Multiple Uses of Frequent Episodes in Temporal Process Modeling

Debprakash Patnaik

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Naren Ramakrishnan, Chair
T. M. Murali
Yang Cao
Manish Marwah
Srivatsan Laxman

August 4, 2011
Blacksburg, Virginia

Keywords: Temporal Data Mining, Graphical Models, Motifs, Frequent Episodes,
Dynamic Bayesian Networks

Multiple Uses of Frequent Episodes in Temporal Process Modeling

Debprakash Patnaik

(ABSTRACT)

This dissertation investigates algorithmic techniques for temporal process discovery in many domains. Many different formalisms have been proposed for modeling temporal processes such as motifs, dynamic Bayesian networks and partial orders, but the direct inference of such models from data has been computationally intensive or even intractable. In this work, we propose the mining of frequent episodes as a bridge to inferring more formal models of temporal processes. This enables us to combine the advantages of frequent episode mining, which conducts level wise search over constrained spaces, with the formal basis of process representations, such as probabilistic graphical models and partial orders. We also investigate the mining of frequent episodes in infinite data streams which further expands their applicability into many modern data mining contexts. To demonstrate the usefulness of our methods, we apply them in different problem contexts such as: sensor networks in data centers, multi-neuronal spike train analysis in neuroscience, and electronic medical records in medical informatics.

This work is supported in part by US NSF grants CNS-0615181, CCF-0937133 and IIS-0905313, HP Labs, General Motors Research and the Institute for Critical Technology and Applied Science (ICTAS) at Virginia Tech.

Acknowledgments

It is my great pleasure to thank the people who made this thesis possible. Dr. Naren Ramakrishnan, my thesis advisor, has been my greatest source of inspiration in this entire journey which now culminates in this dissertation. Every step of the way he has provided valuable insights into the data mining problems. He has helped me understand the different subtleties of data mining methods and probabilistic models.

I am deeply grateful to Dr. Srivatsan Laxman who has spent innumerable sleepless nights sharing with me valuable intuitions about the different data mining problems addressed in this dissertation. He has been a great friend and has helped with words of encouragement and advice every time I hit a wall trying to solve a problem. His positive attitude and his faith in me have been instrumental in the completion of this thesis.

I want to thank Dr. Manish Marwah for giving me the opportunity to work on some of the most challenging problems in knowledge discovery. With his help and guidance we were able formulate a data mining approach to an important problem in sustainability. His eye for detail and continuous persuasion to better our methods has lead to significant improvement in the quality of this work.

I am indebted to Dr. T. M .Murali and Dr. Yang Cao for their unfailing support as my thesis committee members. I am thankful for their technical insights which have helped me overcome some of the key hurdles in this work.

During our work on analysis of electronic medical records, I had the pleasure of working with Dr. Laxmi Parida. Her earlier work in the area of genomics served as an inspiration and formed the basis of our work. I also thank her for helping me formulate the statistical significance test for the problem.

I owe my deepest gratitude to Dr. K. P. Unnikrishnan and Dr. P. S. Sastry. Without their help and motivation I would not have undertaken this research work.

Finally, this thesis would not have been possible without the unwavering support of my wife, Priyanka and my little boy, Aditya and the blessings of my parents. They held the boat steady in the rough waters and kept urging me forward.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The area of data mining [3] is concerned with analyzing large volumes of data to automatically unearth interesting information that is of value to the data owner. Interestingness may be alternatively defined as regularities or correlations in the data, or, irregularities or unexpectedness, depending on the nature of the application. Some typical applications of data mining are deciphering customer buying patterns in market research [4], weather forecasting using remote sensing data [5], motif discovery in proteins and gene sequences [6, 7], and finding patterns in telecommunication alarm sequences [8]. The regularities found in the data can be represented as association rules, clusters, and recurrent patterns in time series etc. The massive sizes of the datasets require any data mining algorithm to be efficient in terms of both memory and time requirements.

Pattern discovery is an important data mining task. In pattern discovery, the algorithms are designed to efficiently discover *frequent* regularities in the data. Some of the pattern classes that can be grouped under this topic are frequent itemsets [4, 9, 10, 11, 12, 13], sequential patterns [14, 15, 16], frequent episodes [17, 8, 18, 19], biclusters [10, 11, 20, 21, 22], and redescriptions [23, 24, 25]. Discovering such patterns is a difficult task as the search space of interesting patterns is most often exponential and requires careful design of algorithms to exploit some regularity or structure in the data. Much of the data mining literature is concerned with formulating useful pattern structures and developing efficient algorithms for discovering such patterns that are of domain interest. Many efficient algorithms have been proposed to unearth specific classes of patterns that occur sufficiently often in the data [26].

Another active area of research in data modeling is that of *graphical models* [27, 28] such as Bayesian networks [29] and Markov random fields [30]. Probabilistic modeling of data differs from pattern mining in that the goal here is to infer a global model that explains the entire data. In contrast, frequent patterns only reveal local features in the data and are not meant to explain the data in its entirety.

In this work we endeavor to connect the above two seemingly disparate methodologies by

exploring specialized classes of probabilistic models that can be learnt using frequent patterns in the context of temporal data. This offers two significant advantages (to the respective methodologies). First, it yields a sound theoretical basis for pattern mining algorithms. Second, the problem of structure learning in graphical models can benefit from fast and efficient pattern mining algorithms.

1.1 Temporal data mining

Datasets with temporal dependencies, i.e., where records are time-ordered, frequently occur in business, engineering, and scientific scenarios. Temporal data mining is concerned with mining of such large sequential datasets. These datasets can be broadly classified into two categories: sequential databases which are large collections of relatively short sequences and discrete event stream data which consist of one long sequence of events. Pattern mining in such datasets differs from classical time series analysis [31] in the kind of information that one seeks to discover. The model parameters (e.g. coefficients of linear regression model or the weights of a recurrent neural network) are unimportant. Interpretable trends or patterns are of greater value. Several efficient techniques for learning different types of patterns in ordered data abound in the temporal data mining literature [32]. The structure of the interesting patterns and data vary in each of the techniques. There are primarily two popular frameworks in temporal data mining, viz. sequential patterns [14] and frequent episodes [8].

Sequential pattern discovery: The framework of sequential pattern discovery was introduced in [14]. The database here consists of a collection of sequences. A sequence is an ordered list of itemsets $\langle I_1, I_2, \ldots, I_n \rangle$. An itemset I_j in turn is a set of items, i.e. $I_j = \{i_1, i_2, \ldots, i_k\}$. A frequent sequential pattern is defined as a sequence of itemsets, which is contained in sufficiently many sequences of the database. Finding motifs in gene/protein sequences and finding purchase patterns in market research data are instances of the sequential pattern mining problem.

Frequent episode discovery: The frequent episode discovery framework was proposed in [8]. In the frequent episode framework, the data given is a single long sequence of events, $\langle (E_1, t_1), (E_2, t_2), \ldots, (E_n, t_n) \rangle$, where E_i represents the i^{th} event type and t_i is the time of occurrence of the i^{th} event. The task here is to find temporal patterns (called *episodes*) that occur sufficiently often along that sequence. Examples of such data are alarms in a telecommunication network, fault logs of a manufacturing plant, and multi-neuronal spike train recordings. The temporal patterns, referred to as episodes, are ordered collections of event types. For example, $(A \rightarrow B \rightarrow C)$ is a temporal pattern where an event type A is followed some time later by a B and then a C, in that order, but not necessarily consecutively.

An episode is declared as interesting if it repeats sufficiently enough in the event sequence. The frequent episode framework aims at discovering all episodes that occur often in the data based on a user-defined frequency threshold. Several level-wise data mining algorithms have been suggested in the existing literature [8, 17] for frequent episode discovery. Unlike in the frequent itemsets problem, identifying a frequency or support measure that allows efficient counting procedures has been shown to be crucial here [33, 34].

1.2 Graphical models

Probabilistic modeling of temporal data has also been an active area of research. It differs from pattern mining in that the intent here is to infer a global model that explains the entire data. (In contrast, frequent patterns only unearth local features in the data.) Dynamic Bayesian networks [35], hidden Markov models [36], and Kalman filters [37] are a few of the dynamical modeling strategies aimed at capturing probabilistic dynamical behavior in complex time evolving systems. These models are now widely used in bioinformatics, neuroscience, and linguistics applications.

In general, probabilistic models aim to provide compact factorizations of the joint probability distributions of sets of random variables. One such example, from Bayesian networks, is shown below:

$$P(X_1, X_2, \ldots, X_n = x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(X_i = x_i | Parent(X_i))$$

Here the joint distribution of random variables X_i is expressed in terms of several local conditional distributions $P(X_i = x_i|Parent(X_i))$. $Parent(X_i)$ is a minimal set of random variables that directly determine the distribution of X_i.

A graphical model in addition allows us to interpret the model in terms of graph theory. Probabilistic models can be represented as graphs where the nodes symbolize random variables and the edges between the nodes encode dependencies between corresponding random variables. Broadly there are two class of graphical models: *Undirected graphical models* and *Directed graphical models*. Undirected graphical models (e.g. *Markov Random Fields*) encode independence relationships of the following form: a node A is independent of every other node in the graph conditioned on the immediate neighbors of A. In general two nodes A and B are conditionally independent given a third set $\mathcal{C}$, if all paths between A and B go through $\mathcal{C}$. On the other hand, in directed graphical models, also known as Bayesian networks, the independence relationships are harder to read-off from the graph and entail a sequences of rules known as d-separation. One of the simplest independence relations captured by a Bayesian Networks is the following: a node A is independent of all its non-descendants given its parent set. Directed models have a more complicated notion of independence yet they can be used to depict causality (since the edges are directed and can represent time ordering).

1.2.1 Dynamic Bayesian networks (DBNs)

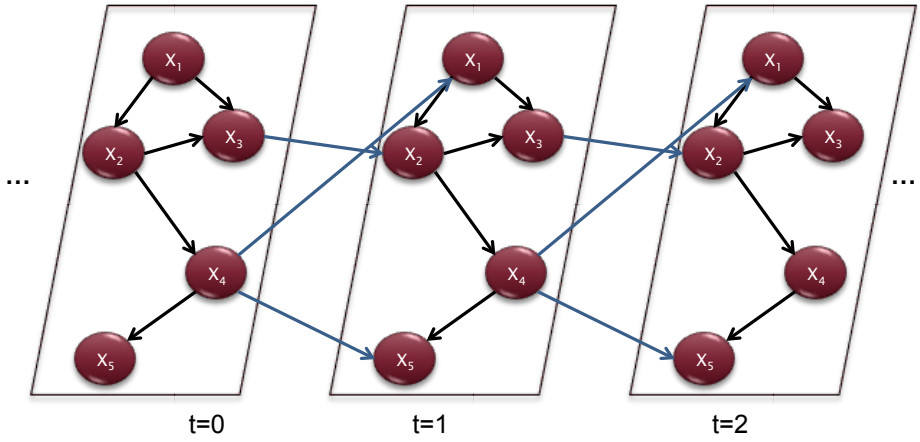


Figure 1.1: Example of an unrolled dynamic Bayesian network. Note the within slice and between slice dependencies in this network.

Dynamic Bayesian networks (DBNs) are a class of Bayesian networks specially designed to capture conditional independencies over time. A typical DBN represents the state of the world using a set of random variables $\mathbf{X}(t) = \{X_1(t), X_2(t), \ldots, X_k(t)\}$. (Some of these random variables could be unobservable.) In the example shown in Figure 1.1, we have dependencies within a time slice and across time slices. Here the temporal dependency is first order Markov because of the single time step dependency.

DBNs bring to modeling temporal data the key advantage of using graph theory to capture probabilistic notions of independence and conditional independence. Learning the DBN structure from the data requires identifying significant edges in the network. There are two broad classes of algorithms for learning DBNs: score-based and constraint-based. Score-based algorithms aim to search in the space of networks guided by a score function. Friedman's sparse candidate approach [35] refers to a general class of algorithms for approximate learning of DBNs. The approach is iterative: in each iteration, a set of candidate parents are discovered for each node, and from these parents, a network maximizing the score function is discovered. Conditioned on this network, the algorithm then aims to add further nodes that will improve on the score. The approach terminates when no more nodes can be added or when a pre-determined number of iterations are met. One instantiation of this approach uses the BIC (Bayesian Information Criterion) as the score function. The simulated annealing approach from [38] begins with an empty network and aims to add edges that can improve the score. With some small probability, edges that decrease the score are added as well. An underlying cooling schedule controls the annealing parameters. Constraint-based algorithms such as GSMN [39], on the other hand, first aim to identify conditional independence relationships across all the variables and then aim to fit the discovered relationships into a network, much like fitting a jigsaw puzzle.

1.3 Goals of the dissertation

As we have seen so far, there are broadly two distinct approaches in analyzing time stamped data, namely, learning a global model that explains the entire data or, identifying local patterns that occur sufficiently often in the data. The former approach falls in the domain of machine learning and statistical modeling whereas the later has been studied extensively in data mining literature. It is natural to explore whether these two threads can be associated with one another.

Many researchers have explored such connections in the context of frequent itemsets and probabilistic models like Bayesian networks and Markov random fields. For instance, frequent itemsets have been used to constrain the joint distribution of item random variables and thus aid in learning models for inference and query approximation [40]. Also probabilistic models are viewed as summarized representations of databases and can be learnt from frequent itemsets in the data [41]. In the area of discrete event sequences, frequent episodes have been linked to learning specialized hidden Markov models for the underlying data [17]. However, there is no broad marriage of these two threads of this research for temporal process modeling, as we seek to explore here. We propose four broad problems in this space, and propose data mining style methods to infer formal temporal representations from data.

Topic 1: Motif mining in multi-variate time-series data

Multi-variate time-series data (i.e., continuous-valued) occur in many domains, most notably sensor networks. Our motivation derives from sensor networks associated with the cooling infrastructure of a data center run by HP. (We present our algorithms in relation to this application but it should be clear that the same approach can be adapted to many other data mining contexts.) The cooling set up consists of an ensemble of chiller units, cooling towers, and computer room air-conditioning units (CRACs). Together these units are responsible for rejecting the heat generated by the IT equipment inside the data center to the outside environment. The various sensors monitor physical variables such as temperature, pressure, flow rates at different points, and utilizations of subsystems. The research question here is to discover patterns (i) that can characterize the operation of the chiller units, (ii) that are relatable to sustainability metrics of interest to the data center administrator, and (iii) which can be used to construct probabilistic models of the sensor network.

We propose an approach to address these issues through motifs, which are characteristic subsequences that approximately repeat several times over a time series. A number of algorithms have been proposed in the past for motif discovery [42, 43, 44, 45]. However, most of these methods are not robust to scaling or stretching of the patterns, can handle only limited amount of noise in the data, and also suffer from the discovery of too many patterns. In view of these shortcomings, we propose an efficient data mining style algorithm for discovering repeating motif structures in multivariate time series data. We describe how

we can evaluate the motifs in terms of sustainability metrics such as total power consumption and frequency of cycling (as rapid switching on and off of chiller units is known to reduce their operating life). This will allow us to compare different motifs and help think about interventions that can move the system from a 'bad' motif state to a more favorable one.

We next generalize these ideas to develop a probabilistic model of chiller operation. If we consider the sensors in the cooling infrastructure as random variables and the data as the trace of a random process involving these sensors, then learning the structure of a dynamic Bayesian network will help reveal important dependencies among the sensors. In particular, we use the discovered motifs to enrich the underlying representation and help define a notion of 'internal state' to track the state dynamics. Such a probabilistic model can then be used for tasks like inference and prediction.

