Modal Logic, Temporal Models and Neural Circuits: What Connects Them

Samantha Kleinberg[1] Marco Antoniotti[2] Naren Ramakrishnan[3] Bud Mishra[1]

[1] Courant Institute of Mathematical Sciences, New York University
[2] DISCo, Università Milano-Bicocca
[3] Department of Computer Science, Virginia Tech

CIMS Technical Report: TR2007-907

Abstract

"Processes" that temporally choreograph a large number of players appear pervasively in many situations, but pose particular challenges when one attempts to understand how these processes are orchestrated at an elemental level, namely, how may one learn what rules are used by the players to bring about this precise evolution of the process? What topologies are used by the network in codifying the rules of interaction? A particular example we study in this paper deals with the analysis of synthetic data simulating Microelectrode array(MEA) recordings. By computing statistically significant temporal patterns expressed in a propositional modal logic, we extract from it functional connectivity among neurons in an ensemble. For this purpose, we propose a novel algorithm founded upon many ideas from temporal logic, model inference and time-series data analysis, all aimed at the MEA inference problem. The approach, described here, has been validated by several examples in this domain with highly promising results.

1 Introduction

Wittgenstein, in his Brown Book [19], describes a process of learning as starting from an ontological basis: "Augustine, in describing his learning of language, says that he was taught to speak by learning the names of things." In this case, one simply learns the objects involved in a complex process, but may not understand what role each object plays in the process. Even when one lacks a complete and exhaustive list of all the possible objects involved in the process, they may still learn the process by inferring an abstract set of rules that explain how these objects are governed. For many purposes, such an incomplete operational learning of processes must suffice. Wittgenstein emphasizes this aspect of learning, when he states: "... Suppose a man describes a game of chess, without mentioning the existence and operations of the pawns. His description of the game as a natural phenomenon will be incomplete. On the other hand, we may say that he has completely described a simpler game." While many problems we face in our daily lives can be associated to such a learning of processes, we still lack a sufficiently rigorous framework, in which these problems can be articulated, analyzed and algorithmized. Examples in which this framework could be successfully applied are ubiquitous and numerous: understanding relations among webpages from the temporal structures embedded in click streams[18], deciphering interactions among financial sectors and markets from patterns in time-series stock-price data[10], delineating associations among genes from time-series data of mRNA abundances[4, 13], or inferring connections among neurons from multi-neuronal spike-train data. In this paper, we focus on the problem of dissection of multi-neuronal data, but refer the readers to other applications that have appeared in the literature.

2 Background

In experimental neuro-science, multi-neuronal recording provides a powerful electrophysiological technique to simultaneously record the activities of multiple neurons. These recordings provide time-series data describing the

synchronized activities of many neighboring neurons, which are assumed to be inter-connected in some complex, but unknown, manner, and thus affect the behavior of one another, determined by the neural connectivity. Thus, by analyzing recordings from working brains, i.e. behaving animals, extracellular multi-neuronal recording can be assumed to provide sufficient information to reconstruct part of the local circuits. However, these neural circuits can be fairly complex, and algorithmic techniques for automatically deducing the circuit topology remain underdeveloped. Furthermore, some of the neural activities recorded from the neighboring cells may be unrelated to the key functional circuits, and without suitable dimension-reduction steps, may lead to statistical model-overfitting, especially when the recorded multi-neuronal data is of limited duration.

In order to stimulate algorithmic development in this area, and evaluate the accuracy of various candidate algorithms, the 4th SIGKDD workshop on temporal data mining (2006) fostered a set of challenge problems from in silico simulations, using a variety of circuit topologies. The intent of this exercise was to evaluate how specific mining algorithms performed on these datasets, and thus, how likely it is that the algorithms would be effective in deciphering the correct neural circuits in real applications. This paper presents an approach that performed remarkably better than all other competing methods.

We will show that our approach analyzes multi-neuronal time-series data to find significant patterns in the temporal structure of the data, and that each such pattern could then be mapped to a particular neural-circuit topology. We avoid overfitting, by reducing the dimension of the data to a smaller size, by eliminating data from certain neurons if they did not correlate to a core set. We enhance the expressiveness of the temporal patterns discovered, by using formulae from temporal logic to describe not only elemental structures, such as when a particular firing preceded another or when several occurred simultaneously in response to an earlier event, but also more complex patterns that could be nested arbitrarily but in a syntactically structured way. This approach hence contains both the time complexity and loss of statistical significance, by only building patterns of sizes up to an upper-limit and composing the associated small sub-circuits to create the global circuit topology. Thus, the complete algorithm comprises three key steps: (1) Preprocessing to prune out unrelated neuronal data, (2) Discovery of temporal patterns (see Algorithm 1 for the pseudocode), and (3) Mapping the patterns into neural circuits. Of all three steps, the most critical and innovative step is the one dealing with step 2, which is discussed in details here; the other steps are built upon well-known techniques already available in the literature or application of obvious algorithmic techniques.

