Model Based Data Reduction for Event Sequences

Emil P. Stefanov
Purdue University
Department of Computer Science
emil@purdue.edu

Mikhail J. Atallah[*]
Purdue University
Department of Computer Science
mja@cs.purdue.edu

Wojciech Szpankowski[†]
Purdue University
Department of Computer Science
spa@cs.purdue.edu

ABSTRACT

Whether they are the audit trails of the events in a computer system, of traffic in a network, of actions by individuals, or records of financial transactions monitored for internal compliance by a financial corporation (or monitored externally by the SEC or FBI), records of events tend to be massive. In this haystack of events can lie buried valuable information whose extraction would be easier if the event record could be reduced. This paper is a step in this direction, in that it gives an algorithm that takes as input a sequence of events that were generated by k Markov models, and separates it into k sequences each of which corresponds to the sub-sequence generated by one of the k models. The input to the algorithm does not include the state space or transition matrix of any of the k models, nor does it include the parameters that were used to mix their respective outputs and produce the merged sequence; thus making our algorithm universal within the class of Markov sources. To identify statistically significant events, we develop an approximate statistical analysis. Finally, we report experimental results demonstrating that our algorithm is both fast and accurate. Our techniques work also remarkably well for higher order Markovian sources. Unlike previous work in this area, we present an algorithm that does not assume that the symbols generated by k different Markov models are disjoint; to our knowledge we are the first to handle this substantially more difficult case of k models with overlapping symbols; Our results are backed by both theoretical analyses and experimental data. Previous work had no experimental results.

[*]Portions of this work were supported by Grants IIS-0325345 and CNS-0627488 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

[†]The work of this author was supported by the NSF Grant CCR-0208709, NIH grant R01 GM068959-01, and NSA grant 07G-044.

Keywords

event sequence, data reduction, Markov model, probabilistic analysis

1. INTRODUCTION

The increasing use of proxies and of network address translation has made it hard to link Internet actions to their originating source. Sophisticated Web-bots deliberately use randomly varying sets of proxies to access target sites (so that 10 simultaneous requests appear to come from 10 different random sources), often ignoring the site's wishes to keep them out (as expressed in the robots.txt file) and even trying to masquerade as human users and escape the bot-detection mechanisms that have been proposed and deployed (see [14] for some of these). The purpose of this can range from collecting information about a competing corporation, about a sensitive topic, or simply to foil intrusion-detection and misuse-detection systems. As a result of this, event-logging mechanisms are unable to distinguish the true origin of an event. Moreover, event-logging mechanisms often do not know ahead of time which subset of events will be of future interest (e.g., from a forensic point of view), and have no choice but to err on the conservative side and record more than what is needed.

Even when the event-logging system knows that only some types of events may ultimately be of interest, it still typically records the other events because they may pertain to the events of interest (this could take the form of correlation or causality). However, an audit log that contains an over-abundance of data makes the task of processing it more difficult. One would expect data reduction to be possible if one has a detailed model of the processes that generate all the recorded events, and of the process that causes them to arrive and be recorded in a particular order. But such a model is typically not available a priori, and must ultimately be derived from the only observable: The recorded event stream. Other potential applications are in biology (e.g., finding coding vs noncoding regions) and intrusion detection (i.e., identifying improper use of a system).

This paper shows that data reduction is possible even without knowing much about the models that generated the events, or about the observation and logging mechanism that recorded them: All that is known is that (i) each is individually Markovian; (ii) they are independent and their outputs are randomly inter-mixed; and (iii) they generate event types that may overlap or not. Take the example of the set of event types where each type corresponds to visiting a particular web page (or, at a coarser level of granularity, a

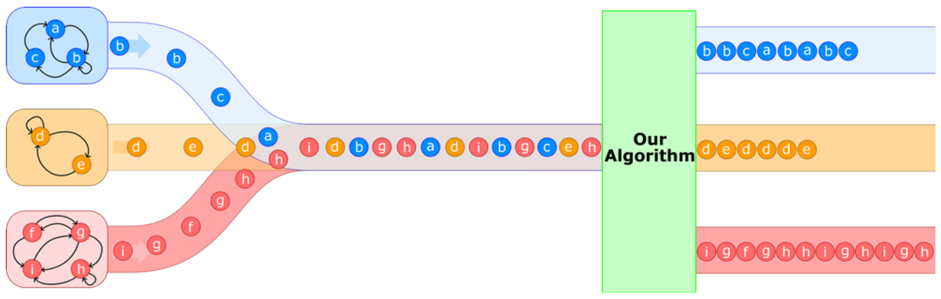

Figure 1: Illustration to the Mixing Sources Mode: One observes a mixture of symbols originating from several different Markov sources and separates the symbols according to the model that generated them.

certain category of web page). They do tend to be Markovian (where we go next tends to depend on where we are now), independent (what Alice does tends to be unrelated to what Bob does). The "disjoint" assumption is sometimes true, often approximately true; our scheme has been tested and found to work for the case of over-lapping generation of the same event types by different Markov sources.

The basic issue considered in this paper is the following: Can the inter-mixed outputs of k distinct Markov models be separated without knowing anything about each model (either its state space or transition matrix) or about the way their respective outputs were inter-mixed to produce the observable sequence? We give an efficient algorithm for this problem, and we experimentally demonstrate its effectiveness both for synthetic data and for test-case data available on the Internet.