Topic 2: A new model class in dynamic Bayesian network learning

Learning the structure of models like dynamic Bayesian networks is a hard problem to be solved optimally. Therefore, the methods proposed for structure-learning have poor runtime and memory performance or employ heuristic search strategies. In contrast, motifs in time series data and frequent episodes in event sequences are interesting patterns that can be efficiently discovered using data mining style algorithms. It is generally perceived that dynamic probabilistic models and frequent episodes or motifs are orthogonal representations of temporal data.

We investigate the question of whether there exist special classes of DBNs that can be directly associated with frequently occurring episodes or motifs in the data. While much of current work has placed restrictions on the graph structure of the DBN, we instead propose placing restrictions on the form of the conditional probability tables (distributions) used in encoding the network. We show how such a class of DBNs can be learnt directly from pattern mining algorithms, thereby leveraging their efficiency, in terms of both runtime and memory for structure learning (which is a hard problem in a general setting). We envisage that such a data mining capability can be applied to modeling discrete event sequences with many applications in areas such as bioinformatics and neuroscience. Specifically, we propose investigating multi-neuronal spike train recordings that capture the electrical activity of neurons in cell cultures and brain slices. Nevertheless, our algorithms can be adapted to more general settings including multi-variate time series data from sensor networks.

Topic 3: Learning augmented partial orders from electronic medical records

Recently, there has been a lot of interest in the health informatics arena. Health care providers all over the country are embracing electronic medical records systems. At a high

level, an electronic medical record can be thought of as a sequence of time-stamped events where each event is characterized by a diagnostic or procedure code. The data consists of such sequences for several thousands of patients. In this work we analyzed 10 years of medical records data from the University of Michigan Medical School. This consisted of over 1.5 million patients and over 40 million diagnostic codes.

Our goal in this work was to extract clinically relevant information from patient histories. We formulated a new model class for summarizing the temporal ordering of frequently occurring events in patient histories. Using an augmented partial order defined over diagnostic codes, we generate ordering rules that are implicitly obeyed in the data. This provides a general framework for understanding the patterns of disease and uncover nuggets of information hiding in the medical records data.

Our proposed framework, first finds all frequently occurring sequences of codes. This results in too many frequent sequences to be analyzed by a health care professional. Next we take these sequences and learn partial orders from the ordering of events in them. Finally we build an interactive visualization tool that enables one to explore the partial orders related to a subset of diagnostic codes and make sense out them. We also present an importance score for the partial-orders based on a statistical significance test. This allows the user to focus on the more surprising patterns in the data before others.

The main contribution of this work is a new class of partial order models known as augmented partial orders which can encode more structure in the data than general partial orders. This model class uses a PQ-tree data structure to represent the nested ordering information. We found this representation to be very useful in summarizing different classes of patterns seen in electronic medical records. We validate out findings with the help of a practicing physician.

Topic 4: Streaming pattern mining for model learning

Most of the standard data mining techniques and probabilistic model learning algorithms require multiple passes through the data. Today, vast amounts of data are being generated in every application domain in streaming fashion. This makes it increasingly difficult to store and process data at speeds comparable to their generation. We investigate the idea of mining repeating patterns in infinite streams where it is not possible to make multiple passes through the data. Several streaming algorithms have been proposed in the literature for mining frequent itemsets but there is a dearth of methods for mining temporal patterns such as episodes or motifs.

In this topic, we design methods for efficiently mining temporal patterns in streaming data sequences and explore the trade offs underlying memory and runtime between several variants of the proposed algorithm. We propose a new class of frequent episodes that allow the episode mining algorithm to provide theoretical guarantees about the discovered episodes. We formulate the problem in the context of mining k-most frequent episodes in a recent

window over the data. The algorithm is designed to satisfy strict memory and runtime constraints. We extensively evaluate our method on synthetic and real datasets.

In Table 1.1, we summarize the different aspects involved in each research topic in terms of the type of data, the probabilistic model, the pattern class, and the application domains.

Table 1.1: Aspects of the proposed research tasks.

Topic	Type of Data	Pattern Class	Temporal Model	Application
Topic 1	Continuous multivariate time series	Motifs/Episodes	Dynamic Bayesian networks	Data center chiller modeling
Topic 2	Discrete-event sequences	Episodes with fixed delays	Excitatory dynamic networks	Neuroscience
Topic 3	Sequential Databases	Parallel Episodes	Augmented Partial Orders	Medical Informatics
Topic 4	Streaming Event Sequences	Episodes	-	Neuroscience & Sensor networks

1.4 Organization of the dissertation

In Chapter 2, we address the problem of motif mining in time series data. Here we present the problem of discovering frequently repeating subsequences or motifs in multi-variate time-series data in more detail and propose an efficient and robust technique for mining such motifs. We present a real world application of the motif mining framework in analysis of the performance of the cooling infrastructure of a HP datacenter. Further, we extend the time-series analysis framework to learn dynamic Bayesian networks from enriched representations of sensor data using motifs. We introduce the notion of internal states and represent the system dynamics as transitions between these states. In addition we use the probabilistic model learnt from data to reason about the state transitions.

In Chapter 3, we consider the problem of identifying new classes of models that can be learnt from frequent temporal patterns, especially episodes; and present the formulation of a new class of models called 'Excitatory Dynamic Networks' (EDNs). This is a specialized class of dynamic Bayesian networks that can be learnt efficiently from frequent episodes discovered in the data. The application studied here is from neuroscience and we demonstrate the usefulness of the proposed framework using both synthetic and real neuronal network datasets.

Next we present a framework for summarizing frequent patterns in electronic medical records in Chapter 4. We provide the details of a three step procedure for mining augmented partial orders in this data, and demonstrate the usefulness of our technique on a large medical records database.

Finally, in Chapter 5 we present the problem of discovering frequent episodes in the streaming context. This is cross-cutting problem applicable to many domains including the ones presented for the first three topics. We propose and evaluate several techniques for discovering top-k most frequent episodes from an event stream, and demonstrate our method on real datasets from neuroscience.

Chapter 6 summarizes our experience with temporal pattern mining and learning probabilistic model. We discuss the unique aspects involved in each of the problem presented in this dissertation. We hope the reader is able to take away a new and deeper understanding of modeling temporal processes in the light of frequent episode mining.

Chapter 2

Sustainability characterization of data centers using motifs

Modern IT infrastructure is ubiquitous, especially in the services sector that requires 'always-on' capability. Practically every large IT organization hosts data centers, operated either in-house or outsourced to major vendors. Over the last decade, data centers have grown from housing a few hundred multiprocessor systems to tens of thousands of individual servers today. This growth has been accompanied with steep increases in power density, resulting in higher heat dissipation, and thus increasing both power and cooling costs. According to the EPA, US data centers have become energy hogs and their continued growth is expected to demand the construction of 10 new power plants by 2011 [46, 47, 48]. One news report [49], perhaps alarmist, claims that a single web search query can use upto half the equivalent energy of boiling a kettle of water! Globally, data centers currently consume 1–2% of the world's electricity [50] and are already responsible for more CO_2 emissions than entire countries such as Argentina or Netherlands. If these trends hold, data center emissions are expected to quadruple by 2020 [48] and some estimates expect the carbon footprint of cloud computing to surpass aviation [46].

Data centers constitute a mix of computing elements, networking infrastructure, storage systems along with power management and cooling capabilities [51], all of which contribute to energy inefficiency. A plethora of approaches are hence available to curtail energy usage in each of the different data center subsystems and achieve sustainable data centers. For instance, huge inefficiencies abound in average server usage (believed to be in the single digits -to- at most 10–15%), and thus one approach to achieve greener IT is to use virtualization and migration to automatically provision new systems as demand spikes and consolidate applications when demand falls. A lot of prior work has focused on a data center's cooling infrastructure, which consumes anywhere between 30% to 50% of the total power consumption of a data center. Within the cooling infrastructure, most of the energy is spent on chillers, which refrigerate the coolant, typically water, used to extract heat from the equipment in the

data center. Dynamic management of an ensemble of chiller units [52] in response to varying load characteristics is an effective strategy to make a data center more energy-efficient. There are even end-to-end methodologies proposed [53] that track inefficiencies at all levels of the IT infrastructure "stack" and derive overall measures of the efficiency of energy flow during data center operation.

A key problem is the unavailability, inadequacy, or infeasibility of theoretical models or "first principles" methodologies to optimize design and usage of data centers. Admittedly, some components of data centers can be readily modeled (e.g., an operating curve for an individual chiller unit, a CFD prediction of airflows through rows of racks for static conditions) but the applicability of these models is severely limited due to simplifying assumptions and excessive computational time. Consequently, data-driven approaches to data center management have become more attractive. By mining sensor streams from an installation, we can obtain a real-time perspective into system behavior and identify strategies to improve efficiency metrics.

In this chapter, we focus primarily on the cooling infrastructure of a data center, especially chillers. Chillers are a key ingredient to keeping data centers functioning; as a case in point, recently, the music service Last.fm had to be shut down due to overheating in its data center. We present CAMAS (Chiller Advisory and MAnagement System) – a temporal data mining solution to mine and manage chiller installations. CAMAS embodies a set of algorithms for processing multivariate time series data and characterizes sustainability measures of the patterns mined. Thus, we show how temporal data mining can bridge the gap between low-level, raw, sensor streams, and the high-level operating regions and features needed for an operator to efficiently manage the data center.

The design and implementation of CAMAS makes the following contributions:

1. We demonstrate an efficient approach to mine motifs in multivariate time series data and which can be used for sustainability characterization. Our algorithm can accommodate 'don't care' states in its definition of motifs and this enables us to uncover expressive patterns in multivariate data.

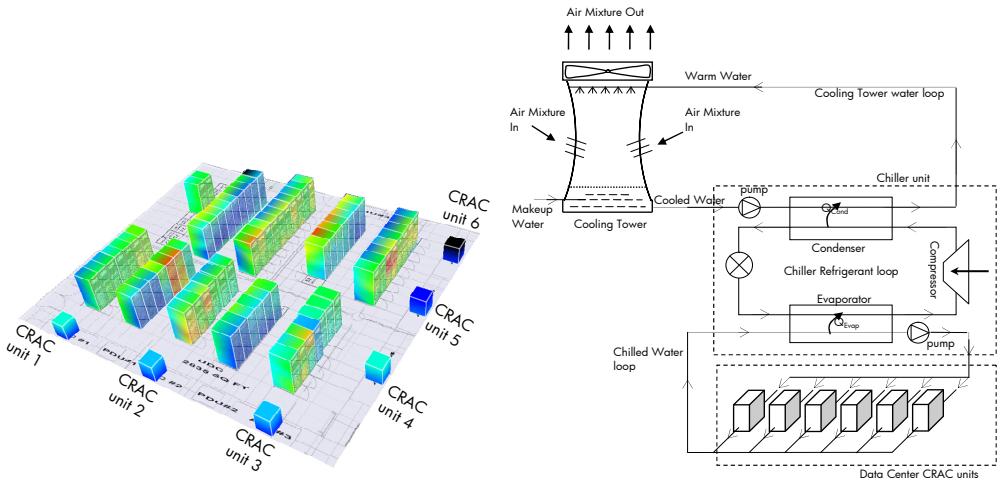
2. We describe how simple association analysis can be utilized to identify inefficient regions of chiller ensemble operation and how protocols for improving the overall efficiency of the system can be readily derived from the results of such associations.

3. We describe how we can construct a complete dynamic Bayesian network (DBN) that both captures the operation of the chiller ensemble and also enables reasoning in what-if and diagnostic scenarios.

2.1 Background

We present below some background about data center chillers and their chiller installations with a view toward motivating the underlying operational problems that can be solved using data mining techniques.

2.1.1 Architecture of a data center



(a) (b)

Figure 2.1: Data center schematic. (a) Thermal map of a data center showing racks arranged in rows and CRAC units, and (b) Typical cooling infrastructure of a data center.

Figure 2.1(a) shows a data center consisting of IT equipment (servers, storage, networking) fitted in racks arranged as rows. A large data center could contain thousands of racks occupying several tens of thousands of square feet of space. Also shown in the figure are computer room air conditioning (CRAC) units that cool the exhaust hot air from the IT racks. Energy consumption in data center cooling comprises work done to distribute the cool air and to extract heat from the hot exhaust air. A refrigerated or chilled water cooling coil in a CRAC unit extracts the heat from the air and cools it within a range of 10°C to 18°C. The cooling infrastructure of a data center is shown in Figure 2.1(b).

Key elements of this infrastructure include CRAC units, plumbing and pumps for chilled water distribution, chiller units and cooling towers. Heat dissipated from IT equipment is extracted by CRAC units and transferred to the chilled water distribution system. Chillers extract heat from the chilled water system and reject it to the environment through cooling towers or heat exchangers. In addition to the IT equipment, the data center cooling infrastructure can account for up to 50% of the total power demand [54]. The CRAC units

provide two actuators that can be controlled. The variable frequency drive (VFD) controls the blower speed and the chilled water value regulates the amount of chilled water flowing into a unit (between 0% and 100%). These built-in flexibilities allow the units to be adjusted according to the workload demand in the data center. The demand is detected via temperature sensors installed on the racks throughout a data center.

2.1.2 Data center chillers

As stated earlier, the focus of this work is on chiller units that receive warm water (at temperature, T_{in}) from the CRAC units, extract heat from it and recirculate the chilled water (at temperature, T_{out}) back to the CRAC units.

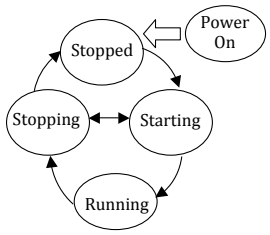


Figure 2.2: Operational state diagram of a chiller unit.

Each chiller is composed of four basic components, namely, evaporator, multi-stage centrifugal compressor, economizer and water-cooled or air-cooled condenser. Liquid refrigerant is distributed along the length of the evaporator to absorb enough heat from the water returning from the data center and circulated through the evaporator tubes to vaporize. The gaseous refrigerant is then drawn into the first stage of the compressor. Compressed gas passes from the multi-stage compressor into the condenser. Cooling tower water circulated through the condenser tubes absorbs heat from the refrigerant, causing it to condense. The liquid refrigerant then passes through an orifice plate into the economizer. Flashed gases enter the compressor while the liquid flows into the evaporator to complete the circuit.

Starting and stopping a chiller is a complex, multi-step process. Fig. 2.2 shows the operational state diagram of a typical chiller. On power-on, the chiller waits for the compressors to start, after a prescribed delay. On startup, the chiller utilization varies to match the cooling load. Based on chiller technology, chiller compressors can throttle in discrete stages or continuously. Feedback control is used to maintain the outlet temperature, T_{out}, close to a user-specified set-point temperature.

We define some terms used in the context of a data center chiller unit.

- **IT cooling load.** This is the amount of heat that is generated (and thus needs to be dissipated) at a data center. It is approximately equivalent to the power consumed by the equipment since almost all of it is dissipated as heat. It is commonly specified in kilowatts (KW).

- **COP.** The coefficient of performance (COP) of a chiller unit indicates how efficiently the unit provides cooling, and, is defined as the ratio between the cooling provided and the power consumed, i.e.,

$$\text{COP}_i = \frac{L_i}{P_i} \tag{2.1}$$

  where L_i is the cooling load on the ith chiller unit and P_i is the power consumed by it. In the data center studied in this work the typical values of COP for air-cooled chillers and water-cooled chillers are 3.5 and 6.5 respectively.

- **Chiller utilization.** This is the percentage of the total capacity of a chiller unit that is in use. It depends on a variety of factors, mainly, the mass flow rate of water that passes through a chiller and the degree of cooling provided, that is, the difference between the inlet and outlet temperatures ($T_{in} - T_{out}$). For a particular T_{out}, an administrator can control the utilization at a chiller through power capping or by changing the mass flow rate of water. The air-cooled chiller are operated in swing-mode to handle rapidly changing cooling load and their utilizations typically vary in the range 20-80%. On the other hand the water-cooled chillers are operated at high utilization $\approx$ 80% and handle the base cooling load.