3 Related Work

The problem of discovering frequent episodes in sequential temporal data sets was proposed by Manilla et. al. in [15]. Their work looks to find groups of events occurring frequently in unison. Prior work by Agrawal and Srikant focused on discovering associations between events in sequences, for which they introduced the Apriori algorithm [2]. That work dealt primarily with finding events occurring either simultaneously or in a related way with certain rules determining this relatedness. This type of pattern finding has been used for mining associations, sequential patterns [3], and episodes [14], but lacks the ability to discover complex general rules.

An *event sequence* is defined as a series of (*label*, *time*) pairs, where the event labels are from some finite alphabet describing types of events. An *episode* is a group of events occurring together. This may be taken to mean either that they occur simultaneously (as in parallel episodes), or that they form a serial or even more complex pattern. We are primarily concerned with finding sequential patterns, which can then be combined using temporal logic to form more descriptive formulae. The essential idea in all of these methods is that in order for a pattern of length $n+1$ to be frequent, its subpattern of length n must also be frequent. By using this, the number of patterns to be tried is reduced in later stages of the algorithm, and the representation of the data may also be simplified as each iteration rules out more patterns [2]. At each step, the patterns of length $n+1$ being tested are called *candidates*, as these patterns may turn out to be frequent, but at this stage they may be treated as hypotheses. Another topic of research has been in studying candidate generation as well as in what order to test candidates [7, 5, 1], including algorithms for mining patterns without generating candidates [11].

In this context, one new idea introduced is that of *windows*. The main intuition is that we can set a time limit on how far apart two events may be while still being correlated. Thus, a window is a time interval $[t_s, t_e]$ defined by its start and end times. The window may start at a certain event, but note that the start time may be different than the start time of the event itself. In the case of neural connectivity, we can use this to look at neurons firing a minimum set period *away* from other neurons in the pattern. In Manilla et. al's work [14], windows are used

to contain entire episodes, ensuring that all events occur within a certain time of one another. The measure of support is then in how many of these short slices the episodes occur. This is a useful methodology for their work in correlating alarm data, where there are not normally long chains of alarms. For our applications to multi-neuronal datasets, we do not restrict an entire episode to be contained in a window in order to enable us to recover long fire chains. In other work [13], we use windows to find genes acting in concert during certain blocks of time and then use temporal logic and ontologies to stitch those blocks together.

Most of the related algorithms determine support for an episode based on how frequently it occurs. In this case, frequency may be the fraction of customers who purchased a group of products[3], the percentage of transactions containing an item[2] or the fraction of windows containing an episode[14]. One difference between our algorithm and the others noted here, is that by using the odds and likelihood, we take into account how often a pattern does not occur, so we will not automatically discard a pattern that appears few times, but always completely (all elements are represented in the same configuration). In this paper we seek to provide a statistical framework for the use of frequency, showing that we can apply the notions of odds and likelihood to the domain of temporal formulae.

Another recent algorithm that bears some similarities to ours is the one proposed by Moerchen in 2006 (See "Algorithms for Time Series Knowledge Mining"(TSKM) [16]). The premise of this algorithm is that patterns can be represented in a hierarchical manner, grouping together items (called *tones*) occurring more or less simultaneously, and then composing non-overlapping tones into larger patterns, called *chords* and *phrases*. The support for tones and chords is based upon the total duration during which they are observed.

One key difference between TSKM and our algorithm is that TSKM searches in a depth-first manner, extending one pattern as long as possible before trying another, while we follow a breadth-first search, namely, first finish testing all patterns of length n before examining any of length $n+1$. In this manner, we limit the number of patterns examined to a manageable size while still being exhaustive, and thus also focus only upon parsimoniously simpler patterns before yielding to rather complex ones. However, heuristically, in Moerchen's tests, the depth-first search improved speed, since often it was only required to obtain a rough idea of the class of patterns to be found. Thus, while avoiding such greedy heuristics, on the challenge dataset our algorithm was adequate to prune at each step and complete the analysis in a reasonable time frame.

In the TSKM algorithm, there is an emphasis on "almost" equal, allowing patterns that occur for more or less the same time period to be labeled "simultaneous." This approach is useful for their applications on sports medicine data, where they are looking for groups of muscles working together to find patterns in inline skating data. The algorithm we present here, however, is better suited to finding exact orderings between events, which is rather critical if one is looking for neural connections that form a series of neural firings. Our measure of support for a pattern has to be thus based on the frequency of the occurrence of an event, relative to the frequency of all events of that size. This measure was found well suited for the challenge application, where all neurons are assumed to fire for the same length of time.

A better intuition on how the two algorithms differ can be gained from the following example, illustrating how they handle Pattern 1 (fig 1.) TSKM algorithm would likely group A, B, and C together as a tone, and then find the relationship of that group to X; whereas we would first calculate the posterior for the pairs X–A, X–B, and X–C. Then, in the logical formulae building step, we would group the three patterns to form a pattern with the meaning "X until (A or B or C)". In the resulting tree, we show the scores for all of the resulting patterns, allowing the user to select one with a less than optimal score, but which may appear more interesting when viewed with outside information.

4 Approach

In this section, we briefly describe our algorithm starting with the underlying data structure. We also describe a few of the pragmatic decisions made to simplify our implementation.