One may view our model as a special case of the hidden Markov model [6] over the cross-product space with a projection of this product space to its components. However, this observation does not translate into a tractable computational solution. A related problem was discussed in [8] where switching sources were studied (cf. [3]), but the work most related to ours is [4] where several intractability results were proved together with some algorithms for separating Markov sources for independent and dependent switches when symbols from different sources are distinct or the mixture is of two identical Markov chains. There are no experimental results presented in [4]. Lemma 1 of [4] presents some conditions for probability estimates but these conditions can hardly be verified on real data. In this paper we present simple universal algorithms for Markov sources (both with disjoint and overlapping alphabets), together with an approximate statistical analysis that works remarkably well in practice.

2. MODEL

For $i \in \{1, \ldots, k\}$, let $M_i = (A_i, T_i)$ be Markovian sources over the alphabet A_i with probability transition matrix T_i. Assume that a distinct symbol is associated with each state of a source model M_i thus allowing us to use the term state and symbol interchangeable for any specific source model. We shall denote $\sigma_i = |A_i|$ and thus T_i is a $\sigma_i \times \sigma_i$ matrix,

Let $\hat{S}_i$ be a sequence generated by the source model M_i. Furthermore, let all A_i be pairwise disjoint. Let $\hat{S}$ be the sequence of events generated by repeatedly flipping a k-sided coin whose probability of side i is p_i: If the outcome of the coin flip is side i then the next symbol of $\hat{S}$ comes from

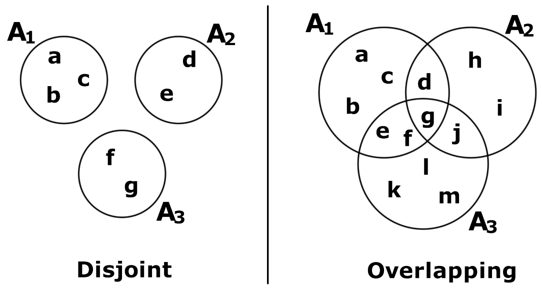

Figure 2: The alphabets of sample Disjoint and Overlapping Markov Merging Models.

$\hat{S}_i$, and that symbol of $\hat{S}_i$ is thereby "consumed" (hence the next time a coin flip produces side i, it is the next symbol of $\hat{S}_i$ that appears in $\hat{S}$). An equivalent definition of $\hat{S}$ is to say that $\hat{S}$ is obtained by running the textbook k-way merging algorithm on the $\hat{S}_i$'s, using the coin flip to determine the outcome of each k-way comparison done by the merging algorithm (a coin toss that turns up side i, places the comparand from $\hat{S}_i$ in the output sequence $\hat{S}$).

The input to the algorithm is a sequence S that is a contiguous portion of the above $\hat{S}$, i.e., a "window of observation" of width $|S| = n$. The output of the algorithm is a partition of the input S into $S_1, \ldots, S_k$ where S_i is the subsequence of S restricted to the events from A_i; hence the algorithm needs to determine all of the A_i's. The algorithm has a small probability of failing to produce such an output; we shall characterize precisely which inputs cause this.

We define a Markov Merging Model to be the model where the outputs of M_i are inter-mixed as specified earlier with respective probabilities $p_1, \ldots, p_k$. A Disjoint Markov Merging Model is a Markov Merging Model where $A_i \cap A_j = \emptyset$ $\forall i \neq j$. An Overlapping Markov Merging Model is a Markov Merging Model where $A_i \cap A_j \neq \emptyset$ for all $i \neq j$. These definitions will be used to distinguish between the different algorithms presented in this paper.

We henceforth use A to denote $\cup_{i=1}^{k} A_i$ and σ to denote $|A|$ (hence $\sigma = \sigma_1 + \ldots + \sigma_k$).

3. PRELIMINARIES

This section presents our analysis that will be needed to justify our algorithms presented in the next sections.

3.1 Definitions

Each Markov model M_i is time invariant (homogeneous) so we define the following stationary probabilities for S_i:

- $P_i(uv) = P(S_i[t] = u, S_i[t+1] = v)$,
- $P_i(v|u) = P(S_i[t] = v|S_i[t-1] = u)$.

The output sequence is stationary, hence we can define the following stationary probabilities for S:

- $P(uv) = P(S[t] = u, S[t+1] = v)$,
- $P(v|u) = P(S[t] = v|S[t-1] = u) = T_i[u,v]$

for any $u, v \in A$.

3.2 The Discriminator

The algorithms presented in this paper use a discriminator, denoted by $\hat{\delta}$, to help determining whether two symbols are generated by the same Markov source (i.e., for symbols u and v, does there exist i s.t. $u \in A_i$ and $v \in A_i$?).

It is extremely easy to compute $\hat{\delta}$ by counting the number of occurrences of pairs of symbols. We use the expression $\#uv$ to denote the number of occurrences of the string uv in S. Then

$$\hat{\delta}(u,v) = \#uv - \#vu.$$

Let $\delta(w)$ denote the number of w occurrences in S. Clearly (cf. [12, 13]),

$$P(w) = \lim_{n\to\infty} \frac{\delta_n(w)}{n}.$$

where for random S the above limit must be "in probability", and even stronger "almost sure" convergence sense. In fact, the rate of convergence of the above limit for Markov processes is exponential [12].