- **Chiller power consumption.** This is simply the power consumed by a chiller unit. Although power meters that measure aggregate power consumption of data center infrastructure elements are usually available, meters that measure power consumed by an individual entity or a specific group (e.g. chillers) may not always be installed. In such cases, if the capacity of the unit and average COP are known, they, together with unit utilization, can be used to estimate power consumed.

$$P_i = \frac{U_i * C_i}{100 * \text{COP}_i} \tag{2.2}$$

  where P_i is the power consumed, U_i the utilization, and C_i the capacity, all pertaining to the i^{th} chiller unit.

2.1.3 Ensembles of chiller units

The number of chiller units required depends on the size and thermal density of a data center. While one unit may be sufficient for a small data center, several units operating as an ensemble may be required to satisfy the cooling demand of a large data center. Figure 2.3 shows an ensemble of chiller units that collectively provide cooling for a data center.

Out of the five units shown, three are air-cooled while the remaining two are water-cooled. Also, to provide a highly available data center and ensure business continuity, sufficient spare capacity is usually provisioned to meet the cooling demand in the event of one or more units becoming unavailable as a result of failure or required maintenance.

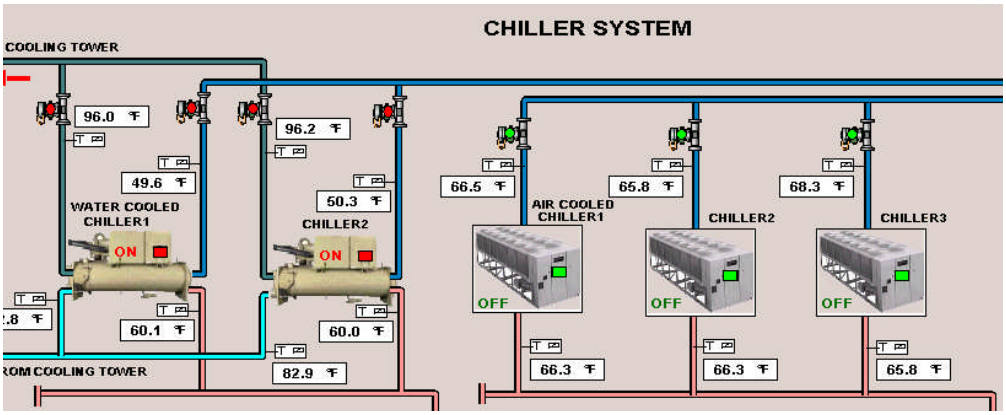


Figure 2.3: Ensemble of five chiller units working in tandem to provide cooling for a large data center.

Although operating curves for individual chiller units exist, no model is available for operation of an ensemble, especially one consisting of heterogeneous units. Additionally, shift and/or drift of response characteristics with time further complicate their management. The operational goals are to satisfy the cooling requirements while minimizing the total power consumption of the ensemble and maximizing the average lifespan of the units. While multiple factors impact the lifespan of a chiller unit, an important one is: rapid and large oscillations in utilization value. High amplitude and frequent variations in utilization due to varying load or some failure condition result in decreased lifespan, and, thus, need to be minimized.

2.1.4 Chiller management issues

There are several issues that lead to inefficient chiller operation and these are some of the topics that motivate the design of CAMAS:

- **Short cycling:** Frequent start and stop cycles lead to fatigue of mechanical parts due to high torque requirements, and, deterioration of electrical circuitry due to high inrush current. Moreover, load fluctuations due to cycling can also lead to drop in power factor and potential penalties from the utility. In case of data centers with on-site generation, such fluctuations can lead to reliability issues at the generators as well. Downstream of chillers, pump performance and cooling tower efficiency can also

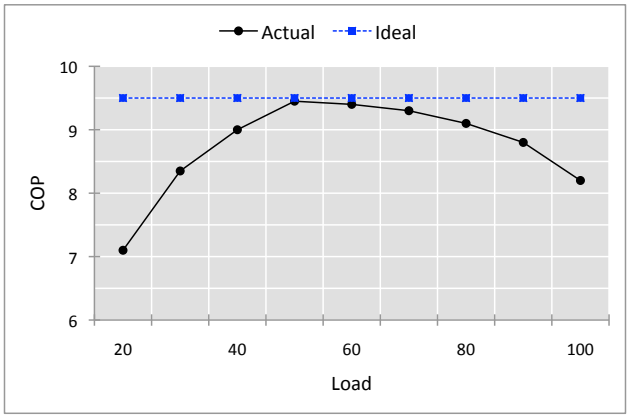


Figure 2.4: Variation of chiller efficiency (COP) with utilization (load).

be adversely affected. Typically chillers have an MTBF (mean time between failure) of 20,000 hours or more, which can reduce exponentially due to oscillations.

- **COP dependence on utilization:** Chillers show poor energy efficiency at low and high utilizations. Figure 2.4 shows a typical variation of efficiency (in terms of coefficient of performance (COP), see Eq. 2.1) with utilization. These curves depend not only on the type of chiller but also on external factors, such as, ambient temperature, supply temperature of the coolant, etc. Further, these curves may even shift with time over the typical 15 to 20 years lifespan of a chiller.

- **Complex and unknown dependencies between external variables and performance:** The performance of an ensemble of chillers depends on many factors, several of which are not explicitly monitored. For example, it depends of ambient temperature and humidity. Further, each installation of chillers is slightly different from other installations, requiring local domain experts to fine tune the performance. Further, these relationships may show a drift with time.

2.2 A framework for data mining in chiller installations

While administration of a single chiller unit is not complicated, configuring an ensemble of chillers for optimal performance is a challenging task, especially in the presence of a dynamically varying cooling load [55]. Typically, heuristics and rules-of-thumb are used to make decisions regarding:

- Which chiller unit(s) should be turned on/off, and when?

- What utilization range should a particular unit be operated at?

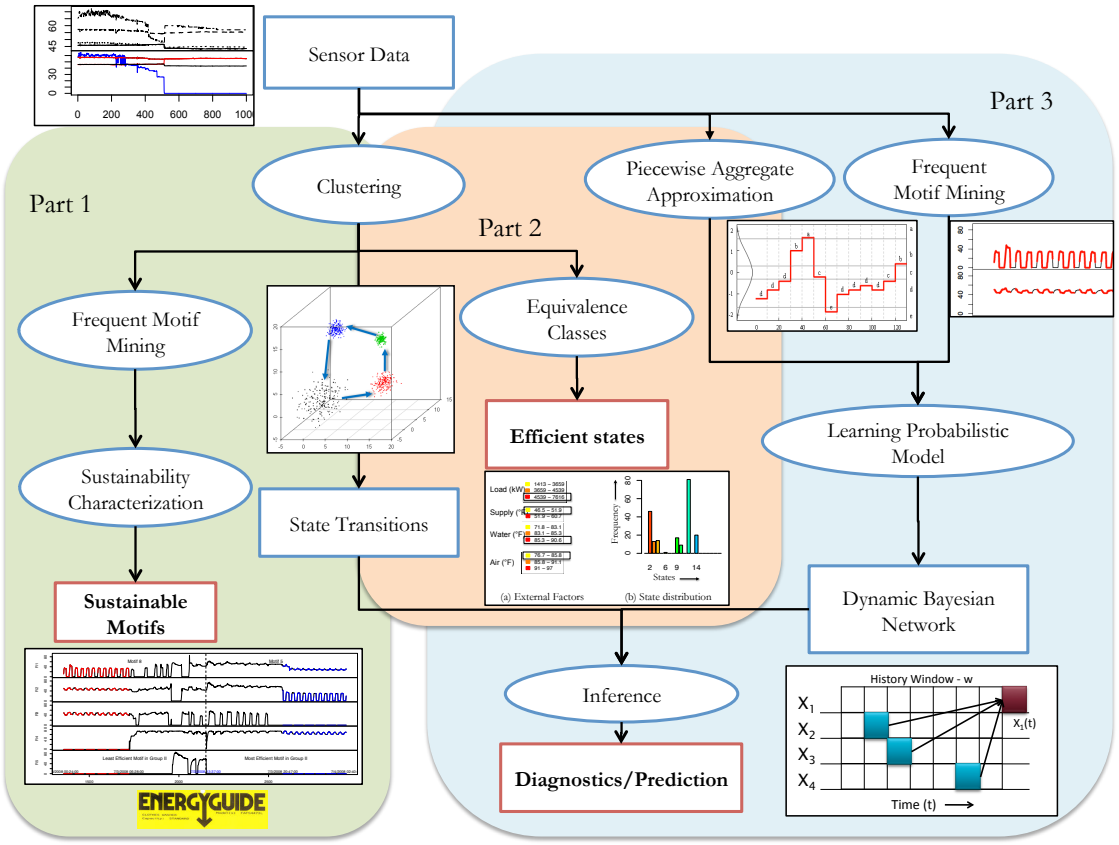


Figure 2.5: CAMAS framework. Three important building blocks are illustrated: (1) Motif discovery and sustainability characterization, (2) Modeling external factors and internal states, and (3) Probabilistic modeling of system variables.

- How should the ensemble react to an increase or decrease in cooling demand?

Any guidance regarding questions posed above while maintaining performance and optimizing the above stated goals will be invaluable to a data center facilities administrator.

The primary goal of CAMAS is to aid in precisely this objective. CAMAS links the multivariate, numeric, time series data streams gathered from chiller units to high level sustainability characterizations. As shown in Fig. 2.5, CAMAS performs such transduction at many levels. First, it transduces multivariate time series streams into symbolic event data from which frequent episodes are mined to yield motifs of chiller operation. We show how these motifs directly map to sustainability characterizations. In particular, we can describe states of chiller operations in terms of the motifs they exhibit and the efficiency regions they involve. Fig. 2.6 describes an example session with CAMAS in this objective. This constitutes Part 1 of the framework (see Fig. 2.5). In Part 2 of the framework, by taking external conditions into account, we demonstrate how we can use association analysis to find inefficient states of operation. In Part 3, we demonstrate how these ideas can be generalized further into a com-

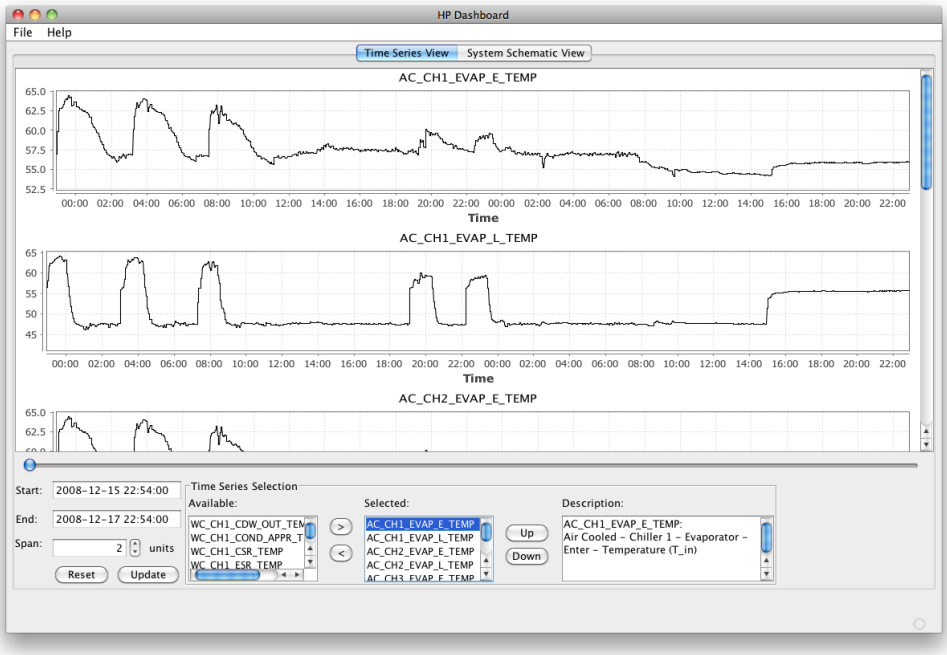


Figure 2.6: A screenshot of CAMAS used to analyze multivariate time series data.

plete framework for reasoning about chiller states and their transitions. We demonstrate a dynamic Bayesian network (DBN) approach to infer probabilistic relationships between chiller variables. Each of these three aspects are covered in the sections below.

2.3 Sustainability characterization using motifs

Motifs are repetitive patterns of occurrence in time series data. In understanding multivariate time series data about chiller utilizations, we seek to identify motifs that underly how different chillers are involved in meeting the varying demand posed by data centers. We would like to identify regions of time series progression that demonstrate better/improved sustainability measures than others.

2.3.1 Approach

Prior work in multivariate change point detection (e.g., see [56]) has posited statistical models for behavior within a segment and tracks changes in model parameters to denote qualitative change points. Our task is exacerbated by the lack of adequate models to characterize chiller behavior and also because of the varying interpretations that are attachable to multivariate data.

First, a motif or a trend can manifest in a single series or in multiple series. Second, even when it manifests in multiple series, the specificity with which it manifests can be fixed or variable. For instance, a motif can be 'three chiller units show oscillatory behavior' versus '*first three* chiller units show oscillatory behavior.' Since we seek to mine motifs so that their presence/absence can be cross-correlated with the chiller design (e.g., air cooled versus water cooled), we seek motifs of the latter form.

Definition 2.3.1. *A multivariate time series* $T = \langle \mathbf{v_1}, \ldots, \mathbf{v_m} \rangle$ *is an ordered set of real-valued vectors. Each real-valued vector* $\mathbf{v_i}$ *represents utilizations across all the chiller units.*

The above definition assumes a uniform sampling and hence the associated times are not explicitly recorded in the definition of T.

Definition 2.3.2. *A segment of the time series* T *is an ordered subset of consecutive vectors of* T *denoted by* $T[i, j] = \langle \mathbf{v_i}, \mathbf{v_{i+1}}, \ldots, \mathbf{v_j} \rangle$*, where* $1 \leq i < j \leq m$*.*

Definition 2.3.3. *A motif represents a set of segments of the time-series* T*,* $\{T[i_1, j_1]$*,* $T[i_2, j_2]$*,* $\ldots, T[i_n, j_n]\}$*, where* $1 \leq i_1 < j_1 \ldots \leq i_{k+1} < j_{k+1} \ldots j_n \leq m$*, such that any pair of segments in the set satisfy a similarity requirement and* $n \geq \theta$*, where* θ *is a user-defined minimum count.*

There are many possible instantiations of the similarity measure [44, 42], each leading to a specific formulation of a motif. We will describe later in this section the specific similarity requirement adopted here.

We decompose our overall goal into motif mining and sustainability characterization stages. Although a streaming algorithm would be more suitable in the context of time series data, the current implementation is intended more as a diagnostic tool than for prediction. We first transduce the continuous multivariate stream into a discrete symbol stream amenable for processing by episode mining algorithms. We perform a k-means clustering on these vectors and use the cluster labels as symbols to encode the time series. Observe that the multivariate series is now encoded as a single symbol sequence.