4.1 Data Format and Preprocessing

The data input may have rows of one of the following two forms: a list of pairs, `eventLabel eventTime`, with one event label and time of its occurrence per row, or these may be assembled together into a list `eventLabel` $\texttt{eventTime}_1$ `...eventTime`$_N$. In our implementation, the first scheme allows us to use a more convenient method

Name	Example	Explanation
F (Sometime)	Fp	Eventually p will hold
G (Always)	Gp	p holds globally
R (Releases)	pRq	q holds and is released by p when it holds
U (Until)	pUq	p holds and continues until q holds
X (Next time)	Xp	In the next state, p holds

Table 1: Temporal Logic path operators

of calculating the occurrence frequency of temporal formulas. During this step, one could also use various noise-reduction procedures from signal processing, but it was found unnecessary. We believe that since the algorithm filters out noise quite quickly, it was not worth carrying out a lossy procedure that also occasionally removes true signal.

4.2 Computation Tree Logic

In order to describe and reason about the occurrence of events developing over time as in neural spike trains, or the regulation of biological processes in time-course gene expression data, we decided to encode these temporal patterns in a powerful logical framework. In the case of gene expression and neural data respectively, we would like to describe complex temporal relationships with such statements as "always G0 or M precedes G1, in a Cell Cycle", or "always Neuron A fires until Neuron B fires." In both cases, we may wish to describe when and if certain events occur by using well-formed formulae of propositional temporal logic, a logic consisting of propositions, Boolean connectives and modal operators. A particularly useful example of such a logic is Linear Temporal Logic (LTL) [17], which has found many applications in computer hardware design, communication protocols and engineering. LTL consists of logical operators $(\neg, \vee, \wedge, \rightarrow)$ and the modal operators in table 1.

LTL is limited in that with it we may only consider one possible path through time. With another propositional temporal logic that includes the concept of *branching* time, we can reason about paths as well as states, occurring through time, where nondeterminism can be modeled by splits of a path along multiple possibilities. This aspect of branching time temporal logics enables one to reason about events that *possibly* occur, but which are not necessarily inevitable. In this case, we can represent the idea that the occurrence of an event may be dependent on which path we take through time. A popular branching time temporal logic [6], Computation Tree Logic (CTL) [9], is widely used for this purpose. Unlike LTL, we can reason in CTL not only about one time line, but with multiple time lines representing multiple possible evolutions of a system. CTL, thus, allows statements such as "it is possible that eventually Neuron B will continue to fire until Neuron A does."

The components of CTL are a set of state operators and a set of path operators. These include path quantifiers A (All paths) and E (Exists a path), and all the linear time operators in LTL shown in table 1. The syntax differs from LTL in that there must be alternating *pairs* of path quantifiers and linear time operators. For example, one possible formula is $\mathsf{AG}p$, where p is a path formula and the formula means "For all paths, globally p will always be true." This additional stipulation follows from the fact that there may be multiple paths through time, and hence, we must specify whether formulae pertain to all paths or at least one path. One example of an invalid CTL formula would be $\mathsf{FG}p$, as it does not follow the requirement of a paired path quantifier and operator.

The whole of CTL may be defined inductively in terms of the following rules about states (S1-S3 below) and paths (P0), with AP being the set of atomic propositions:

- S1 A proposition p in AP is a state formula.
- S2 If p and q are state formulae, then so are $p \wedge q$ and$\neg p$.
- S3 If p is a path formula, then $\mathsf{E}p$ and $\mathsf{A}p$ are state formulae.
- P0 If p and q are state formula, then $\mathsf{X}p$ and $p\mathsf{U}q$ are path formulae.

CTL is a subset of CTL* which is defined by the above rules, with the substitution of P1-P3 below for P0:

- P1 Each state formula is also a path formula.
- P2 If p and q are path formulae, then so are $p \wedge q$ and $\neg p$.
- P3 If p and q is a path formulae, then so are $\mathsf{X}p$ and $p\mathsf{U}q$.

4.3 Model hypothesis generation and reconstruction

The second main feature of our approach is connecting the concepts from temporal logic and neural circuits to create a system that allows us to test the veracity of various hypotheses as well as to generate these hypotheses automatically. One method of achieving this is by accounting for the frequency with which all possible hypotheses or temporal formulae hold. It may not be that the formulae are always true throughout the universe of events we are looking at, but they may be true often enough that they should be deemed valid. Furthermore, for an entire formula or pattern of events to be frequent, each of its components must occur frequently as well. Since we are not making any initial predictions and are testing all possible temporal formulae[1], we can use this property to great advantage in reducing the complexity and size of the set of formulae being tested. Thus, in practice, we only examine the occurrences of all of the *components* of the hypotheses, starting from the smallest possible pattern and building upward. Along the way, infrequent components are identified and discarded, as an infrequent component cannot become a frequent pattern[2].

To begin, we could simply test for formulae describing things such as "event e occurs" (i.e. $\mathsf{AF}e$) and count how many times this statement is true throughout the given sequence of events. Again, we do not test if a formula holds over the entire dataset, as in many cases we may only see parts of the whole pattern occurring at any given time and we want the ability to find pattern members who occur *most*, if not all of the time. Once we have these counts, we skim off those which occur significantly more often than the rest. We will next derive the calculations for a pattern of size two, but it can easily be extended for any n.

