.



(a) Flu-related topics (regimes $r = 2$)



(b) Seasonal sweets topics (regimes $r = 2$)



(c) Game-related topics (regimes $r = 3$)

Figure 11: Sense-making: AUTOPLAIT automatically spots meaningful discontinuities on *GoogleTrend* data: (a) flu-related keywords for nine years: it detects one unusual pattern in 2009 (i.e., swine flu); (b) sweets-related keywords: it finds that the trend suddenly changed in 2010, because of the release of the android OS; (c) game-related keywords: it discovers three phases of "game console war" (e.g., regime #1: Xbox and PlayStation, #2: Wii's golden age, #3: mobile games appeared).

Acknowledgement. This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research Number 24500138, Grant-in-Aid for JSPS Fellows Number 25·7946. This material is based upon work supported by the National Science Foundation under Grants No. IIS-1247489 CNS-1314632 and by the U.S. Army Research Office (ARO) and DARPA under Contract Number W911NF-11-C-0088. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

9. REFERENCES

[1] D. Agarwal, B.-C. Chen, and P. Elango. Spatio-temporal models for estimating click-through rate. In *WWW*, pages 21–30, 2009.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[3] C. Böhm, C. Faloutsos, and C. Plant. Outlier-robust clustering using independent components. In *SIGMOD*, pages 185–198, 2008.

[4] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, NJ, 3rd edition, 1994.

[5] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88, 2004.

[6] H. Chen, W.-S. Ku, H. Wang, and M.-T. Sun. Leveraging spatio-temporal redundancy for rfid data cleansing. In *SIGMOD*, pages 51–62, 2010.

[7] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[8] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1999.

[9] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[10] E. Fox, E. Sudderth, M. Jordan, and A. Willsky. Bayesian Nonparametric Methods for Learning Markov Switching Processes. *Signal Processing Magazine, IEEE*, 27(6):43–54, 2010.

[11] E. B. Fox, E. B. Sudderth, M. I. Jordan, and A. S. Willsky. Sharing features among dynamical systems with beta processes. In *NIPS*, pages 549–557, 2009.

[12] Y. Fujiwara, Y. Sakurai, and M. Yamamuro. Spiral: efficient and exact model identification for hidden markov models. In *KDD*, pages 247–255, 2008.

[13] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD*, pages 11–22, 2004.

[14] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[15] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD Conference*, pages 593–604, 2007.

[16] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Access methods for markovian streams. In *ICDE*, pages 246–257, 2009.

[17] L. Li, C.-J. M. Liang, J. Liu, S. Nath, A. Terzis, and C. Faloutsos. Thermocast: A cyber-physical forecasting model for data centers. In *KDD*, 2011.

[18] L. Li, J. McCann, N. Pollard, and C. Faloutsos. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*, 2009.

[19] L. Li, B. A. Prakash, and C. Faloutsos. Parsimonious linear fingerprinting for time series. *PVLDB*, 3(1):385–396, 2010.

[20] Y. Matsubara, Y. Sakurai, C. Faloutsos, T. Iwata, and M. Yoshikawa. Fast mining and forecasting of complex time-stamped events. In *KDD*, pages 271–279, 2012.

[21] Y. Matsubara, Y. Sakurai, B. A. Prakash, L. Li, and C. Faloutsos. Rise and fall patterns of information diffusion: model and implications. In *KDD*, pages 6–14, 2012.

[22] A. Mueen and E. J. Keogh. Online discovery and maintenance of time series motifs. In *KDD*, pages 1089–1098, 2010.

[23] R. T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Trans. Knowl. Data Eng.*, 14(5):1003–1016, 2002.

[24] T. Palpanas, M. Vlachos, E. Keogh, and D. Gunopulos. Streaming time series summarization using user-defined amnesic functions. *IEEE Transactions on Knowledge and Data Engineering*, 20(7):992–1006, 2008.

[25] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *Proceedings of VLDB*, pages 697–708, Trondheim, Norway, August-September 2005.

[26] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, pages 262–270, 2012.

[27] J. Rissanen. A Universal Prior for Integers and Estimation by Minimum Description Length. *Ann. of Statist.*, 11(2):416–431, 1983.

[28] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream monitoring under the time warping distance. In *ICDE*, pages 1046–1055, 2007.

[29] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.

[30] V. Shnayder, B.-r. Chen, K. Lorincz, T. R. F. F. Jones, and M. Welsh. Sensor networks for medical care. In *SenSys*, pages 314–314, 2005.

[31] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of ACM SIGMOD*, pages 611–622, 2004.

[32] N. Tatti and J. Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *KDD*, pages 462–470, 2012.

[33] M. Toyoda, Y. Sakurai, and Y. Ishikawa. Pattern discovery in data streams under the time warping distance. *VLDB J.*, 22(3):295–318, 2013.

[34] P. Wang, H. Wang, and W. Wang. Finding semantics in time series. In *SIGMOD Conference*, pages 385–396, 2011.

[35] J. G. Wilpon, L. R. Rabiner, C. H. Lee, and E. R. Goldman. Automatic recognition of keywords in unconstrained speech using hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(11):1870–1878, 1990.

[36] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114. ACM, 1996.