We now estimate $\hat{\delta}$ based on δ defined as

$$\delta = (n-1)\cdot(P(uv) - P(vu)).$$

Observe that

$$\begin{aligned}\lim_{n\to\infty} \frac{\hat{\delta}}{n-1} &= \frac{(n-1)P(uv) - (n-1)P(vu)}{n-1}\\ &= P(uv) - P(vu)\\ &= \delta,\end{aligned}$$

and then

$$\begin{aligned}&\lim_{n\to\infty} (\delta - \hat{\delta})\\ &= (n-1)(P(uv) - P(vu)) - ((n-1)P(uv) - (n-1)P(vu))\\ &= 0.\end{aligned}$$

Thus for large values of n (when S is long), we can treat $\hat{\delta}$ as an approximation to δ. We will later quantify how good is this approximation.

We now show that if for some i we have $u \in A_i$ and $v \in A_i$, then $\delta = 0$; therefore, u and v must belong to the same alphabet. In the case of disjoint sources, this implies that u and v are generated by the same source model.

Let us now concentrate on disjoint alphabets. We first calculate $P(uv)$, the probability of occurrence of the string uv at a certain position in S. Recall that $P(uv)$ is the same for each position in S thereby allowing us to use the single notation $P(uv)$ for any fixed position in S.

Observe that $P(uv)$ is the sum of the probabilities of two disjoint events:

1. The merging process has determined that at position x in S, the output of the source model M_i will be placed, and at position $x+1$, the next output of the same source model M_i will be placed. The probability that the merging process will pick the model i twice is ${p_i}^2$ and the probability of M_i outputting a u followed by a v is $P_i(uv) = P_i(u)P_i(v|u)$. Since all k models have to be accounted for, the overall probability of this event occurring is
$$\sum_{i=1}^{k} {p_i}^2 \cdot P_i(uv).$$
2. The merging process has determined that at position x in S, the output of the source model M_i will be placed, and at position $x+1$, the output of a different source model M_j $(i \neq j)$ will be placed. The probability that the merging process will pick model i then j is $p_i p_j$ and the probability of M_i outputting u and M_j outputting v is $P_i(u)P_j(v)$. Since all models where $i \neq j$ have to be accounted for, the overall probability of this event occurring is
$$\sum_{i\neq j} p_i p_j \cdot P_i(u)\cdot P_j(v).$$

These two events are disjoint and hence

$$P(uv) = \sum_{i=1}^{k} {p_i}^2 \cdot P_i(uv) + \sum_{i\neq j} p_i p_j \cdot P_i(u)\cdot P_j(v).$$

Similarly,

$$P(vu) = \sum_{i=1}^{k} {p_i}^2 \cdot P_i(vu) + \sum_{i\neq j} p_i p_j \cdot P_i(v)\cdot P_j(u).$$

Hence,

$$P(uv) - P(vu) = \sum_{i=1}^{k} {p_i}^2 \cdot (P_i(uv) - P_i(vu)).$$

and

$$\begin{aligned}\delta(u,v) &= (n-1)\cdot(P(uv) - P(vu)) && (1)\\ &= (n-1)\sum_{i=1}^{k} {p_i}^2 \cdot (P_i(uv) - P_i(vu)). && (2)\end{aligned}$$

The critical observation is that if u and v belong to disjoint alphabets, then u or v cannot occur in S_i and so the strings uv and vu do not occur in S_i. Therefore, $P_i(uv) - P_i(vu) = 0 - 0 = 0$. In other words, if $\forall i \in \{1,\ldots,k\}$, $u \notin A_i$ or $v \notin A_j$, then $P_i(uv) - P_i(vu) = 0 - 0 = 0$, and $\delta(u,v) = 0$.

3.3 Estimating the Discriminator

Our algorithms rely on the ability to estimate $\delta(u,v)$ and determine with high probability whether small $\hat{\delta}$ implies $\delta(u,v) = 0$. As argued, we shall use $\hat{\delta} = \#uv - \#vu$ as an estimator of $\delta(u,v)$. To simplify our presentation, we write $\hat{\delta}_n$ for $\hat{\delta}(u,v)$ when the sequence is of length n.

In the sequel, we present a simplified analysis of $\hat{\delta}_n$. Our goal is to assure that if $\hat{\delta}_n \leq \varepsilon$ for some $\varepsilon > 0$, then with high probability $\hat{\delta}_n = 0$. We accomplish it by appealing to the Chebyshev inequality

$$P(\hat{\delta}_n > \varepsilon) \leq \frac{Var[\hat{\delta}_n]}{\varepsilon^2}.$$

Thus, we must aim at evaluating variance $Var[\hat{\delta}_n]$.