We start with the hypothesis $A \rightarrow B$ and the evidence A, and B. By Bayes' Rule, we know that the probability of the hypothesis given the evidence is:

$$P(A \rightarrow B|A,B) = \frac{P(A,B|A \rightarrow B)P(A \rightarrow B)}{P(A,B)} \tag{1}$$

What the formula above says is that our belief in $A \rightarrow B$, after we see the evidence A, and B (our posterior probability), is equivalent to our prior belief, $\mathrm{P}(A \rightarrow B)$ multiplied by the likelihood that we will see A and B if it is true. The intuition in counting occurrences of a pattern is reflected very closely in the actual calculation, as we use these counts as well as the number of events $e \in E$ and the number of times the formula does not hold to determine the likelihood of a pattern. This expression can also be formulated in terms of prior odds and a likelihood ratio that gives the posterior odds.

Continuing this example for $A \rightarrow B$ given A and B (computing the posterior odds for a pattern of length 2), by using the usual manipulations of the basic Bayesian framework, we end up with the three equations, where O denotes odds and L denotes likelihood.

$$O(A \rightarrow B) = \frac{P(A \rightarrow B)}{1 - P(A \rightarrow B)} \tag{2}$$

$$L(A,B|A \rightarrow B) = \frac{P(A,B|A \rightarrow B)}{P(A,B|\neg A \rightarrow B)} \tag{3}$$

$$O(A \rightarrow B|A,B) = \frac{P(A \rightarrow B|AB)}{P(\neg A \rightarrow B|A,B)} \tag{4}$$

Thus, we see that (Equation 2)$\times$(Equation 3) gives Equation 4, the posterior.

We consider some patterns *interesting* enough to be extended depending on the value of their posterior odds. Fine tuning this parameter of our setup is experiment-dependent. Another value that may be considered is the allowed time lapse between events A and B. They may be considered with any separation in time, with a minimum separation or maximum separation, or with a set minimum and a set maximum (i.e., a window or slot of time in which we will consider B to be related to A). A full-fledged development of this strategy will be completely investigated in a future work.

The algorithm detailed in Algorithm 1 shows the procedure starting from an alphabet of terms labeling events: A, a list of events: E (with each item in the list being an alphabet term and the time at which it acted), a set

[1]The universe of temporal formulae shrinks after each stage, as components are removed, and there are fewer ways in which to combine the components that remain.

[2]This does not necessarily mean that infrequent components of frequent patterns will not be interesting to us, simply that we do not automatically seek such events.

Algorithm 1 Finding temporal patterns

Given:
A, an Alphabet
E, a list of paired $a \in A$ and times t {There may be multiple events or no events at each t from t_0 to final time T}
n, the maximum length of patterns {Length of a pattern is the number of alphabet identifiers within it, regardless of how many edges or formulae connect them}
Threshold, the minimum posterior odds of the Bayesian analysis

Initialize:
Candidates $\leftarrow \emptyset$
Patterns $\leftarrow \emptyset$
Singles $\leftarrow \emptyset$
Base case:
Singles $\leftarrow A$
for S in *Singles* **do**
  $S.Score \leftarrow$ Bayesian(S, E)
  if $S.Score > Threshold$ **then**
    Add S to *Patterns*
    Add S to *Candidates*
  else
    Remove S from *Singles*
  end if
end for

General Case:
{Try extending patterns until maximum length is reached}
for $i = 2$ to n **do**
  Candidates $\leftarrow$ Extend each C in *Candidates* with each S in *Singles* and each possible permutation of this extension
  {each C will have length $i - 1$ before this operation, and length i after}
  for C in *Candidates* **do**
    $C.Score \leftarrow$ Bayesian(C, E)
    if $C.Score > Threshold$ **then**
      Add C to *Patterns*
      {Note that C remains in *Candidates* for later extension}
    else
      Remove C from *Candidates*
    end if
  end for
end for

maximum pattern length: n, and a threshold for the Bayesian likelihood calculation. The section "Base Case" on line 9 shows the procedure for a pattern of length 1. This step consists of iterating over each $a \in A$ and computing the likelihood of a. If it is greater than the set threshold (i.e. it is "valid"), it remains in the group of length 1 patterns and is also added as both a valid pattern and a candidate pattern that may be then extended. Each time a pattern is found to be valid, it is added to the list of patterns to be returned. So, eventually, one may get a list such as: A, B, $A \rightarrow B$, and $A \mathsf{U} B$. In this notation, $\rightarrow$ means only that A is followed by B. It is a weaker statement than $A \mathsf{U} B$, which means A *until* B.

During the $(i+1)^{\text{th}}$ iteration of the procedure, the remaining events from the last iteration of length i are recombined into all possible patterns of length $i+1$ and the resulting formulae are tested as described before(See line 21 of algorithm 1). For the second iteration, for each candidate (C) and "single" term (S) there are only two possible combinations $C \rightarrow S$ or $S \rightarrow C$. In general, the number of combinations is twice the length of the pattern (each single term can be added before or after each part of the pattern).