Already we have suitably transformed the multivariate numeric data to discrete symbols. We raise the level of abstraction further by doing a run-length encoding of the symbol sequence and noting where transitions from one symbol to another occur. This gives us a sequence of events for input to serial episode mining as illustrated below.

$$\begin{array}{rcl} \text{Symbol Sequence} & : & \text{d d d b a c c d d d d c b} \\ & & \Downarrow \\ \text{Event Sequence} & : & \langle (\text{d-b}, 4), (\text{b-a}, 5), (\text{a-c}, 6), (\text{c-d}, 8), \\ & & \quad (\text{d-c}, 12), (\text{c-b}, 13) \rangle \end{array}$$

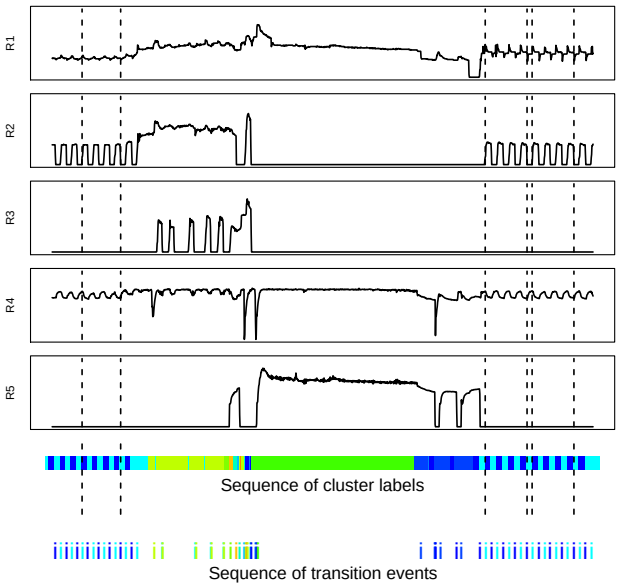


Figure 2.7: Illustration of change detection in multi-variate time series data.

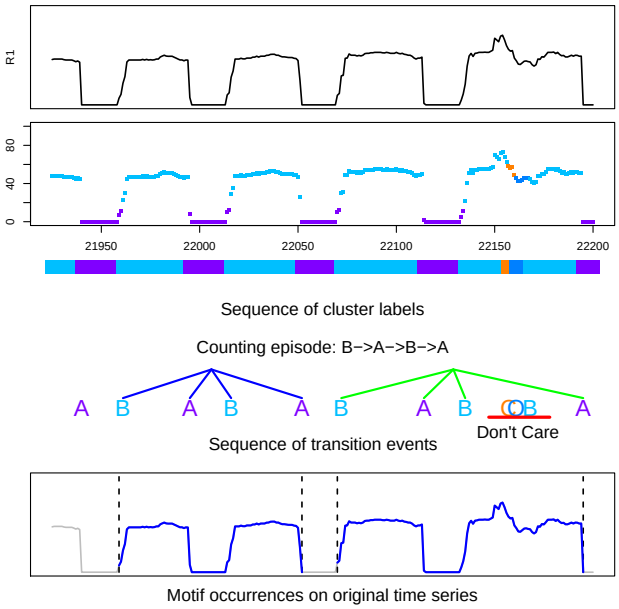


Figure 2.8: Illustration of motif mining in a single time-series using frequent episodes

Frequent episode mining is now conducted over this sequence of transitions. We adopt the framework of serial episodes with inter-event constraints. The structure of a serial episode α is given as:

$$\alpha = \langle E_1 \overset{(0,d_1]}{\rightarrow} E_2 \ldots \overset{(0,d_{n-1}]}{\rightarrow} E_n \rangle \tag{2.3}$$

Here $E_1, \ldots, E_n$ are the event-types participating in the episode α and, for our domain, these event types are cluster symbol indices. Note that a serial episode requires a total order among the events. Each pair of event-types in α is associated with an inter-event constraint. For example, the pair $E_1 \rightarrow E_2$ is associated with $(0, d_1]$ such that in an occurrence of α, event E_2 occurs no later than time d_1 after event E_1. Referring back to our definition of a motif, we see that the similarity requirement is thus every pair of segments must have an occurrence of the same serial episode defined over transition events. A key feature of episode mining is that the event occurrences can be interspersed with 'don't care' states and this enables us to uncover expressive patterns in the sequential data. While other approaches exist to accommodate don't care states [44], our algorithm is deterministic and guarantees finding repeating patterns in the discrete domain with a given support threshold. However some patterns in the raw time-series can go unnoticed due to coarse discretization.

2.3.2 Algorithms

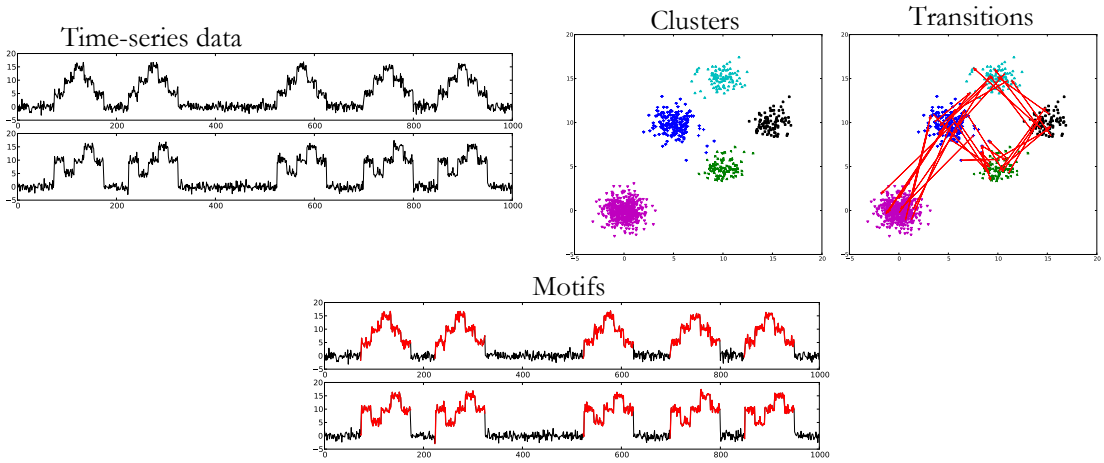
The mining process follows the level-wise procedure ala *Apriori*, i.e., candidate generation followed by counting. The candidate generation scheme is based on matching the $n-1$ size suffix of one n-node frequent episode with the $n-1$ size prefix of the another n-node frequent episode at a given level to generate candidates for the next level. The time complexity of the candidate generation process is $O(m^2 n)$, where n is the size of each frequent episode in the given level, m is the number of frequent episodes in that level, since all pairs of frequent episodes need to be compared for a prefix-suffix match.

The algorithm for counting the set of candidates episodes is given in Algorithm 1. The count or frequency measure is based on non-overlapped occurrences [17]. Two occurrences of an episode are said to be non-overlapped if the events in one occurrence appear between the events in the other occurrence. This notion most naturally eliminates the problem of trivial matches highlighted in [43] where a match is found between two slightly shifted segments of the time series. Algorithm 1 takes as input the event-sequence and a set of candidate episodes and returns the set of frequent episodes for a given frequency threshold θ. The algorithm counts the maximum number of non-overlapped occurrences of each episode with the inter-event time constraint $(0, T]$. This approach also allows repeated symbols or events in the episodes.

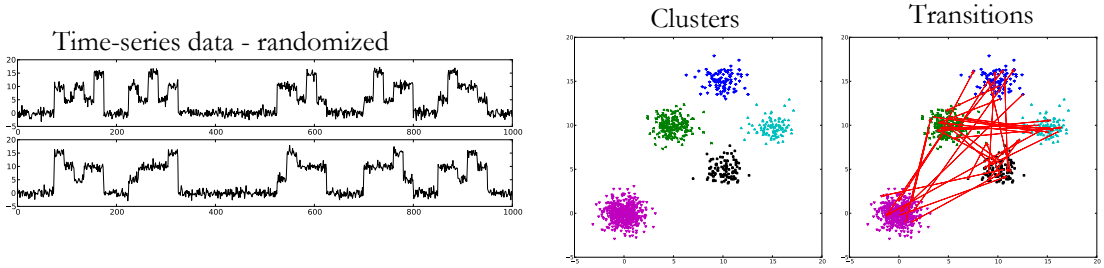
To illustrate how the overall multivariate motif mining approach works, we use some synthetic examples. In Figure 2.9(a), we demonstrate a multivariate time series consisting of two

components. A motif (corrupted by Gaussian noise) is embedded in this data. The motif comprises of a triangular waveform in the first series and simultaneously exhibits a 'H' shaped waveform in the second time series. This motif repeats five times in the given data.

First, the time information is stripped from the data and clustering is performed with number of clusters, $k = 5$. The cluster transitions are overlaid in the plot. These transitions are used to encode the multivariate numeric data into symbolic form and serve as the input to frequent episode mining. One episode is discovered which occurs five times and this episode is unpacked into the original data and overlaid on the dataset as shown. This serves as an example of a motif that we embedded and were able to uncover.



(a) Positive Example



No motifs were found

(b) Negative Example

Figure 2.9: Illustration of the proposed motif mining algorithm.

Next in Figure 2.9(b), we take the above time series data and randomize the time points so that the higher order information necessary to exhibit the motif is destroyed. The clusters still remain the same but there are no frequent episodes discovered in the data, and hence there are no motifs.

The worst case time complexity of the counting algorithm is given by $O(lnm)$, where l is the

number of events in the data sequence, m is number of candidate episodes and n is the size of the episode. The algorithm makes one pass of all the events in the event sequence and every time an event that belongs to an episode is seen, the data structure s for the episode is updated. A hash map is used to efficiently locate only a subset of relevant episodes for each event seen in the event sequence. Since our method allows repeated symbols, in the case of such episodes the same event can update s structure at most n times. Therefore if the level-wise growth of candidates is sufficiently arrested by a suitably chosen threshold, the algorithm scales linearly with data size.

Recall here that the events in mined frequent episodes correspond to transitions from one symbol to another. Our hypothesis here is that if motif occurrences are matched at transitions under an inter-transition gap constraint, then the corresponding time series subsequences will match under a suitable distance metric. In addition the episode mining framework allows for robustness to noise and scaling. The distance metric under which such motifs can be shown to be similar needs further investigation. Nonetheless this technique is found to be very effective in unearthing similar time series subsequences in real data sets.

2.3.3 Sustainability characterization

It is difficult (and subjective) to compare two motifs in terms of their sustainability impact by inspecting them visually. Therefore, it is necessary to quantify the sustainability of all motifs by computing a sustainability metric for them. This would enable quantitative comparisons between motifs; their categorization as 'good' or 'bad' from the sustainability metric point-of-view; and, furthermore, this information could be used to provide guidance to an administrator or a management system regarding the most 'sustainable' configurations of the chiller ensemble under a particular load. There are several sustainability metrics, such as, power consumed, carbon footprint, and exergy loss [57], where exergy is defined as the energy that is available to be used. Note that typically optimizing a sustainability metric, such as power consumed, also minimizes the total cost of operation.

We estimate two sustainability metrics for each motif: (1) the average COP of the motif; and, (2) a metric reflecting the frequency and amplitude of oscillations in utilization values. The average COP is calculated using Eqn. 2.1, where the load during motif i, averaged over all its occurrences, is used as L_i and the averaged power consumption of the motif as P_i. The power is estimated using Eqn. 2.2 with a averaged constant COP of 3.5 for the air-cooled and 6 for the water-cooled chillers. The COP of a motif quantifies the cooling effectiveness of the ensemble during that motif. In order to estimate the frequency of oscillations of a motif, we compute the number of mean-crossings, that is, the number of times the utilization crosses the mean value. This is very similar to number of zero-crossings that is commonly used in speech processing for estimation of frequency. This, together with standard deviation of a motif, allows oscillatory behavior to be compared.

Algorithm 1 Counting occurrences of serial episodes with inter-event time constraint $[0, T)$

Input: Candidate episodes $C = \{\alpha_1, \ldots, \alpha_m\}$, where $\alpha_i = E_{\alpha_i(1)} \rightarrow \ldots E_{\alpha_i(N)}$ is a N-node episode, Inter-event time constraint T and frequency threshold θ, Event sequence $S = \{(E_i, t_i)\}$.

Output: Frequent episodes $F : \alpha \in F$ if $\alpha.count \geq \theta$

```
/*Initialize*/
waits = φ
for all α ∈ C do
    α.count = 0
    s = Array of size N, each cell initialized to -∞
    for i = 1 to|α| do
        waits[E_α(i)].append(α, s, i)
for all (E_k, t_k) ∈ S do
    for all (α, s, i) ∈ waits[E_k] do
        if (i = 1) or (t_k − s[i − 1] ≤ T) then
            /*First event or Satisfies the time constraint*/
            if (i = |α|) then
                α.count = α.count + 1
                Reinitialize all elements of s to −∞
            else
                s[i] = t_k
Output F = {α : α ∈ C such that α.count ≥ θ}
```

2.3.4 Results

We applied our motif mining methodology to chiller data obtained from a large HP production data center covering 70,000 square feet with 2000 racks of IT equipment. Its cooling demand is met by an ensemble of five chiller units. The ensemble consists of two types of chillers – three are air-cooled and the remaining two are water-cooled. The data characteristics are given in Table 2.3.4. We mined 20 clusters from this data, with a view toward identifying well separated clusters.

Table 2.1: Parameters used in mining time-series motifs

Parameter	Value
Number of clusters	20
Support Threshold	10 or more occurrences
Inter-event gap lower-bound	5 min
Inter-event gap upper-bound	60 min

Cluster ID	Power (in tons)	Avg Util (in %)	Dev.	#Samples	Color
4	944.38	185.34	187.88	681	
14	859.10	168.60	204.35	699	
20	832.14	163.31	151.58	781	
18	829.87	162.86	151.09	772	
12	768.14	150.75	47.68	1485	
19	706.24	138.60	166.98	1382	
1	666.01	130.71	109.98	653	
5	659.68	129.47	115.78	1299	
8	597.20	117.20	40.71	3723	
6	583.26	114.47	143.87	667	
9	479.37	94.08	81.38	949	
16	464.30	91.12	164.41	562	
17	456.84	89.66	71.88	1504	
15	431.86	84.75	80.54	975	
11	395.65	77.65	215.82	1250	
7	348.92	68.48	59.97	2330	
3	326.92	64.16	58.03	4313	
2	325.93	63.97	50.96	2280	
13	312.77	61.38	164.51	970	
10	268.89	52.77	21.08	1388	

Figure 2.10: Characterization of operating states of the ensemble of chillers using clustering

Table 2.2: Dataset I for motif mining.

Number of data points	Date range	Number of Air-cooled Chillers	Number of Water-cooled Chillers
28,663 ($\approx$ 480 hours)	July 2, 2008 to July 7, 2008; Nov 27, 2008 to Nov 30,2008; Dec 16, 2008 to Dec 26, 2008	3	2

Motifs

All the parameters using in mining motifs in multivariate time series data is shown in Table 2.1. In all, 22 motifs were discovered in the chiller utilization data whose qualitative properties are summarized in Table 2.5 (more on this later). From a quantitative point of view these motifs can be clustered into groups based on load. One such group (Group II) is depicted in Table 2.3 with other quantitative measures. Although each group has very similar load levels, the COP within a group varies with the motifs. In Group II, for instance, motif 8 has a COP of 4.87, while motif 5 is significantly more efficient at a COP of 5.4. This information provides key insights to an administrator regarding motifs that are more energy efficient compared to others. Furthermore, this information could be codified into rules for setting chiller ensemble configuration based on the current load.

Table 2.4 shows the most efficient and the least efficient motifs for Group II based on power consumption, determined by the motif's COP value. Also shown are the potential power savings assuming that the least efficient motif could be transformed into the most efficient one.

Table 2.5 provides a qualitative description of all the motifs. They are grouped together manually based on similar utilization behavior of each of the chillers in the ensemble. Two factors are considered – the average utilization during the motif, and, the oscillatory behavior. The utilization level is labeled as low (L), medium (M) or high (H) based on the average utilization ranges of (0, 35), (35, 65) or (65, 100), respectively. The oscillatory behavior is marked (1) none (N) – which indicates steady values with minor variations; (2) small (S) – which indicates the values show oscillations that have a small amplitude; and finally, (3) large (L) – which indicates oscillations with a large amplitude. Furthermore, units that are not operating are marked 'OFF'.