We shall prove that

$$Var((n-1)\delta) = Var(\#uv - \#vu) \approx$$
$$n(P(uv)+P(vu)) - 4(n-1)(P(u)P(uv)+P(v)P(vu))$$

To derive this result we start out with a few definitions. Let $N_i(ab)$ be 1 when $S[i] = a$ and $S[i+1] = b$ and 0 otherwise (i.e., it is 1 iff string ab appears at position i in the overall sequence S). Let $D_i = N_i(uv) - N_i(vu)$. Our goal is to derive an approximation for the variance of $\hat{\delta}_n = \#uv - \#vu = \sum_{i=1}^{n} D_i$. Clearly,

$$Var(\hat{\delta}_n) = \sum_{i=1}^{n} Var(D_i) + \sum_{i \neq j} Cov(D_i, D_j).$$

Since

$$\begin{aligned} E[D_i] &= E[N_i(uv)] - E[N_i(vu)] \\ &= 1 * P(N_i(uv)=1) - 1 * P(N_i(vu)=1) \\ &= 0 \end{aligned}$$

and

$$\begin{aligned} E[{D_i}^2] &= E[(N_i(uv) - N_i(vu))^2] \\ &= E[(N_i(uv))^2 + N_i(uv)N_i(vu) + (N_i(vu))^2] \\ &= E[(N_i(uv))^2] + E[(N_i(vu))^2] \\ &= 1^2 P(N_i(uv)=1) + 1^2 P(N_i(vu)=1) \\ &= P(uv) + P(vu) \end{aligned}$$

we arrive at

$$Var(D_i) = E[{D_i}^2] - E[D_i]^2 = P(uv) + P(vu)$$
$$\sum_{i=1}^{n} Var(D_i) = n(P(uv) + P(vu)).$$

Now we approximate the second summation:

$$Cov(D_i, D_j) = E[(D_i - E[D_i])(D_j - E[D_j])].$$

We find

$$\begin{aligned} Cov(D_i, D_{i+1}) &= E[(D_i - E[D_i])(D_{i+1} - E[D_{i+1}])] \\ &= E[D_i D_{i+1}] \\ &= E[(N_i(uv) - N_i(vu))(N_{i+1}(uv) - N_{i+1}(vu)).] \end{aligned}$$

There are two disjoint events when this value is nonzero: (1) uv appears at position i in S with probability $P(uv)$ and u appears at position $i+2$ in S with probability $P(u)$, causing vu to appear at position $i+1$. (2) vu appears at position i in S with probability $P(vu)$ and v appears at position $i+2$ in S with probability $P(u)$, causing uv to appear at position $i+1$. Hence,

$$\begin{aligned} &Cov(D_i, D_{i+1}) \\ &= P(uv)P(u)(1-0)(0-1) + P(vu)P(v)(0-1)(1-0) \\ &= -P(u)P(uv) - P(v)P(vu) \end{aligned}$$

We will only consider the covariances $Cov(D_{i-1}, D_i)$ and $Cov(D_i, D_{i+1})$ for each i. In other words, we will assume that the effect of D_j on D_i, coming from the memory of the mixing process, is neglected in our analysis when $|i-j| > 1$ (cf. [13] how to compute it for Markov sources). Thus

$$\begin{aligned} \sum_{i \neq j} Cov(D_i, D_j) &\approx 2\sum_{i=2}^{n} Cov(D_{i-1}, D_i) + 2\sum_{i=1}^{n-1} Cov(D_i, D_{i+1}) \\ &= 4\sum_{i=1}^{n-1} (-P(u)P(uv) - P(v)P(vu)) \\ &= -4(n-1)(P(u)P(uv) + P(v)P(vu)). \end{aligned}$$

Finally, combining the two summations, we get:

$$Var(\hat{\delta}_n) = n(P(uv)+P(vu)) - 4(n-1)(P(u)P(uv)+P(v)P(vu))$$

as needed.

3.4 Using the Discriminator

We have shown that if u and v do not belong to the same source model alphabet, then $\delta(u,v) = 0$. Note that the converse is not necessarily true: if $\delta(u,v) = 0$, then u and v never belong to the same source model. This leads to the so called *indistinguishable cases* (cf. [4]).

We can pinpoint when the converse is not necessarily true. This happens when $\forall i \; P_i(uv) = P_i(vu)$. Such a Markov is called reversible, because the probability of encountering a uv in S is the same as the probability of encountering a vu for all pairs (u,v).

Based on the estimator $\hat{\delta}$, we will now define an Attraction Graph (call it G) which is used by the algorithm for overlapping sources. The vertices of the attraction graph consist of the elements of A (the union of all alphabets). If $\hat{\delta} > \varepsilon$, for some threshold $\varepsilon > 0$, we assume that $\hat{\delta}_n > 0$ with high probability and we establish an edge in the graph. (Notice that $\hat{\delta}$ acts as an estimator for $\hat{\delta}_n$.)

4. DISJOINT SOURCES ALGORITHMS

In this section we present our separation algorithms for known and unknown k.

4.1 Accurate Algorithm for Known k

In the previous section, we saw that $\hat{\delta} \approx 0$ is a good indicator that x and y belong to different models. However, $\hat{\delta} \not\approx 0$, which indicates that x and y are from the same model, is an even better indicator because it is **only** true when x and y are from the same model. This suggests using $\hat{\delta}$ as a measure of the likelihood of x and y being in the same model. Observe that large values of $\hat{\delta}$ may be interpreted that x and y are likely to be in the same model.