Note also that there may be remaining patterns of *length* $< i$, but these will not be augmented during this iteration, as they were not successfully extended during earlier iterations and will not be in the candidate list (they will remain in the pattern list to be returned later). During each iteration, patterns which were not valid are removed as candidates, and at each step we will be extending fewer patterns as we approach length n.

After collecting all formulae of interest, we would like to connect them in a way that allows us to see the likelihoods of the components as well as of the larger temporal formulae and patterns. One method of doing this is by constructing a neural circuit [8, 12], and in fact this follows directly from the collected formula. We simply connect all of the interesting components[3] starting from those of size one up to the largest patterns. For instance, if our collected formulae include A, B and $A \wedge B$, we would connect A and B as children of $A \wedge B$, with the edges labeled with weights. In this case, those weights are the calculated posterior odds. These formulae can then be further combined with others using logical connectives.

For example, if we now have the subtree signifying $A \mathsf{U} B$ and previously combined formulae to get $C \mathsf{U} B$, these may be logically combined to form $(A \vee C) \mathsf{U} B$, another subtree of the overall network. As a pattern, this circuit would be a graph with A and C connected to B by directed edges. In this manner, by combining patterns we are able to recover bigger circuits from the data, while using logical rules, we were able to form more meaningful descriptions of these patterns. By assembling the formulae as a potentially un-rooted tree, we allow for the possibility that we may recover one pattern that is exactly correct, as well as some with lower likelihoods but which meet our threshold for validity. We are thus able to choose sub-optimal paths in the final network and, using outside knowledge of the experimental data, be able to determine their relevance or whether they are worth exploring.

5 Experimental Studies

As described earlier, our experimental studies have so far consisted of analyzing the challenge datasets to recover the neural circuits form multi-neuronal time series data. Below we describe the generated datasets and the performance of the algorithm described earlier.

5.0.1 Generation of Synthetic Datasets

The synthetic data consisted of 5 sets of data, each with a different planted pattern. These patterns were created using 26 neurons, labeled A-Z. The firing of each neuron depends on the input it receives from other neurons, so by weighting the edges in the network between the 26 neurons, it is possible to make some connections more likely than others. Thus the data contained firings of all neurons, with the planted patterns repeated more frequently than others.

For each of the five patterns there were four data files, as data was generated for two different noise levels, 0.1 and 0.25. Then, for each noise level there were two sets, each having 100,000 firings generated using the network embedded with the pattern to be discovered. Each line in these data files contained a neuron label and the firing time of that neuron. It is possible for multiple neurons to fire at a single time, so the actual number of ticks of time was somewhat less than 100,000.

[3] An interesting component is defined here as one that has a score above our threshold.

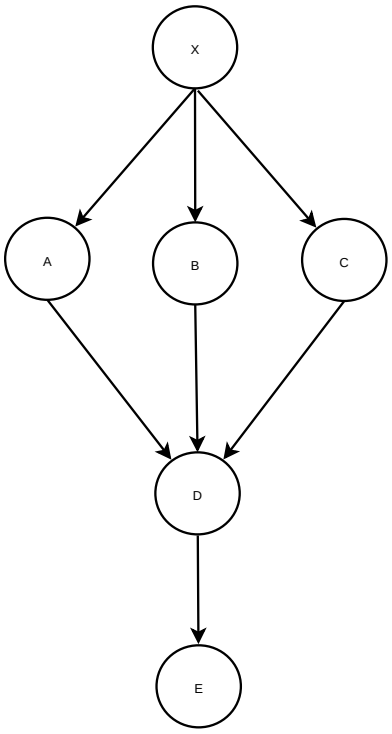

Figure 1: Pattern 1 as recovered

To plant the patterns, first a graph between the 26 neurons was created, using random connections with weights evenly distributed in the interval [-1,1]. Then, the patterns were added by increasing the weights between nodes involved in the pattern. When each neuron fires, it remains "on" for 40 units of time and then has a refractory period of 20 units, during which it cannot fire again. Additionally, in the interval [20,40] after the neuron fires, each neuron to which it is connected is considered for firing. If the amount of input to a neuron (from all neurons) exceeds a certain threshold, it then fires. Neurons are also allowed to fire randomly, with the probability of this random firing corresponding to the noise level of each file.

5.0.2 Procedure

We recovered the networks from the synthetic data using the procedure outlined previously, with some features tuned to the nature of the data. In particular, we only considered neurons to be potentially related if they fired within the interval $[20, 40]$, i.e. within 20 to 40 time units of one another.[4]

After running the procedure for patterns of size two, we found that the likelihood as computed above was on the order of 10 times higher for frequently occurring patterns than for random patterns and noise. The calculation for this synthetic data, for a pattern size of two, is the likelihood multiplied by the prior odds, as shown in equations 2, 3 and 4:

$$\frac{E}{(\neg A \rightarrow B)} \times \frac{\frac{A \rightarrow B}{E}}{1 - \frac{A \rightarrow B}{E}} \tag{5}$$

where E is the number of events in the sample, and the other terms are counts of the number of times that particular formula is true in the sample. We point out that the likelihood ratio has been simplified in this calculation. This simplification is necessary owing to the fact that $P(A, B|A \rightarrow B)$ is 1, so the ratio is then $\frac{1}{P(A,B|\neg A \rightarrow B)}$, where the denominator of the ratio is also a fraction, so we end up with the equation shown above.