Comparing motifs 5 and 8

We investigate now in more detail the differences between motifs 5 and 8 of Group II. While both motifs 5 and 8 have three chillers turned on, they are of different types. In motif 8, all three operating chillers (C1, C2 and C3) are air-cooled. In motif 5, two air-cooled (C1 and C2) and one water-cooled chiller (C4) are running. The qualitative behavior of the chillers in the two motifs are shown in Table 2.5. In motif 5, one chiller runs at high utilization (C4 at 66.5%), while the other two run at low utilizations (11.3% and 33.8%). In motif 8, one chiller runs at low utilization (17.63) while the other two operate at the medium range (49.1% and 44.3%). The amount of oscillatory behavior in the two motifs is about the same as indicated by the average rate of mean-crossings and standard deviation in Table 2.3. In both the motifs, one air-cooled chiller show large oscillations.

Economical incentives for sustainable operation

Business growth is an important factor in planning for investment in infrastructure. Chillers, of the kind described in this work, can cost upwards of $150k and typically, depreciate over 15-20 years. In the absence of intelligent management, the annual operating cost, even at 50% load, can be equally high. Maximizing utilization and efficiency is crucial to achieve a favorable return on investment. Operating chillers among favorable motifs that satisfy these conditions will enable IT administrators to sustain growth in business without major capital or maintenance costs. It will not only prolong the useful life of the existing equipment but postpone expensive retrofits and further infrastructure investments.

The cost savings by operating in favorable motif regimes can be quite easily characterized by multiplying the kW savings by $0.11 \times 24 \times 365$ to obtain the annual savings in dollars ($). Here, $0.11 is assumed to be the cost per KWh (kilo watt hour) of electricity. In this case switching from motif 8 to motif 5 gives us a nearly $40,000 in potential annual savings! Extrapolating this cost saving to other similar motifs gives us an idea of the utility of data mining algorithms in helping achieve cost effectiveness.

Carbon footprint calculation

Saving 1kWh of energy is equivalent to preventing release of 0.8 kg of carbon dioxide into the atmosphere. Based on the energy savings number, we can calculate the reduced carbon footprint.

$$\begin{aligned} \text{Saved Power} &= 41\text{kW} \\ \text{Saved Energy (annually)} &= 41\text{kW} \times 8760 \text{ hrs} \\ \text{Reduction in Carbon footprint} &= 41\text{kW} \times 8760 \times 0.8 \\ &= 287328 \text{ kg } CO_2 \end{aligned}$$

Observe that this is just the operational footprint; there is also an "embedded carbon footprint" of the chiller unit (as added in its manufacturing process). By maximizing operational life and utilization, we are managing this embedded carbon in the equipment as well. In other words, we are limiting the increase in embedded carbon in the environment while delivering the cooling required.

Table 2.3: Summary of the quantitative measures associated with motifs. Shown here are statistics for motif in one of the load groups.

Motif	Load (KW)	Power (KW)	COP	MC	Std	Span (min)	N
Group II							
8	**2028**	**417**	**4.87**	**.042**	**5.05**	**50**	**10**
2	2073	400	5.18	.048	4.28	113	17
5	**2076**	**384**	**5.4**	**.048**	**4.25**	**49**	**35**
4	2083	387	5.38	.048	4.31	82	23
6	2119	422	5.02	.044	4.08	51	23
3	2121	414	5.13	.042	4.31	119	11
1	2123	395	5.38	.046	4.39	172	10

MC – number of mean-crossings per minute
N – number of occurrences of the motif

2.4 Modeling the influence of external variables

While motifs are very useful in assessing repetitive patterns of occurrence, they constitute only a relative minority of the state space that the chiller ensemble operates in. Another interesting goal is to characterize the regions of state space occupied by the chiller ensemble and correlate them to the external conditions that the data center, and in particular, the

Table 2.4: The most and least efficient motif for each group and the potential power savings if the operational state of the chiller ensemble could be transformed from the least to the most efficient motif.

	Load (KW)		Most Efficient Motif	Least Efficient Motif	Potential Power Savings	
	Ave.	Std			KW	%
Group II	2089	35	5	8	41	9.83%

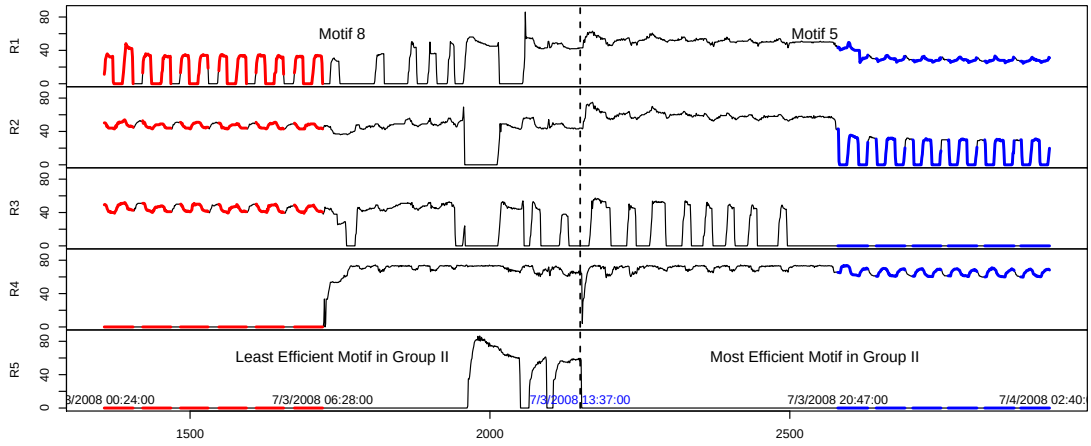


Figure 2.11: Two motifs with similar load but widely varying levels of efficiency. Observe how the least efficient motif transitions to the most efficient motif in the given time period.

chiller ensemble is subjected to. Once again, we focus on the utilization vectors of the same chiller system. But after a recent upgrade, the installation now has seven chillers instead of five as described earlier: five air-cooled and two water-cooled. We demonstrate the use of association analysis techniques to identify key conditions that underlie sources of inefficiency.

2.4.1 Approach

We discretize the utilization state-space of chillers by clustering using seed prototype vectors. The seed prototypes are obtained by combining low, medium and high utilization values for individual chiller units to obtain a total of $3^7 = 2187$ prototype utilization vectors. These prototypes are used to seed clusters for a k-means algorithm and clusters that are assigned zero members are removed from the analysis. The resulting clusters are still too many for human consumption; consequently, we applied hierarchical clustering with distance between pairs of clusters as the average distance between all pairs of points across the two clusters. The agglomerative process of merging clusters was stopped at a level where there was sufficient separation between the clusters being merged. This ensures that the states

Table 2.5: Summary of the qualitative behavior of all the motifs. The ones showing similar behavior are grouped together.

Motifs	Num of Units ON	Chiller behavior									
		C1		C2		C3		C4		C5	
		Util.	Oscil.	Util.	Oscil.	Util.	Oscil.	Util.	Oscil.	Util.	Oscil.
7, 15	2	M	L	OFF		H	S	OFF		OFF	
13	2	L	L	OFF		M	S	OFF		OFF	
1,2,4,5,9,10	3	L	N	L	L	OFF		H	N	OFF	
3,6,11,12	3	OFF		L	N	L	L	H	N	OFF	
8,14	3	L	L	M	N	M	S	OFF		OFF	
16	3	L	S	M	N	M	S	OFF		OFF	
17	4	M	N	M	L	OFF		H	N	H	S
18	4	M	N	M	S	L	L	H	N	OFF	
19	4	M	S	L	L	OFF		H	N	H	L
20	4	M	N	L	L	L	L	H	N	OFF	
21	4	OFF		M	S	L	L	H	N	H	L
22	4	OFF		H	N	M	L	H	N	H	S

Utilization (Util.) – Low (L), Medium (M), High (H)
Oscillations (Oscil.) – None (N), Small (S), Large (L)

represent qualitatively distinct regions. The prototypes for the final states are shown in Figure 2.12. Each state is annotated with the percentage fraction of total time spent by the system in that state, the average time spent there before moving into a different state, and the average power consumed. The states are colored using the rainbow palette based on the power consumed in a state, red being the highest power consuming state and blue, the lowest power state.

As a preliminary analysis, we can organize the states from Fig. 2.12 in many different ways and assess their characteristics. For example, we can group the states based on power consumption and investigate the fraction of time that is spent in, say, high power states. We can assess the states by asking where the system spends most of its time and determine the profile of these states in terms of the different chillers. This can be used to validate if the load balancing objectives of the system are being met. Finally, we can also investigate which chillers are on/off in different states. In states 1, 5 and 8 none of the water cooled chillers are running. This is unusual for this installation since the scheduling policy mandates the use of atleast one water-cooled chiller at all times to handle the base cooling load.

Given the same external conditions, the chiller ensemble is expected to exhibit similar power consumption characteristics. For example, for a given cooling load the chiller system should

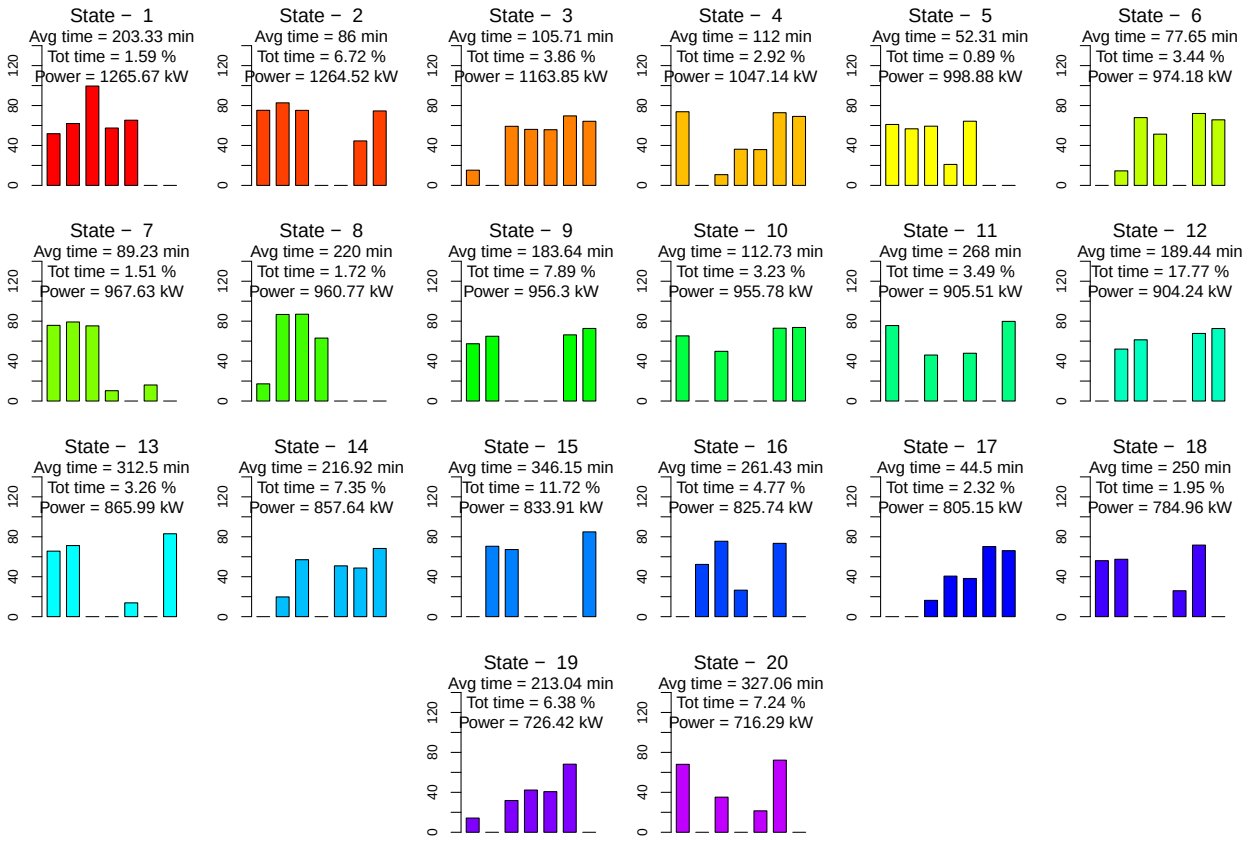


Figure 2.12: Prototype vectors of the 20 discrete states of the chiller ensemble. Each bar represents a particular chiller and its mean utilization level in a given state of the system. The first five bars in each plot indicate the air-cooled chillers and the last two indicate the water-cooled ones.

ideally consume the same amount of power. In practice, the system may not be able to operate in the most optimal mode to meet the required load under given external conditions. In CAMAS, we analyze the distribution of states observed under given external conditions and investigate its characteristics.

The external conditions known to influence the operations of chiller are cooling load, supply water temperature, ambient air temperature, and ambient water temperature. The cooling load is the total cooling requirement of the data center and office spaces that are served by the chiller ensemble. The supply water temperature is the temperature at which chiller water is supplied to the air-conditioning units. The set point of the supply water temperature impacts the coefficient of performance (COP) of the chillers. Similarly ambient air temperature and ambient water temperature (for water-cooled chillers only) affect COP. Once these external factors are held fixed, the distribution of states exhibited by the system exposes inefficiencies in operation; in particular, it can help identify high power states that can potentially be substituted by low power states (e.g. where fewer chillers are operating) to meet the same load under similar external conditions.

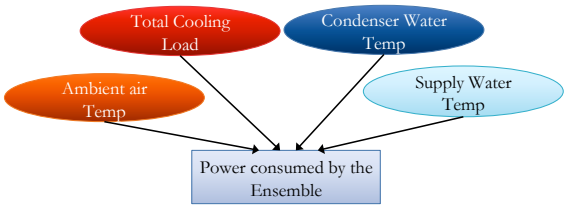


Figure 2.13: External factors affecting chiller performance.

2.4.2 Results

Table 2.6: Dataset II for modeling external factors.

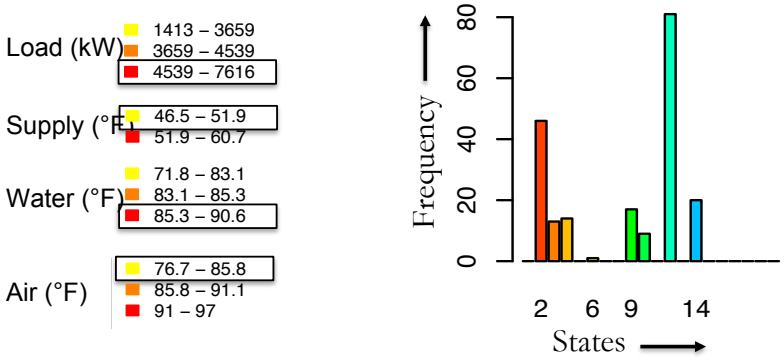
Number of data points	Date range	Number of Air-cooled Chillers	Number of Water-cooled Chillers
77,356	Apr 15, 2010 to July 14, 2010	5	2

The details of the data used in this analysis is given in Table 2.6. For the purposes of this analysis, we discretize the time-series for load, supply temperature, ambient water temperature and ambient air temperature into three, two, three, and three levels respectively (see Table 2.7). We use equal frequency bins for each level. Note that only two levels of discretization are used for supply temperature since the dynamic range of this variable is very narrow. Our hypothesis is that, under the same operating conditions, if we observe multiple states with widely varying power consumption profiles then this is an indicator of inefficiencies that exist in the system. It should be potentially possible to move the system

Table 2.7: Parameters used in analysis of the effect of external factors.

Parameter	Value
Number of clusters(discrete states)	20
Number of (equal-frequency) bins	
Load	3
Supply Temperature	2
Ambient-water Temperature	3
Ambient-air Temperature	3

from an inefficient high power state to a more efficient state to meet the same operation conditions.