In summary, the discriminator works better for deciding whether x and y are generated the same model. Therefore, our algorithm takes action only for pairs x, y for which $\left|\hat{\delta}(x,y)\right|$ is large. The algorithm relies heavily on this observation, by grouping pairs x, y in descending order of $\left|\hat{\delta}(x,y)\right|$ over all pairs where $x \neq y$.

The algorithm is as follows:

1. Compute $\hat{\delta}(x,y)$ for all $x \neq y$.

2. Sort the pairs generated in step 1 in descending order of $\left|\hat{\delta}(x,y)\right|$.

3. Place each of the σ event types in a group of its own, then go through the sorted pairs (x,y) and, if x and y

are in different groups, then merge those two groups. Stop looking at such pairs (x, y) when exactly k groups remain.

4. Return the k groups and, along with each group, the subsequence of S whose symbols come from that group.

We now turn to the implementation and complexity of the above algorithm.

- The computation of $\delta(x, y)$ for all $x \neq y$ can be done in $O(n)$ time and $O(\sigma^2)$ space as follows: Sweep through the sequence and count the number of occurrences of each symbol x (i.e., $\#x$). This takes $O(n+\sigma)$ time and $O(\sigma)$ space. During the same sweep, for every length 2 substring xy of S, increment $\#(xy)$. This takes $O(n)$ time and $O(t)$ space where t is the number of distinct length 2 substrings of S, hence $t \leq \min\{n, \sigma^2\}$.
- Since there are $O(\sigma^2)$ values in T, sorting them can be done in $O(\sigma^2 \log \sigma)$ time.
- The repeated merging of groups can be done efficiently in a total of $O(\sigma^2 \cdot \alpha(t, \sigma))$ time, where α is the inverse Ackermann function, which is dominated by the $O(\sigma^2 \log \sigma)$ time of the previous step, and in $O(\sigma^2)$ space, by using the well known Union-Find data structure. [2]

Therefore, the overall complexity of the algorithm is $O(n + \sigma^2 \log \sigma)$ time and $O(\sigma^2)$ space. In any practical situation, the length n of the event sequence is much larger than the number σ of event types that we always have $n \geq \sigma^2 \log \sigma$. For this reason, we can say that the algorithm is $O(n)$ time.

4.2 Accurate Algorithm for Unknown k

[Carefully review the content below because it is brand new.]

If k, the number of source models, is not given, the algorithm in the previous section can be modified to only consider pairs (x, y) for which there is an edge in the Attraction Graph:

1. Compute the Attraction Graph, G, as defined in the preliminaries section.
2. Place each of the σ event types in a group of its own, then for each pair of symbols (x, y), if there exist an edge in G between x and y, then merge the group of x with the group of y. If x and y happen to already be in the same group, skip to the next pair.
3. Return the groups and, along with each group, the subsequence of S whose symbols come from that group. The number of resulting groups is the value of k.

The complexity analysis of the above algorithm is similar to that of the previous one:

- The computation of the Attraction Graph requires $\hat{\delta}(x, y)$ to be computed for each pair where $x \neq y$. Each $\psi(x, y)$ can be computed in constant time from $\hat{\delta}(x, y)$ using the closed form equation given in the preliminaries section. The attraction graph can then be built using an edge list representation by testing each value of $\psi(x, y)$. Hence computing the Attraction Graph takes $O(n)$ time and $O(\sigma^2)$ space.
- For the same reason as for the previous algorithm, the repeated merging of groups can be done in $O(\sigma^2 \cdot \alpha(t, \sigma))$ time and $O(\sigma)$ space.

Therefore the overall complexity of this algorithm is $O(n + \sigma^2 \cdot \alpha(t, \sigma))$ time and $O(\sigma^2)$ space. As mentioned earlier, in any practical situation, the length n of the event sequence is so much larger than the number σ of event types that we always have $n \geq \sigma^2 \cdot \alpha(t, \sigma)$. For this reason, we can say that the algorithm is $O(n)$ time.

4.3 A More Robust Algorithm

We propose here another algorithm for sources that may be approximated by Markovian models but with a little bit of error. This algorithm is more robust for such sources than the algorithm in the previous section, but less accurate for rigorously Markovian sources. We tested it on sequences that consisted of randomly intermixed text from two different natural languages (subsets of books in English and Spanish) and it very successfully separated each of them into its two constituent subsequences. We stress that the software that did this did not have any information about the structure or properties of English and Spanish. It could have just as well worked if the roles of English and Spanish had been played by two new unknown languages coming form outer space.

Let $\hat{\delta}$ be as previously defined:

$$\hat{\delta}(x, y) = \#(xy) - \#(yx)$$

The algorithm is then as follows:

1. Compute the pair $(x, y) \in A^2$ that minimizes $|\delta(x, y)|$. Set $U = \{x, y\}$. Then, repeat the following until $|U| = k$:
   (a) Compute $w \in A$ that minimizes
   $$\sum_{z \in U} |\delta(z, w)|$$
   (b) Include w in U.
2. U should now contain exactly one element from each of the models, call them $u_1, \ldots, u_k$. Let $V_1, \ldots, V_k$ be sets such that $V_i = \{u_i\}$.
3. In turn for each $y \in A - U$ (in random order), put y in one of the V_i's according to the following criterion. Compute the k scalar quantities $f(y, V_i) = \sum_{x \in V_i} |\delta(x, y)|$, $i = 1, \ldots, k$. The V_i in which y is inserted is the one that has maximum $f(y, V_i)$; note that it is the new (augmented) V_i that is used in the insertion of the next y considered.
4. Return the groups $V_1, \ldots, V_k$ and, along with each group, the subsequence of S whose symbols come from that group.