[4]Future work would include procedures for recovering this window as well as the pattern of neurons.

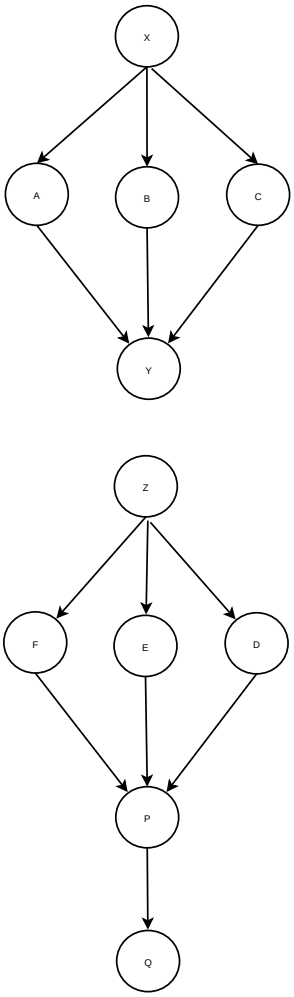

Figure 2: Pattern 2 as recovered

While it is possible to connect patterns of length two together to find larger patterns, we have avoided such a greedy approach, as it does not allow the construction of the neural circuit using the fullest potential of our algorithm. Furthermore, in this case, we will not have the correct weights for all edges. Also, all neurons that were not in the final pattern set were weeded out by the successive iterations, so the set of patterns being extended dwindles rapidly.

Additionally, we can also find patterns that occur infrequently enough to be of interest, such as when one neuron negatively influences another (i.e. if neuron A fires, it is less likely that neuron B will fire). In that case, we would need a second likelihood threshold, and would look for patterns falling on the other end of the spectrum.

5.1 Example networks "planted" and reconstructed

Of the five planted patterns, four were recovered exactly. For two of the other patterns, we either missed an edge or found additional superfluous edges which on first glance were indistinguishable from the planted edges. However, after tuning the thresholds, most erroneous edges could be removed. We found that noise in the system did not alter our results with the synthetic data, because while for the most part it simply increased the overall counts uniformly, they still remained in roughly the same proportion. Additionally, we tested our system using the full set of 100,000 events as well as on subsequences of the provided data. We found that even for as few as 2,000 events, we were able to recover the same patterns as with the full dataset. The time needed (from processing the data to finding the patterns) for 2,000 entries was 5.609 seconds, while the dataset with 100,000 events took 180.4

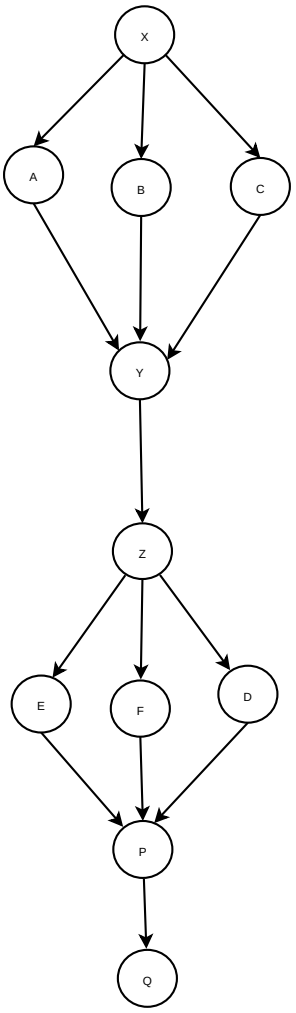

Figure 3: Pattern 3 as recovered

seconds. Below we discuss our results for each of the five patterns to be reconstructed. The tables contain data showing our results for the five patterns, in each case using the first file with the lower noise level. Results were similar for the other data files.

5.1.1 Pattern 1

This pattern consisted of six nodes combining both parallel and serial configurations. After applying the base case of the algorithm, we were able to identify the labels involved in the pattern (see Table 2). The support for these nodes were the only ones greater than the threshold of .045. Nodes not in the pattern had scores in the 0.02-0.03 range. Then, looking at patterns of length 2, seven had posterior odds greater than the threshold of 0.8. The lowest was 0.86, and the highest was 2.35. Looking at the actual patterns(fig. 1), we can see that these posterior odds are consistent with the diagram. The patterns AD,BD and CD have odds that are about a third of that for XA XB and XC. A, B, and C are only triggered by X, but D may be triggered by either A, B or C. Also, after examining the odds, we concluded that X may have a stronger effect on B and C than on A. Looking at the patterns of length 3, there was better support for $((C\ D)\ E)$ than for $(C\ (D\ E))$. At first glance, it may seem that the posterior odds should be the same, but in fact we can see that what this means is that when C triggers D, the odds are that E is then triggered. However, the odds of C triggering D, which then triggers E are lower, as B and A may also be triggering that subpattern. Here one of the advantages of our method becomes clear, as our