(a) External Factors (b) State distribution

Figure 2.14: Frequency distribution of different states under a given external condition.

Consider Fig. 2.14 which depicts the distribution of states for a given setting of external conditions (i.e. load is high, supply temperature is low, ambient water temperature is high and ambient air temperature is low). Under these conditions, notice that State 2 (with average power consumption = 1256kW) and State 12 (with average power = 904kW) are observed with relatively high frequency. In this example if we are able to switch from State 2 to State 12, the energy savings over a three month period is roughly about 20,336 kWh. More examples of such skewed distributions are shown in Figure 2.15.

Furthermore, we can use the external factors as a handle to reason about how the system can be moved from its current state to a more desired state. For example, we can investigate if the system can be moved from State 2 to State 12, by changing some of the external conditions. The example in Fig. 2.16 shows such a scenario where load decreases with other factors remaining the same. We see that the high power states are no longer seen. Here the focus is on changes in state distribution given a change in external conditions.

These examples illustrate that there are numerous opportunities over the course of a chiller

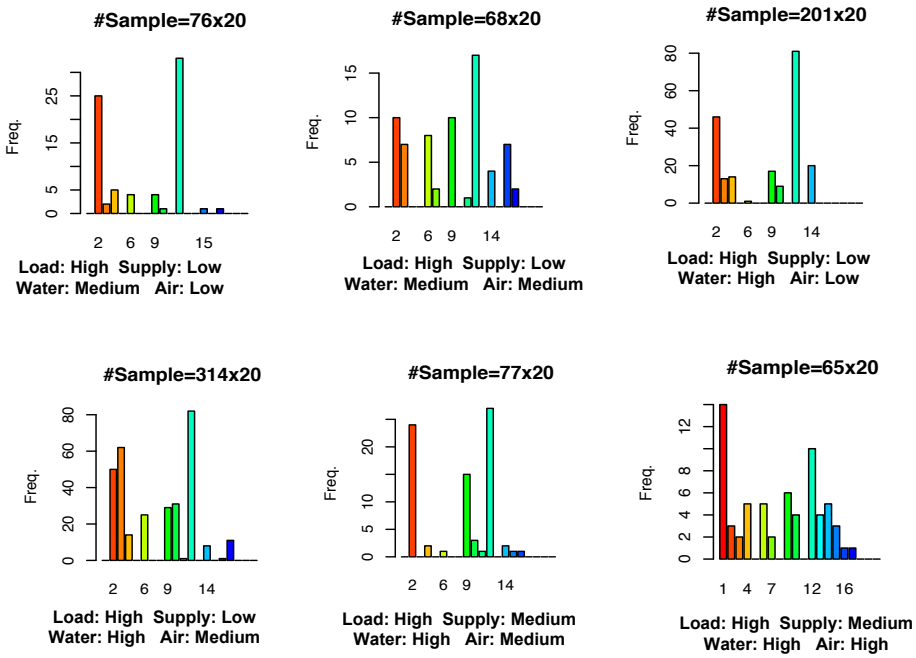


Figure 2.15: More examples of skewed state-distributions. It can be seen that there are high power states and low power states under similar external conditions.

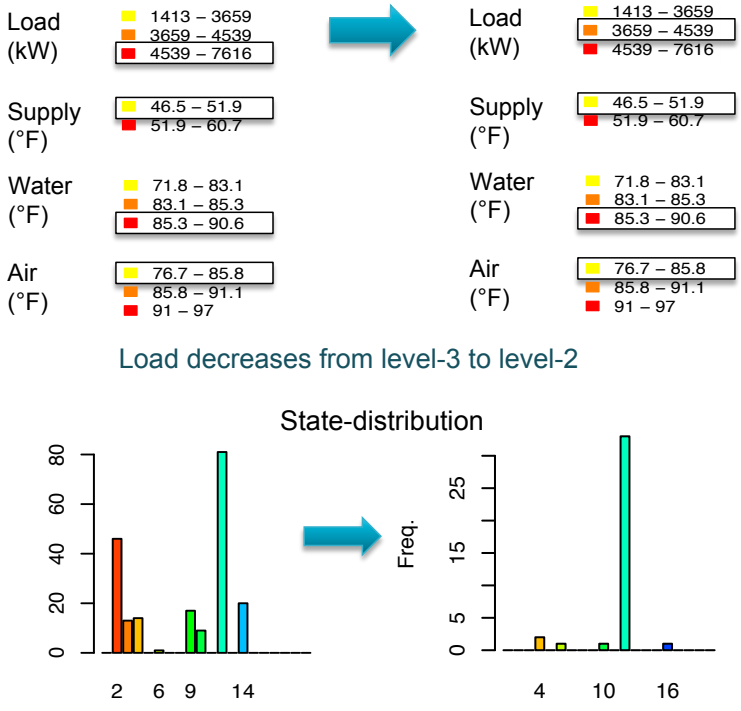


Figure 2.16: Change in state distribution with change in load.

ensemble operation where changes in modes of operation (states) would result in energy savings. Our next step is to use these ideas to develop a comprehensive framework for reasoning about chiller states and transitions.

2.5 Reasoning about states and transitions

In this section, we demonstrate the overarching framework of CAMAS to help monitor and analyze a chiller ensemble, while keeping the formulation general enough so that it could be extended to any similar infrastructure.

2.5.1 Approach

As shown in Fig. 2.5, data reduction is performed in a few different ways. The raw time series data is compressed using piece-wise aggregate approximation following discretization using equal frequency bins. This helps capture the average dynamics of the variables. A higher-order aspect of the time series involves repeating or oscillatory behavior. This is inferred by mining frequently repeating motifs or patterns in the time series, as described in Section 2.3. This information is integrated with the average behavior by recording the windows in which a time series exhibits motif patterns. Finally, it is also pertinent to mine relationships involving "control" actions that were taken during the operation of the chiller units. We focus on ON/OFF actions.

A graphical model in the form of a Dynamic Bayesian Network (DBN) is learnt from the above data. This model captures the dependencies between the variables in the system over different time lags. Here, we focus on the target variable of utilization and seek to identify suitable parents for modeling in the DBN. Unlike classical methods to learn BNs [58], we demonstrate a powerful approach to learn DBNs by creating bridges to the frequent episode mining literature [59]. To apply the learned DBN, we define states of the system by clustering together the combined utilization of the chiller units. This allows the operation of the chiller ensemble to be represented as a sequence of state transitions. We now use the dependencies and (conditional) independency relationships found in learning the graphical model to find the most probable explanation behind the state transitions. This framework can then be applied for activities like data center diagnostics, performance improvement, load balancing, and preventive maintenance.

As is well known, a Bayesian network (BN) is a graphical model denoted by $B = (G, P)$ where G is a directed acyclic graph (DAG) and P is a set of conditional probability distributions. The graph $G = (V, E)$ consists of a set of nodes V representing the random variables $\{X_1, \ldots, X_N\}$ in the system and a set of directed edges E. Each directed edge in E, denoted by $i \rightarrow j$, indicates that random variable X_i is a parent of random variable X_j. The conditional probabilities in P are used to capture statistical dependence relationships

between child nodes and parent nodes. In particular, given a random variable X_i in the graph, we denote by $\text{par}(X_i)$ the set of random variables that are parents of X_i . The statistical dependence between X_i and its parent nodes $\text{par}(X_i)$ is captured by the conditional probabilities $P(X_i|\text{par}(X_i)$.

To model a discrete-time random process $\mathbf{X}(t), t = 1, \ldots, T$; $\mathbf{X}(t) = [X_1(t)X_2(t) \cdots X_N(t)]$ as studied here, we use the more expressive formalism of dynamic Bayesian networks. In particular, we focus on time-bounded causal networks, where for a given $w > 0$, the nodes in $\text{par}(X_i(t))$, parents for the node, $X_i(t)$, belong to a w-length history window, $[t - w, t)$. Note that parent nodes cannot belong to the current time slice t for $X_i(t)$.

This assumption limits the range-of-influence of a random variable, $X_k(t)$, to variables within w time-slices of t and also indicates that the random variables $X_i(t)$ and $X_j(t)$ are conditionally independent given their corresponding parent sets in the history window. Further, we also assume that the underlying data generation model is stationary, so that joint-statistics can be estimated using contingency tables.

The learning of network structures involves learning the parent set, $\text{par}(X_i(t))$, for each $X_i(t)$, $i = 1, \ldots, N$. In this work we assume that there are no spurious independencies in the data, i.e., if a random variable $X_j(t - \tau)$ is a parent of $X_i(t)$, then the mutual information $I(X_i(t); X_j(t-\tau)|\mathcal{S})$ conditioned on a subset $\mathcal{S} \subseteq \text{par}(X)$ is always greater than zero. Moreover time-bounded causality enables us to learn the parents of each node $X_i(t)$ independent of any other node in the same time slice. We use a greedy approach to learn the parent set of each node $X_i(t)$. And proceed by adding a node which has the highest conditional mutual information to the parent set of $X_i(t)$ as shown in Eqn 2.4:

$$\text{par}^{i+1}(X_i(t)) \leftarrow \text{par}^{i}(X_i(t)) \cup \arg\max I(X_i(t); X_j(t-\tau)|par^{i}(X_i(t))) \tag{2.4}$$

The search is continued until the number of nodes in the parent set is k (where k is an user-defined parameter) or the conditional mutual information drops to zero. The structure learning is then followed by maximum likelihood estimation of the conditional probability tables.

2.5.2 State transition modeling

In analyzing complex systems it is usual to try and summarize the operating characteristics into a finite set of states where the systems spends most of its time. The utilization information of the chiller ensemble is important from both the aspect of efficiency and sustainability. Therefore we define the states of our system in terms of the combined utilization of all the chiller units.

In order to obtain a finite set of states, we fist perform a k-means clustering on the utilization vectors and use the cluster labels as symbols to encode the multi-variate time series of the

combined utilization. Thus the multivariate series of utilizations is now encoded as a single long state sequence.

Over the state sequence the points of interest are the times where the system moves from one state to another. The exact transition times can be affected by the clustering scheme used (e.g., number of clusters) but on the average they capture changes in the operating characteristics.

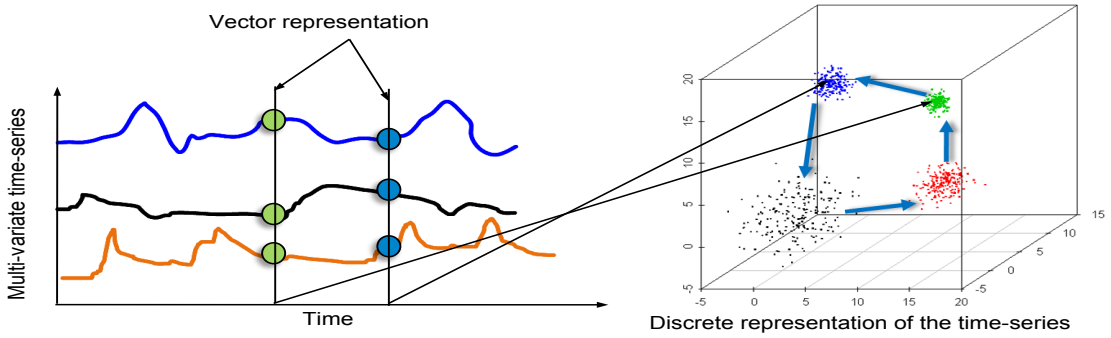


Figure 2.17: Illustration of rendering multivariate time series data as a sequence of state-transitions using clustering. The arrows are drawn between corresponding vectors on the time series plot and the 3-d plot.

An interesting question to ask now is: What causes the state transitions to take place. In the context of our modeling, this question translates to: what factors in the system cause the utilization of the chiller to go up or down? This can be answered by observing the changes that take place in the system around the state transitions. The problem with this approach is the lack of availability of sufficient data around the change points. We propose an alternative approach where we decompose the question by asking what causes each of the changes in the chiller units. From the graphical model already learnt we know the set of variables in the system that directly affect the utilization. These variables belong to the parent set of the utilization nodes. The task of finding the most probable causes for a transition amounts to evaluating the following probability distribution:

$$\Pr(\mathcal{S}) = \prod_{i \in \text{UtilizationVariables}} \Pr(\mathcal{S}_i | X_i(t) = a_i, X_i(t-1) = b_i) \tag{2.5}$$

where $\mathcal{S}_i = \text{par}(X_i(t)) \setminus X_i(t-1)$ and $\mathcal{S} = \cup \mathcal{S}_i$. The most likely values that $\mathcal{S}$ takes can be considered the best explanation of the transition. Here t is the time at which a state transition occurs, a_i, b_i are the discrete values the utilization variable takes before and after the state-transitions. These can be approximated by the cluster centers of each cluster used to define a state.

Table 2.8: Dataset III for state transition modeling.

Number of data points	Date range	Number of Air-cooled Chillers	Number of Water-cooled Chillers
28,289 ($\approx$ 576 hrs)	Oct 21, 2009 to Nov 13, 2009	3	2

Table 2.9: Parameters used in state transition modeling.

Parameter	Value
Number of clusters k (discrete states)	10
Window size (for piece-wise avg. aggregation)	15 min
Number of history windows (used in DBN learning)	5

2.5.3 Results

We applied our methods to chiller data obtained from a the same data center described in Section 2.3.4. All the parameters used in state transition modeling using DBNs are listed in Table 2.9.

During the period from October 21, 2009 to November 13, 2009 the cooling demand of the data center was met by an ensemble of five chiller units (see Table 2.8). The ensemble consists of two types of chillers: three air-cooled and the remaining two water-cooled. The data collected over this period totaled to over 576 hours of data consisting of over 47 variables.

Graphical model

For learning the graphical models, the raw time series data was aggregated over windows of 15 minutes and augmented with results from motif mining conducted over the same data. The motif information is in form of a binary valued sequence for each time series. The binary values indicate whether or not motif occurrences were seen in each 15 minute window. A finer grained representation of the motifs will be explored in the future.

Table 2.10: A few important system variables in the chiller ensemble data.

System Variable	Description
AC_CH(i)_EVAP_E_TEMP	Temperature of water entering air-cooled chiller i
MAIN_HEADER_TEMP_1_F	Water temperature at distribution header
WC_CH(i)_IN_TEMP	Temperature of water entering water-cooled chiller i
WC_CH(i)_OUT_TEMP	Temperature of water leaving water-cooled chiller i
AC_CH(i)_RLA	Percentage utilization of air-cooled chiller i
WC_CH(i)_RLA	Percentage utilization of water-cooled chiller i

Since utilization of a chiller mostly determines its energy consumption, we have shown that portion of the learned dynamic Bayesian network in Figure 2.18. Three air-cooled chiller utilization variables: AC_CH1_RLA, AC_CH2_RLA, AC_CH3_RLA; and two water-cooled ones: WC_CH1_RLA and WC_CH2_RLA are shown together with their parents and grandparents. In the DBN, almost every node has itself from the previous time slice as one of its parents, which is expected since the data is continuous valued time series data. In order to assist in creation of a causal Bayesian network, partial domain information was incorporated through constraints such as limiting nodes that could become parents. Three utilization variables show direct dependency on water temperature. This is, again, expected since the aim of the chiller ensemble is to maintain water temperature close to the set point.