The complexity of the above is $O(n + \sigma^2)$ by an analysis similar to the one given in the previous section (minus the use of the UNION-FIND data structure, which is not needed in this algorithm).

The basis for the correctness of the above algorithm are the following observations: A very small value for $|\delta(x, y)|$ is an indicator that x and y are in different models, and this

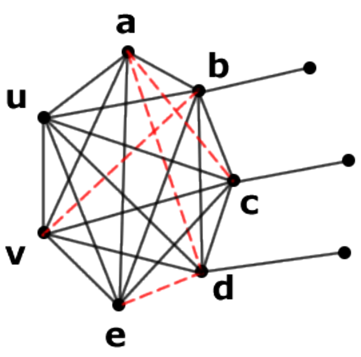

Figure 3: The merge cost of u and v is defined as the number of edges that must be added to the subgraph containing neighbors of u or v in order to make it a complete graph. In this example, the 4 dotted edges are not present in the neighborhood ($\{a, b, c, d, e\}$) of u and v so the merge cost $g(u, v)$ is 4.

is used to generate U with k different elements (one from each model). The second observation, that is the basis of building the V_i's, is that a large $f(y, V_i)$ is an indicator that y is in V_i, which is why the winning V_i (the one that will gobble up y) is the one having the largest $f(y, V_i)$.

5. OVERLAPPING SOURCES ALGORITHM

One of our main improvements of the work done in [4] is the formulation of an algorithm which infers mixtures of overlapping alphabets. In this section, we describe the algorithm. Its steps are as follows.

1. Compute the attraction graph G as described in the earlier sections.

2. For every pair of distinct vertices u, v in G compute the grouping cost $g(u, v)$ as follows.

   - Let $N(u, v)$ be the set of vertices of G that are adjacent to u or v or both, and let $G(u, v) = (V(u, v), E(u, v))$ be the subgraph of G that is induced by $N(u, v)$. Then $g(u, v)$ is the number of pairs x, y of vertices in $G(u, v)$ such that there is no edge (x, y) in G; equivalently, $g(u, v)$ is the number of edges that would have to be added to $G(u, v)$ in order to turn it into a complete graph. Therefore $g(u, v) = |V(u, v)|(|V(u, v)| - 1)/2 - |E(u, v)|$.

3. Sort the vertex pairs $\{u, v\}$ by increasing $g(u, v)$ values. Let L be this sorted list of pairs.

4. Initialize a set called *Unlabeled* to consist of all the vertices of V. As the algorithm proceeds, vertices will move out of *Unlabeled* to one of the following other sets (all of which are initially empty): $Pure_1, \ldots, Pure_k$, and *Unpure*. The intent is that $Pure_i$ will contain symbols that are in model i only (i.e., not in any of the other models), and *Unpure* will contain the symbols that are in more than one model. The way this is done is by processing the elements of L in left to right order and and creating the pure sets $Pure_i$. Advance along L is as follows.

   (a) Pick the next pair (u, v) from the list L subject to the condition that both u and v are unlabeled.

   (b) Put u and v in a new $Pure_i$, and move every vertex x of $N(u, v)$ out of *Unlabeled* and into one of two destinations: Move x to $Pure_i$ if x has no edge linking it to a vertex outside of $N(u, v)$, otherwise move x to *Unpure*.

5. Create a new set $Unclassified = Unlabeled \cup Unpure$.

6. At this point the algorithm has identified the pure subset, $Pure_i$, of each source model's alphabet, A_i. The elements that are in $Unclassified$ now need to be placed in the A_i's. Unlike the pure elements, the unclassified elements might belong to more than one source model alphabet. Use the pure sets as pivots to determine which alphabets to place the $Unclassified$ elements in the following way: for each w in $Unclassified$, place w in A_i if there is an edge between w and some element in $Pure_i$.

The complexity of this algorithm can be derived:

1. Computing the Attraction Graph takes $O(n)$ time and $O(\sigma^2)$ space as described in the previous section.

2. Since there are $O(\sigma^2)$ pairs, sorting them can be done in $O(\sigma^2 \log \sigma)$ time.

3. Iterating through L requires $O(\sigma^2)$ time since L is of size $O(\sigma^2)$.

The most amount of memory that can be used by the lists is $O(\sigma^2)$ in the case where each symbol appears in each alphabet.

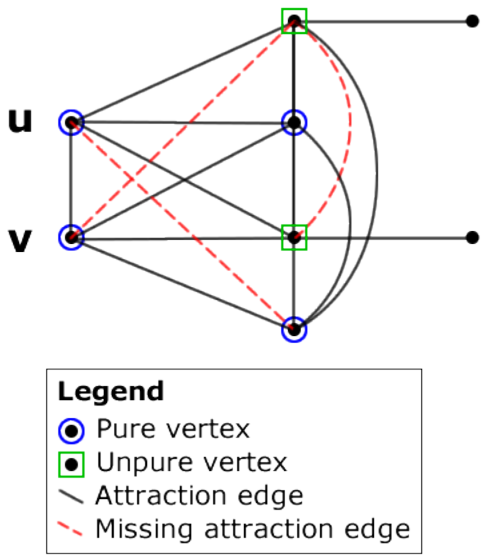

Figure 4: This Figure illustrates the approach of the algorithm for overlapping sources of identifying a pure set of symbols and the connected unsure symbols. The distinction between pure and unpure vertices is that pure vertices are only connected to the neighbors of u and v while the unpure vertices are also connected to at least one vertex that is not connected to one of u or v.