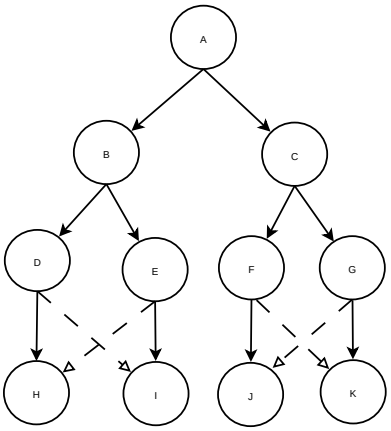

Figure 4: Pattern 4 as recovered

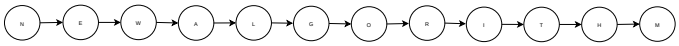

Figure 5: Pattern 5 as recovered

method allows one to quantify the relationships between the neurons. Finally, for patterns of length four, there were four chains supporting the way we connected the nodes. Four other patterns were recovered, suggesting that after D triggered E, X was triggered and the pattern began again.

5.1.2 Pattern 2

Pattern 2 was comprised of two instances of Pattern 1, totalling 11 neurons. The first threshold we used when recovering the 11 neurons was 0.045. This yielded 10 neurons. This threshold missed P, which had a score of 0.037. Lowering our threshold to 0.035, we recovered all of the neurons in the pattern, plus four erroneous neurons (H,M,O and S). These extra neurons were not found to be in any significant patterns of length 2, of which we recovered 13. For this we used a threshold of 0.5. We were able to recover that the odds are better for sub-patterns stemming from X (such as those of length two with X as the first node) than for other sub-patterns of Pattern 2. This is consistent with the description of the data, which states that the first node of a pattern is given more weight in order to bias the output in favor of repetitions of the pattern.

5.1.3 Pattern 3

This pattern consisted of 11 nodes connected to form a variety of patterns of length 7. Our first attempt to recover the nodes involved in the pattern used a threshold of 0.04. This recovered 10 of the nodes, but missed P, which had a score of 0.039. Next, we lowered the threshold to 0.035 and recovered all 11 nodes, plus H, M and O. Then, looking at patterns of length 2, with a threshold of 0.75, we recovered all of the correct connections. During this step, the three extra nodes found in the first step (H, M and O), were not involved in any patterns. We found it was better to use a lower threshold to recover as many nodes as possible, as unrelated nodes tended to drop out as computation progressed. As shown in Table 4, the only nodes involved in patterns of length 2 were those intended to be part of the pattern being recovered. For patterns of length three, there were some cases with patterns such as (Z (C Y)) as well as ((C Y) Z). In all such cases, where the ordering of nodes remained the same, we connected the nodes in our resulting graph according to the pattern with the high score.

5.1.4 Pattern 4

The pattern to be recovered was a four level binary tree. We approached this pattern in a manner identical to the others, though we found the recovery of the correct connections in some cases to be a little more difficult without disambiguating them manually. In particular, the tree recovered (see Fig. 4 and Table 5) is correct for the first

Pattern	Posterior Odds	Pattern	Posterior Odds
A	0.06428	X A	2.0185
B	0.06715	X B	2.3465
C	0.06820	X C	2.2793
D	0.04773	D E	1.8553
E	0.05967	C D	0.88454
X	0.06516	B D	0.90490
		A D	0.85891

Table 2: Odds of sub-patterns for Pattern 1, up to length 2

Pattern	Posterior Odds	Pattern	Posterior Odds
A	0.06096	A Y	0.78831
B	0.05859	B Y	0.75048
C	0.05782	C Y	0.75365
D	0.05535	D P	0.59831
E	0.05163	E P	0.55121
F	0.05678	F P	0.64212
H	0.03665	P Q	0.93904
M	0.03979	X C	1.8497
O	0.03800	X B	1.8006
P	0.03776	X A	1.8420
Q	0.05082	Z F	1.3211
S	0.03715	Z E	1.4105
X	0.05501	Z D	1.2936
Y	0.05444		
Z	0.05011		

Table 3: Odds of sub-patterns for Pattern 2, up to length 2

three levels, but then some of the third level nodes (D, E, F, and G. See dashed edges) were connected to the fourth level nodes(H,I, J and K) with a likelihood that met our initial threshold, though these connections did not maintain the binary tree structure that was to be recovered[5]. This was especially true of the data sets with higher noise levels. For each of these fourth level nodes, however, there was one third level node connecting to it with much higher odds than that with which it connected to the others, shown with the solid lines in the figure(fig. 5). After increasing the threshold to .55, only one erroneous edge remained, namely, the one between E and H, but after accounting for the posterior odds for the other nodes which were all greater than 1 and comparing, we could determine that this edge was not intentionally part of the pattern. Thus our approach to pattern recovery is able to determine only the most likely patterns, as would be expected. However, by suitably decreasing the threshold parameter to a less stringent value, it can recover all frequent patterns and then use outside information to decide what patterns should be kept.

5.1.5 Pattern 5

This pattern was a linear chain of nodes spelling the phrase "NEWALGORITHM". First, beginning with a threshold of 0.04, we recovered the 12 neurons in this pattern. Then, looking at patterns of length two, beginning with a threshold of .50, we recovered all of the paired connections. Note that the actual posterior odds here were much higher than our threshold (see Table 6). In fact, the lowest value here is 1.74. For finding patterns longer than two, when the threshold was kept at 0.5, we found a few erroneous patterns, but the gap in posterior odds between legitimate patterns and false patterns was large.