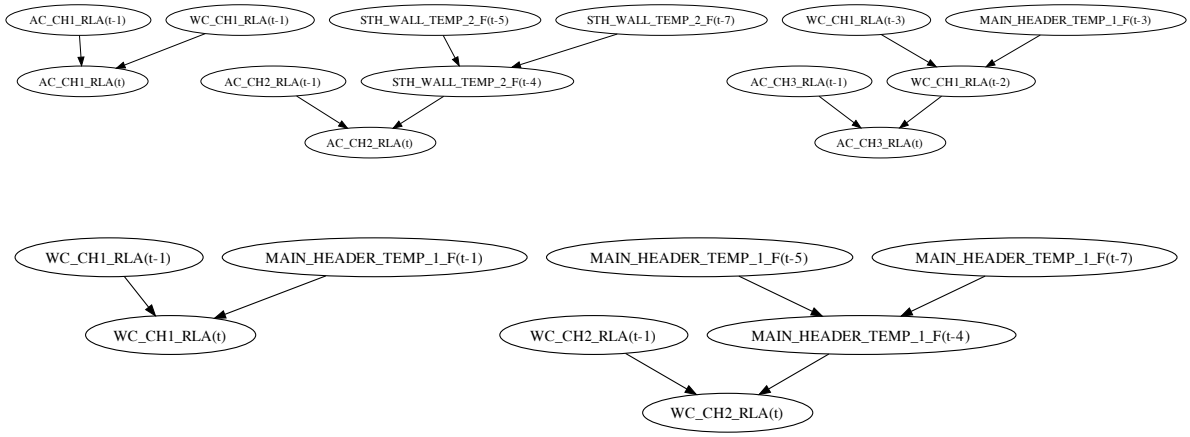


Figure 2.18: Graphical Model learned from the discretized time series data. Shown are the parent nodes for only the utilization variables of the five chiller units.

More interesting relationships are revealed by two air-cooled units (1 and 3) that directly depend on the utilization of a water cooled chiller. This would indicate that these two chillers ramp up or down based on how the water cooled chiller (which has higher capacity) is operating.

State Transitions

The operational states of the system, as shown in Figure 2.19, are obtained by clustering together the combined utilization of all the chillers in the ensemble. Here we use k-means clustering with $k = 10$ and organize the states in decreasing order of total power consumption. Since the objective of this work is to operate a chiller ensemble more energy efficiently, we color code the system states to reflect their power consumption. The color red indicates the maximum power consumption (about 3298 KW) while the color blue indicates the least consumption (2148 KW). Also shown, for each state, are the average utilization values of

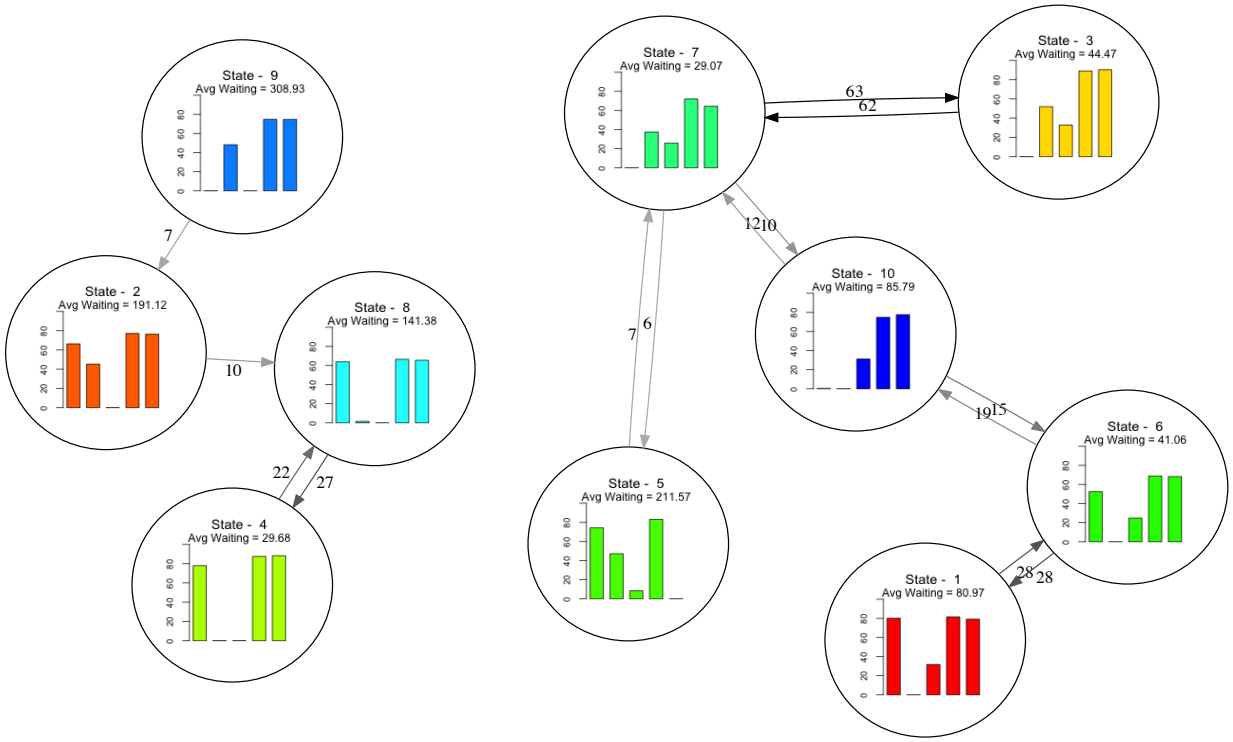


Figure 2.19: Transitions in the state-based representation of the combined utilization of the chiller ensemble. Edges representing 5 or fewer transitions are not shown. And edges representing more than 5 transitions are labeled by the actual counts.

the five chillers as a histogram with first three (starting from the left) being air-cooled ones and the last two being water-cooled units. The time spent by the system in each state is also listed (maximum in state 9, while least in state). The arrows show the transitions between states, with the gray-scale indicating the frequency of the transition (darkest implying most often).

The system states are mainly characterized by the number and kind of chiller units operating, their utilization levels and power consumption. Some states are quite similar, e.g., states 3 and 7, which have the same chiller units operating with not much difference in their utilization levels. Other states show marked difference, e.g., state 10 has three chillers operating (one air-cooled and two water-cooled) while state 6 has four working units (two air-cooled and two water-cooled). Note that consequently state 10 consumes less power than state 6. A data center chiller operator will be interested in understanding the variables that influence transitions from state 10 to 6.

Transition: State 10 → State 6

When the chiller ensemble makes a transition from state 10 to state 6, air cooled chiller-1 turns on. The graphical model can be queried to provide the most probable explanation for this change. Using the model from Section 2.5.3, we estimate values of parent of utilization node when these state transitions take place, as listed in Table 2.11. Note that utilization levels of the parent (WC_CH1_RLA) are high when these transitions take place. These and other similar insights would facilitate more energy efficient management of the chiller resources.

Table 2.11: List of most-likely value assignments of the parent-set of node AC_CH1_RLA i.e. utilization of air-cooled chiller 1.

$\text{par}(X_i(t))$	Delay	Value	$\Pr(\text{par}(X_i(t))\|X_i(t), X_i(t-1))$
WC_CH1_RLA	1	$(75.37, 77.38]$	0.27
WC_CH1_RLA	1	$(72.36, 75.37]$	0.18
WC_CH1_RLA	1	$(77.38, 79.71]$	0.18

The above models can be generalized to incorporate system alarms as nodes in the graphical model with the objective of discovering the variables that most significantly influence a particular alarm. This causal inference would provide valuable information to a data center chiller operator on corrective steps needed to handle the alarm. Furthermore, operator actions would be added to the model to enable discovery of dependencies between state transitions and such actions. This would allow an operator to query what actions could be taken (if any at all) to move the system from a less energy efficient state to a more efficient one.

2.6 Related work

2.6.1 Mining systems and installations

Many researchers have explored the use of data mining and machine learning to troubleshoot, analyze, and optimize computer systems and installations. There are a variety of projects with a diversity of foci, ranging from the mechanical equipment that power and cool the data center, to network-level diagnostics, to user-level applications and the system calls they make. For instance, modeling of rack-level temperature data specifically in relation to CRAC (computer room air conditioning) layout has been undertaken in [60, 61]. Optimization opportunities at multiple levels of smart center architecture have also been studied in [53]. More recent work [62] focuses on sensor data mining to identify anomalous and deviant behavior. Other related work includes the InteMon system from CMU [63, 64] that dynamically tracks correlations among multivariate time series [65], a characteristic of many

sensor streams. Interactive visualizations for system management have also been investigated [66]. Diagnosing network-level traffic, e.g., for volume anomalies, has been studied in [67]. The performance of Internet-scale applications deployed over multiple data centers is characterized using automatically mined signatures in [68]. Online failure prediction with applications to network security is discussed in [69]. Failure prediction using event logs has also been explored [70], especially in the context of IBM Blue Gene systems. Xu et al. [71] use text analysis and mining capabilities, in conjunction with some source code analysis, to study console logs from applications.

2.6.2 Approaches to model time series data

There are numerous formalisms available to model time series data and good surveys are in [72, 73]. One broad class of representations conducts global decompositions of the time series (e.g., PCA, DFT, wavelets) and aims to use only the most significant components to model the series. Alternatively, piecewise representations are easier to compute and also lend themselves to a streaming mode of operation. SAX [74] performs a piecewise aggregate approximation (the aggregate refers to the notion of modeling the given single time series by a linear combination of multiple time series, each expressed as a box basis function) and symbolize the resulting representation so that techniques from discrete algorithms can be adapted toward querying, matching, and mining the time series. In our work, we have aimed to identify appropriate representations of multivariate time series data that are efficient to compute as well as amenable to characterizing sustainability.

2.6.3 Finding motifs in time series

Motif mining is the task of finding approximately repeated subsequences in time series, and studied in various works, e.g., [44, 45, 43, 42]. Mining motifs in symbolized representations of the time series can drawn upon the rich body of literature in bioinformatics, where motifs have been used to characterized regulatory regions in the genome. As the work closest to ours, we explicitly focus on the SAX representation, which also provides some significant advantages for mining motifs. First, a random projection algorithm is used to hash segments of the original time-series into a map. If two segments are hashed into the same bucket, they are considered as candidate motifs. In a refinement step all candidate motif subsequences are compared using a distance metric to find the set of motifs with the highest number of non-trivial matches. A contrasting framework, referred to as frequent episode discovery, is an event based framework that is most applicable to symbolic data that is not uniformly sampled [17, 75, 8, 19]. This enables the introduction of junk, or "don't care" states, into the definition of what constitutes a frequent episode. We have shown how by mining motifs in change point data, we can identify suitable oscillatory patterns and how they can be related to sustainability metrics.

2.6.4 Dynamic Bayesian networks

As a class of probabilistic models, dynamic Bayesian networks (DBNs) are an outgrowth of Bayesian networks research, although specific forms of DBNs (e.g., HMMs) have a long and checkered history. Most examples of DBNs can be found in specific state space and dynamic modeling contexts [76]. In contrast to their static counterparts, exact and efficient inference for general classes of DBNs has not been studied well. Similar to our work, ref. [77] uses a (dynamic) Bayesian network model of a production plant to detect faults and anomalies. Unlike this work, however, we show how such networks can be efficiently learned using frequent episode mining over sensor streams. Furthermore, our networks are defined over clustered representations of the time series data rather than raw sensor signals.

2.7 Discussion

We have demonstrated a powerful approach to data mining for data centers that helps situate trends gathered from sensor streams in the context of sustainability metrics useful for the data center engineer. In particular, we have attempted to raise the level of abstraction with which a data center administrator interacts with sensor streams. In CAMAS, we have demonstrated the ability to i) compose event occurrences to recognize episodes of recurring behavior, ii) distinguish energy-efficient behavior patterns from others, and iii) identify and suggest opportunities for better sustainable operation.

Our future work objectives fall into two categories. First, we would like to use temporal data mining to uncover a complete model of the data center cooling infrastructure which can then be tuned/controlled/optimized, thus making data mining an integral part of the data center architecture. To achieve this, we need to model the transfer function in a way that encapsulates workload changes, manual steering of chiller operation, and other intermittent transients and recoverable faults. One approach is to develop a hidden state transition model (e.g., a hidden process model [78]) where the actions by an operator or other changes in the operational environment denote the hidden states (since these are often not recorded) and the observed transitions could be the emissions emitted by the probabilistic model. As our second objective, we would like to analyze more components of a data center, including the IT and power subsystems. Although our current studies have focused on the chiller subsystem, we posit that our methods are general and can be targeted toward these other complex system modeling tasks.

Chapter 3

Discovering excitatory relationships using dynamic Bayesian networks

Discrete event streams are prevalent in many applications, such as neuronal spike train analysis, physical plants, and human-computer interaction modeling. In all these domains, we are given occurrences of events of interest over a time course and the goal is to identify trends and behaviors that serve discriminatory or descriptive purposes. A Multi-Electrode Array (MEA) records spiking action potentials from an ensemble of neurons which after various pre-processing steps, yields a spike train dataset providing real-time and dynamic perspectives into brain function (see Fig. 3.1). Identifying sequences (e.g., cascades) of firing neurons, determining their characteristic delays, and reconstructing the functional connectivity of neuronal circuits are key problems of interest. This provides critical insights into the cellular activity recorded in the neuronal tissue. Similar motivations arise in other domains as well. In physical plants the discrete event stream denotes diagnostic and prognostic codes from stations in an assembly line and the goal is to uncover temporal connections between codes emitted from different stations. In human-computer interaction modeling, the event stream denotes actions taken by users over a period of time and the goal is to capture aspects such as user intent and interaction strategy by understanding causative chains of connections between actions.

Beyond uncovering structural patterns from discrete events, we seek to go further, and actually uncover a generative temporal process model for the data. In particular, our aim is to infer dynamic Bayesian networks (DBNs) which encode conditional independencies as well as temporal influences and which are also interpretable patterns in their own right. We focus exclusively on excitatory networks where the connections are stimulative rather than inhibitory in nature (e.g., 'event A stimulates the occurrence of event B 5ms later which goes on to stimulate event C 3ms beyond.') This constitutes a large class of networks with relevance in multiple domains, including neuroscience.

Probabilistic modeling of temporal data is a thriving area of research. The development

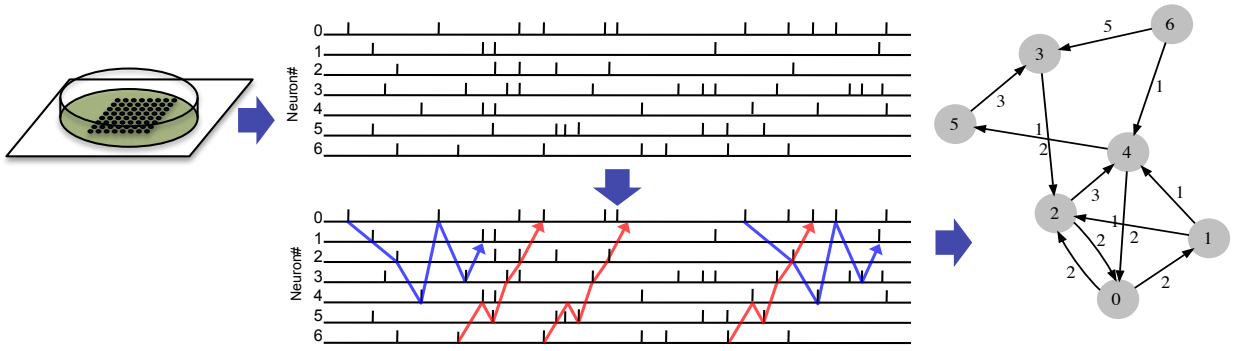


Figure 3.1: Event streams in neuroscience: A multi-electrode array (MEA; left) produces a spiking event stream of action potentials (middle top). Mining repeating firings cascades (middle bottom) in the event stream helps uncover excitatory circuits (right) in the data.

of dynamic Bayesian networks as a subsuming formulation to HMMs, Kalman filters, and other such dynamical models has promoted a succession of research aimed at capturing probabilistic dynamical behavior in complex systems. DBNs bring to modeling temporal data the key advantage that traditional Bayesian networks brought to modeling static data, i.e., the ability to use graph theory to capture probabilistic notions of independence and conditional independence. They are now widely used in bioinformatics, neuroscience, and linguistics applications.