Therefore the total time complexity is $O(n + \sigma^2)$, and because of the previously mentioned insignificant size of σ^2 relative to n, the total time complexity can be expressed as $O(n)$. The total space complexity is $O(\sigma^2)$.

6. ASSOCIATING INSTANCES OF SYMBOLS TO ALPHABETS

After the alphabets are inferred using one of the algorithms described in the previous sections, an additional algorithm must be run to associate specific a occurrence of each symbol in the mixed sequence to its source alphabet. For non-overlapping alphabets, doing this is trivial because no two alphabets contain the same symbol and so there could only be one alphabet that generated the symbol. Overlapping alphabets, however, share some of the symbols. In this section, we propose a sliding window algorithm for associating occurrences of symbols to overlapping alphabets.

Assume that the overlapping alphabets $A_1, ..., A_k$ have been computed (e.g., using one of the algorithms described in the previous sections). The goal of the algorithm is to compute $\alpha[i]$ $(i = 1..n)$, the alphabet from which $S[i]$ came. So $\alpha[10] = 3$, indicates that $S[10]$ came from A_3. Run the following algorithm for each symbol $S[i]$.

1. Let w be the window radius. As is discussed later, our experiments show that $w = 5$ is a good choice in most situations.

2. Let $X = \{x_{-w}, ..., x_0, ..., x_w\}$ be the interval of length $2w + 1$ centered at position i in S (hence $x_0 = S[i]$).

3. For each possible mapping of symbol occurrences to alphabets $\dot{\alpha} : \{-w, ..., w\} \rightarrow \{1, ..., k\}$, let $\dot{\alpha}^*$ be the one with the highest probability of occurrence where the probability is computed as follows:

   (a) Based on $\dot{\alpha}[-w], ..., \dot{\alpha}[w]$, let y_j be the subsequence of X containing all of the symbol occurrences mapped to A_j.

   (b) The probability of occurrence of X (consisting of all y's) is:

$$\prod_{j=1}^{k} \prod_{l=1}^{|A_j|-1} P_j(y_j[l], y_j[l+1])$$

4. Assign $\alpha[i] = \dot{\alpha}^*[0]$.

There are k^{2w+1} mappings that need to be considered. Computing the probability of each mapping takes $O(w)$ time and $O(w)$ space. The set of mappings need to be computed for each occurrences of a symbol in the sequence so the total complexity is $O(n \cdot w \cdot k^{2w+1})$ time and $O(w)$ space. Although it is exponential, this complexity is acceptable because $w = 5$ (constant) and k, the number of alphabets is usually small.

7. EXPERIMENTAL RESULTS

In this section, we show the accuracy with which the algorithm infers the underlying alphabets of a sequence.

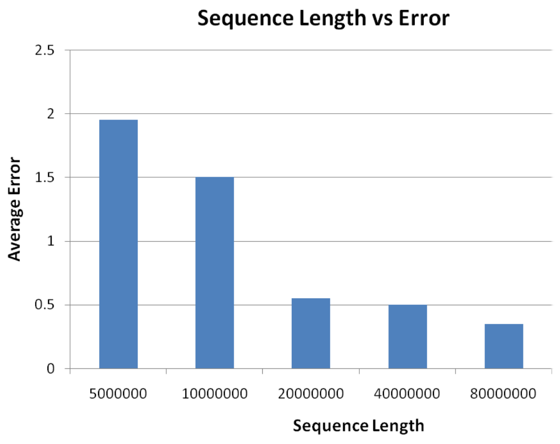

Figure 5: This Figure shows inference error as a function of sequence length for problems consisting of two 8-state Markov chains with 50% overlap.

7.1 Sequence Length

The longer the length of a sequence, the more accurate the algorithm is. We show this experimentally.

For each trial the following was done:

1. Create two random Markov chains with alphabets of 8 symbols each. The alphabets are picked such that exactly 4 of the symbols appear in both.

2. Using the mixture processes defined in this paper, create a sequence (lengths differ between trials).

3. Feed the sequence we just generated into the algorithm described in this paper in order to infer the alphabets.

4. Calculate the error between the alphabets that were used to generate the sequence and the alphabets inferred by the algorithm.

Error is calculated as follows. Let A^* and B^* be the correct alphabets and A and B be the alphabets inferred by the algorithm. Then the error is $\min(|A^* - A| + |A - A^*| + |B^* - B| + |B - B^*|, |A^* - B| + |B - A^*| + |B^* - A| + |A - B^*|)$.

The results can be seen in Figure 5.

7.2 Comparison to COLT04 Algorithm

The most closely related paper to our work is [4]. This section compares the algorithm in Section 4.1 to that of [4].