[5]Described in the table are the results for one low noise dataset where B was also incorrectly identified as triggering both G and F.

Pattern	Posterior Odds	Pattern	Posterior Odds
A	0.05956	A Y	0.78768
B	0.05721	B Y	0.81720
C	0.05559	C Y	0.76896
D	0.06007	D P	0.80292
E	0.05691	E P	0.85173
F	0.06158	F P	0.82746
H	0.03515	P Q	1.0720
M	0.03889	X C	1.7667
O	0.03582	X B	1.7934
P	0.03948	X A	1.7191
Q	0.05047	Y Z	1.0054
X	0.05319	Z F	2.3966
Y	0.05296	Z E	2.4018
Z	0.06553	Z D	2.0494

Table 4: Odds of sub-patterns of Pattern 3, up to length 2

Pattern	Posterior Odds	Pattern	Posterior Odds
A	0.05653	A C	1.7987
B	0.05431	A B	2.4798
C	0.04589	B G	0.53101
D	0.05981	B F	0.50686
E	0.05659	B E	1.9007
F	0.05678	B D	1.9450
G	0.05690	C G	1.5640
H	0.06344	C F	1.4644
I	0.06292	D H	2.7544
J	0.05806	E I	2.5380
K	0.05994	E H	0.56339
		F J	2.3837
		G K	2.5123

Table 5: Odds of sub-patterns of Pattern 4, up to length 2

6 Discussion

This paper describes a novel technique to infer dynamics of a temporal process by examining the output time-series data from such a process. The algorithm described here works by first detecting many important temporal patterns that occur more often than what would, if they were simply due to mere chances. Such patterns are encoded in terms of the formulae in an interesting class of modal logic: namely, the propositional branching-time temporal logic of CTL. While CTL enjoys an enormous popularity in the domain of model checking as it is applied to hardware and protocol verification, the current application of the logic is novel in finding patterns and combinatorial interconnections among components of a generating system. A related set of ideas has also appeared in the TSKM and Apriori algorithms described earlier, but there are still many key differences in how the patterns are explored and exploited.

The resulting algorithm has been applied to a challenge dataset generated from the fourth SIGKDD workshop on temporal data mining, to understand how algorithms like these could be applied to infer the underlying neural circuits from multi-neuronal time series data. The algorithm presented here is computationally efficient and yet surpasses in its accuracy all other algorithms that have analyzed this dataset. Nonetheless, our approach is still in its infancy, and requires additional work to place it on a firmer statistical foundation, especially in terms of computing p-values (e.g., by using scan statistics), and then using them to control the false-discovery rates (FDR).

Pattern	Posterior Odds	Pattern	Posterior Odds
A	0.05081	A L	2.0776
E	0.05489	E W	2.8986
G	0.04681	G O	1.7359
H	0.05853	H M	3.8739
I	0.05609	I T	2.6264
L	0.04356	L G	2.0444
M	0.05900	N E	2.5346
N	0.05181	O R	2.7063
O	0.05106	R I	2.6572
R	0.05331	T H	4.6024
T	0.05817	W A	2.0071
W	0.05285		

Table 6: Odds of sub-patterns of Pattern 5, up to length 2

Acknowledgements

We acknowledge the helpful contribution of Debprakash Patnaik, K.P. Unnikrishnan (General Motors R & D) and P.S. Sastry (Indian Institute of Science, Bangalore) for creating the challenge datasets and descriptions.

References

[1] R.C. Agarwal, C.C. Aggarwal, and VVV Prasad. Depth first generation of long patterns. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 108–118, 2000.

[2] R. Agrawal and R. Srikant. Fast algorithms for Mining association rules. *Proc. of the 20th VLDB*, 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, 1995.

[4] Z. Bar-Joseph. Analyzing time series gene expression data. *Bioinformatics*, 20(16):2493–2503, 2004.

[5] R.J. Bayardo Jr. Efficiently mining long patterns from databases. *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93, 1998.

[6] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20(3):207–226, 1983.

[7] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 255–264, 1997.

[8] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[9] E.A. Emerson. Model checking and the Mu-calculus. *DIMACS Series in Discrete Mathematics*, 31:185–214, 1997.

[10] J.D. Hamilton and J. Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[12] JJ Hopfield and DW Tank. Computing with neural circuits: a model. *Science*, 233(4764):625, 1986.

[13] S. Kleinberg, M. Antoniotti, S. Tadepalli, N. Ramakrishnan, and B. Mishra. Remembrance of experiments past: a redescription approach for knowledge discovery in complex systems. In *International Conference on Complex Systems*, 2006.

[14] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[15] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovering frequent episodes in sequences. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD95)*, pages 210–215, 1995.

[16] F. Moerchen. Algorithms for time series knowledge mining. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[17] A. Pnueli. The temporal logic of programs. 1977.

[18] J. Srivastava, R. Cooley, M. Deshpande, and P.N. Tan. Web usage mining: discovery and applications of usage patterns from Web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23, 2000.

[19] L. Wittgenstein. *The Blue Book and the Brown Book.* s.n., 1934.