The algorithm described in [4] has a weakness in that it uses the "$\approx$" (approximately equal) operator for determining dependency relationships between symbols. The paper [4] does not explain how to determine if two values satisfy the $\approx$ condition. One possible way to test for approximate equality is to use a constant error margin ϵ and define $x \approx y$ as $|x - y| < \epsilon$. However, this does not work in practice because the algorithm is extremely sensitive to the value of ϵ and ϵ is different for every kind of problem.

Our data in Figure 5 shows that even a slight change in the ϵ parameter can cause the algorithm to fail every time. The task in the experiment was to separate a mixed sequence generated by two 10-state Markov chains. The mixed sequences varied in length between 2,000 and 20,000 symbols.

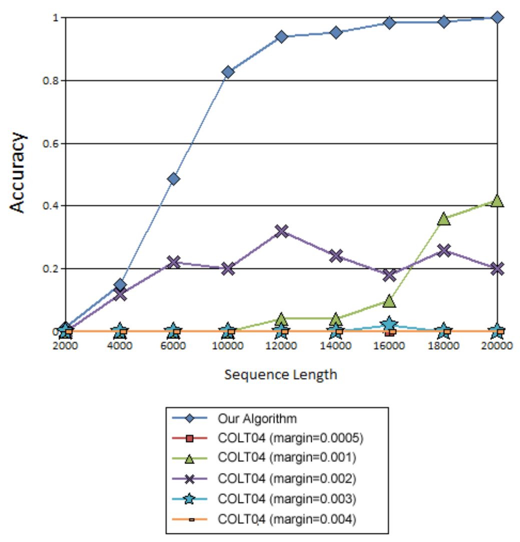

Figure 6: Accuracy Comparison to COLT04 Algorithm

For each set of parameters, accuracy was computed by calculating the percent correct solutions of each algorithm to 50 randomly generated problems. The same set of problems were given to each algorithm.

In Figure 5, five of the six series show the accuracy of the COLT04 ([4]) algorithm and the other series shows our algorithm. The COLT04 algorithm does best when ϵ (the error margin) is between 0.001 and 0.002. Values of ϵ outside that range result in almost 0% accuracy.

Our algorithm has two advantages over the COLT04 one: (1) It does not use $\approx$ (approximate equality) and therefore does not require an ϵ parameter. (2) It is much more accurate.

8. CONCLUSION

This paper presents several algorithms for separating randomly intermixed Markov chain sequences.

The accurate algorithm for known k in Section 4.1 is a major improvement of the algorithm in [4] because it eliminates the need to use an error margin to do approximate comparisons thus making the algorithm several times more efficient (as shown in Figure 6). This algorithm also requires a remarkably little amount of data to perform the separation.

The robust algorithm in Section 4.2 is able to better handle approximately Markovian data at the cost of accuracy. For purely Markovian data sources this algorithm does not produce as accurate results as the one in Section 4.1, but for Markovian data sources with noise, the accuracy is substantially better than both the algorithm in Section 4.1 and the one presented in [4].

Finally, in Section 5, we provide an algorithm which identifies the source alphabets even if they share some symbols. Although this algorithm requires more data than the ones in Section 4, it tackles a much harder variant of the separation problem and to the best of our knowledge, it is the first algorithm which accomplishes this.

Since the number of different symbols is usually much less than the total sequence length, each of the algorithms presented in this paper runs in linear time. The algorithms can also work with the sequence as a stream and require no more than two passes over the data. This means that the algorithms scale well for large datasets and could be used on databases which cannot be entirely stored in RAM.

9. REFERENCES

[1] Project gutenberg, 2007.

[2] A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1974.

[3] S. Arora and R. Kannan, Learning Mixturs of Arbitrary Guassians, *Proc. Symposium on Theory of Computing*, 247-257, 2001.

[4] T. Batu, S. Gupta and S. Kannan, Inferring Mixtures of Markov Chains, *Proc. COLT*, LNCS 3120, 186-199, 2004.

[5] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley and Sons, Inc., New York, 1991.

[6] Y. Ephraim and N. Merhav. Hidden markov processes. *IEEE Trans. Inform. Theory*, IT-48:1518–1569, June 2002.

[7] e. J.D. Ferguson. *Application of Hidden Markov Models to Text and Speech.* IDA-CRD, Princeton, NJ, 1980.

[8] Y. Freund and D. Ron, Learning to model sequences generated by switching distribitions, *Proc. Conference on Computational Learning Theory* (COLT'95), 41-50, New York, 1995.

[9] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77:257–286, February 1989.

[10] M. Régnier and W. Szpankowski. On the approximate pattern occurrences in a text. In *Proc. Compression and Complexity of SEQUENCE'97, IEEE Computer Soci ety*, pages 253–264, Positano, 1997.

[11] M. Régnier and W. Szpankowski. On pattern frequency occurrences in a markovian sequence. *Algorithmica*, 22:631–649, 1998.

[12] P. Shields. *The Ergodic Theory of Discrete Sample Paths.* American Mathematical Society, Providence, 1996.

[13] W. Szpankowski. *Average Case Analysis of Algorithms on Sequences.* John Wiley & Sons, New York, 2001.

[14] P. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. *Data Mining and Knowledge Discovery*, 6:9–35, 2002.