A contrasting line of research in modeling and mining temporal data is the counting based literature in the KDD community [8, 17, 79]. Similar to frequent itemsets, the notion of frequent episodes is the object of interest here. Identifying frequency measures that support efficient counting procedures (just as support does for itemsets) has been shown to be crucial.

It is natural to question whether these two threads, with divergent origins, can be related to one another. Many researchers have explored this question. Pavlov, et. al used frequent itemsets to place constraints on the joint distribution of item random variables and thus aid in inference and query approximation [40]. In [41] probabilistic models are viewed as summarized representations of databases and demonstrate how to construct MRF (Markov random field) models from frequent itemsets. Closer to the topic of this work, [17] linked frequent episodes to learning Hidden Markov Models for the underlying data. Similar in scope to the above works, we present a unification of the goals of dynamic Bayesian network inference with that of frequent episode mining. Our motivations are not merely to establish theoretical results but also to inform the computational complexity of algorithms and spur faster algorithms targeted for specific domains.

Our main contributions are three-fold:

1. **New model class of excitatory networks**: Learning Bayesian networks (dynamic or not) is a hard problem and to obtain theoretical guarantees we typically have to

place restrictions on network structure, e.g., assume the BN has a tree structure as done in the Chow-Liu algorithm. Here, we place restrictions on the form of the conditional probability tables (CPT) instead of network structure; this leads to a tractable class of DBNs for inference, referred to here as *Excitatory Dynamic Networks (EDNs)*.

2. **New methods for learning DBNs**: We demonstrate that optimal EDNs can be learnt by creating bridges to frequent episode mining literature. In particular, the focus on EDNs allows us to relate frequent episodes to parent sets for nodes with high mutual information. This enables us to predominantly apply fast algorithms for episode mining, while relating them to probabilistic notions suitable for characterizing DBNs.
3. **New applications to spike train analysis**: We demonstrate a successful application of our methodologies to analyzing neuronal spike train data, both from mathematical models of spiking neurons and from real cortical tissue.

The chapter is organized as follows. Sec. 3.1 gives a brief overview of DBNs. Sec. 3.2 develops the formalism for learning optimal DBNs from event streams. Sec. 3.3 defines excitatory networks and presents the theoretical basis for efficiently learning such networks. Sec. 3.4 introduces fixed-delay episodes and relates frequencies of such episodes with marginal probabilities of a DBN. Our learning algorithm is presented in Sec. 3.5, experimental results in Sec. 3.6 and conclusions in Sec. 3.7.

3.1 Bayesian networks: Static and Dynamic

Formal mathematical notions are presented in the next section, but here we wish to provide some background context to past research in Bayesian networks (BNs). As is well known, BNs use directed acyclic graphs to encode probabilistic notions of conditional independence, such as that a node is conditionally independent of its non-descendants given its parents (for more details, see [27]). The joint pdf encoded by a BN can be succinctly written in product form, one term for each node in the network, where the term denotes the conditional probability of the node given its parents.

The earliest known work for learning BNs is the Chow-Liu algorithm [80]. It showed that, if we restricted the structure of the BN to a tree, then the optimal BN can be computed using a maximum spanning tree algorithm. In particular, the optimal BN is the tree that maximizes the sum of mutual information quantities between pairs of nodes. The Chow-Liu algorithm is noteworthy because it established a class of networks (trees) for which inference is tractable. More recent work, by Williamson [81], generalizes the Chow-Liu algorithm to show how (discrete) distributions can be approximated using the same general ingredients as the Chow-Liu approach, namely mutual information quantities between random variables. Meila [82] presents an accelerated algorithm that is targeted toward sparse datasets of high dimensionality.

More generally, however, BN learning must be approximate and we must settle for lack of optimality guarantees. Most methods provide good search heuristics [83]. There are two broad classes of algorithms for learning BNs: score-based and constraint-based. Score-based algorithms aim to search in the space of networks guided by a score function. Friedman's sparse candidate approach [35] refers to a general class of algorithms for approximate learning of BNs. The approach is iterative: in each iteration, a set of candidate parents are discovered for each node, and from these parents, a network maximizing the score function is discovered. Conditioned on this network, the algorithm then aims to add further nodes that will improve on the score. The approach terminates when no more nodes can be added or when a predetermined number of iterations are met. One instantiation of this approach uses the BIC (Bayesian Information Criterion) as the score function. The simulated annealing approach from [38] begins with an empty network and aims to add edges that can improve the score. With some small probability, edges that decrease the score are added as well. An underlying cooling schedule controls the annealing parameters. Constraint-based algorithms such as GSMN [39] on the other hand first aim to identify conditional independence relationships across all the variables and then aim to fit the discovered relationships into a network, much like fitting a jigsaw puzzle.

As a class of networks, DBNs are a relatively newer development although specific forms of DBNs (e.g., HMMs) have a long and checkered history. Most examples of DBNs can be found in specific state space and dynamic modeling contexts [76]. In contrast to their static counterparts, exact and efficient inference for general classes of DBNs has not been studied well. While many of the above algorithms for BNs can be adapted toward DBNs, our aim is to go further and exploit the specific modeling contexts that DBNs were designed for.

3.2 Learning optimal DBNs

Consider a finite alphabet, $\mathcal{E} = \{A_1, \ldots, A_M\}$, of event-types (or symbols). Let $s = \langle (E_1, \tau_1), (E_2, \tau_2), \ldots, (E_n, \tau_n) \rangle$ denote a data stream of n events over $\mathcal{E}$. Each $E_i, i = 1, \ldots, n$, is a symbol from $\mathcal{E}$. Each $\tau_i, i = 1, \ldots, n$, takes values from the set of positive integers. The events in s are ordered according to their times of occurrence, $\tau_{i+1} \geq \tau_i, i = 1, \ldots, (n-1)$. The time of occurrence of the last event in s, is denoted by $\tau_n = T$. We model the data stream, s, as a realization of a discrete-time random process $\mathbf{X}(t), t = 1, \ldots, T$; $\mathbf{X}(t) = [X_1(t)X_2(t)\cdots X_M(t)]'$, where $X_j(t)$ is an indicator variable for the occurrence of event type, $A_j \in \mathcal{E}$, at time t. Thus, for $j = 1, \ldots, M$ and $t = 1, \ldots, T$, we will have $X_j(t) = 1$ if $(A_j, t) \in s$, and $X_j(t) = 0$ otherwise. Each $X_j(t)$ is referred to as the *event-indicator* for event-type, A_j, at time t.

Example 3.2.1. *The following is an example event sequence of $n = 7$ events over an alphabet, $\mathcal{E} = \{A, B, C, \ldots, Z\}$, of $M = 26$ event-types:*

$$\langle (A, 2), (B, 3), (D, 3), (B, 5), (C, 9), (A, 10), (D, 12) \rangle \tag{3.1}$$

The maximum time tick is given by $T = 12$. *Each* $\mathbf{X}(t),\ t = 1, \ldots, 12$, *is a vector of* $M = 26$ *indicators. Since there are no events at time* $t = 0$ *in the example sequence (3.1), we have* $\mathbf{X}(1) = \mathbf{0}$. *At time* $t = 2$, *we will have* $\mathbf{X}(2) = [1000 \cdots 0]'$. *Similarly,* $\mathbf{X}(3) = [01010 \cdots 0]'$, *and so on.*

A DBN [84] is a DAG with nodes representing time-indexed random variables and arcs representing conditional dependency relationships. We model the random process $\mathbf{X}(t)$ (or equivalently, the event stream s), as the output of a DBN. Each event-indicator, $X_j(t)$, $t = 1, \ldots, T$ and $j = 1, \ldots M$, corresponds to a node in the network, and is assigned a set of parents, which is denoted as $\pi_j(t)$. A parent-child relationship is represented by an arc (from parent to child) in the DAG. In a DBN, nodes are conditionally independent of their non-descendants given their parents. The joint probability distribution of $\mathbf{X}(t)$ under the DBN model, can be factorized as a product of $P[X_j(t) \mid \pi_j(t)]$ for various j, t. In this work we restrict the class of DBNs using the following two constraints:

A1 *[Time-bounded causality]* For user-defined parameter, $W > 0$, the set, $\pi_j(t)$, of parents for the node, $X_j(t)$, is a subset of event-indicators out of the W-length history at time-tick, t, i.e. $\pi_j(t) \subseteq \{X_k(\tau) \ :\ 1 \leq k \leq M,\ (t - W) \leq \tau < t\}$.

A2 *[Translation invariance]* If $\pi_j(t) = \{X_{j_1}(t_1), \ldots, X_{j_\ell}(t_\ell)\}$ is an ℓ-size parent set of $X_j(t)$ for some $t > W$, then for any other $X_j(t')$, $t' > W$, its parent set, $\pi_j(t')$, is simply a time-shifted version of $\pi_j(t)$, and is given by $\pi_j(t') = \{X_{j_1}(t_1 + \delta), \ldots, X_{j_\ell}(t_\ell + \delta)\}$, where $\delta = (t' - t)$.

While **A1** limits the range-of-influence of a random variable, $X_k(\tau)$, to variables within W time-ticks of τ, **A2** is a structural constraint that allows parent-child relationships to depend only on relative (rather than absolute) time-stamps of random variables. Further, we also assume that the underlying data generation model is stationary, so that joint-statistics can be estimated using frequency counts of suitably defined temporal patterns in the data.

A3 *[Stationarity]* For every set of ℓ event-indicators, $X_{j_1}(t_1), \ldots, X_{j_\ell}(t_\ell)$, and for every time-shift δ (either positive or negative) we have $P[X_{j_1}(t_1), \ldots, X_{j_\ell}(t_\ell)] = P[X_{j_1}(t_1 + \delta), \ldots, X_{j_\ell}(t_\ell + \delta)]$.

The joint probability distribution, $Q[\cdot]$, under the Dynamic Bayesian Network model can be written as:

$$Q[\mathbf{X}(1), \ldots, \mathbf{X}(T)] = P[\mathbf{X}(1), \ldots, \mathbf{X}(W)] \times \prod_{t=W+1}^{T} \prod_{j=1}^{M} P[X_j(t) \mid \pi_j(t))] \tag{3.2}$$

Learning network structure involves learning the map, $\pi_j(t)$, for each $X_j(t)$, $j = 1, \ldots, M$ and $t > W$. Let $I[X_j(t)\ ;\ \pi_j(t)]$ denote the *mutual information* between $X_j(t)$ and its

parents, $\pi_j(t)$. DBN structure learning can be posed as a problem of approximating the data distribution, $P[\cdot]$, by a DBN distribution, $Q[\cdot]$. Let $D_{KL}(P||Q)$ denote the KL divergence between $P[\cdot]$ and $Q[\cdot]$. Using **A1**, **A2** and **A3** we now show that for parent sets with sufficiently high mutual information to $X_j(t)$, $D_{KL}(P||Q)$ will be concomitantly lower.

The Kullback-Leibler divergence between the underlying joint distribution, $P[\cdot]$, of the stochastic process, and the joint distribution, $Q[\cdot]$, under the DBN model is given by

$$D_{KL}(P||Q) = \sum_{\mathcal{A}} \left(P[\mathbf{X}(1), \ldots, \mathbf{X}(T)] \times \log \frac{P[\mathbf{X}(1), \ldots, \mathbf{X}(T)]}{Q[\mathbf{X}(1), \ldots, \mathbf{X}(T)]} \right) \tag{3.3}$$

where $\mathcal{A}$ represents the set of all possible assignments for the T M-length random vectors, $\{\mathbf{X}(1), \ldots, \mathbf{X}(T)\}$. Denoting the entropy of $P[\mathbf{X}(1), \ldots, \mathbf{X}(T)]$ by $H(P)$, the entropy of the marginal, $P[\mathbf{X}(1), \ldots, \mathbf{X}(W)]$, by $H(P_W)$, and substituting for $Q[\cdot]$ from Eq. (3.2), we get

$$\begin{aligned} D_{KL}(P||Q) = &- H(P) - H(P_W) \\ &- \sum_{\mathcal{A}} \left(P[\mathbf{X}(1), \ldots, \mathbf{X}(T)] \times \sum_{j=1}^{M} \sum_{t=W+1}^{T} \log P[X_j(t) \mid \pi_j(t)] \right) \end{aligned} \tag{3.4}$$

We now expand the conditional probabilities in Eq. (3.4) using Bayes rule, switch the order of summation and marginalize $P[\cdot]$ for each j, t. Denoting, for each j, t, the entropy of the marginal $P[X_j(t)]$ by $H(P_{j,t})$, the expression for KL divergence now becomes:

$$\begin{aligned} D_{KL}(P||Q) = &- H(P) - H(P_W) \\ &- \sum_{j=1}^{M} \sum_{t=W+1}^{T} H(P_{j,t}) - \sum_{j=1}^{M} \sum_{t=W+1}^{T} I[X_j(t)\,;\,\pi_j(t)] \end{aligned} \tag{3.5}$$

$I[X_j(t)\,;\,\pi_j(t)]$ denotes the *mutual information* between $X_j(t)$ and its parents, $\pi_j(t)$, and is given by

$$I[X_j(t)\,;\,\pi_j(t)] = \sum_{\mathcal{A}_{j,t}} \left(P[X_j(t), \pi_j(t)] \times \log \frac{P[X_j(t), \pi_j(t)]}{P[X_j(t)]\ P[\pi_j(t)]} \right) \tag{3.6}$$

where $\mathcal{A}_{j,t}$ represents the set of all possible assignments for the random variables, $\{X_j(t), \pi_j(t)\}$. Under the translation invariance constraint, **A2**, and the stationarity assumption, **A3**, we have $I[X_j(t)\,;\,\pi_j(t)] = I[X_j(t')\,;\,\pi_j(t')]$ for all $t > W$, $t' > W$. This gives us the following final expression for $D_{KL}(P||Q)$:

$$\begin{aligned} D_{KL}(P||Q) = &- H(P) - H(P_W) \\ &- \sum_{j=1}^{M} \sum_{t=W+1}^{T} H(P_{j,t}) - (T - W) \sum_{j=1}^{M} I[X_j(t)\,;\,\pi_j(t)] \end{aligned} \tag{3.7}$$

where t is any time-tick satisfying $(W < t \leq T)$. We note that in Eq. (3.7), the entropies, $H(P)$, $H(P_W)$ and $H(P_{j,t})$ are independent of the DBN structure (i.e. they do not depend on

the $\pi_j(t)$ maps). Since $(T-W) > 0$ and since $I[X_j(t)\,;\,\pi_j(t)] \geq 0$ always, the KL divergence, $D_{KL}(P||Q)$, is minimized when the sum of M mutual information terms in Eq. (3.7) is maximized. Further, from **A1** we know that all parent nodes of $X_j(t)$ have time-stamps strictly less than t, and hence, no choice of $\pi_j(t),\ j = 1, \ldots, M$ can result in a cycle in the network (in which case, the structure will not be a DAG, and in-turn, it will not represent a valid DBN). This ensures that, under the restriction of **A1**, the *optimal DBN structure* (namely, one that corresponds to a $Q[\cdot]$ that minimizes KL divergence with respect to the true joint probability distribution, $P[\cdot]$, for the data) can be obtained by *independently* picking the highest mutual information parents, $\pi_j(t)$, for each $X_j(t)$ for $j = 1, \ldots, M$ (and, because of **A2** and **A3**, we need to carry-out this parents' selection step only for the M nodes in any one time slice, t, that satisfies $(W < t \leq T)$). Moreover, from Eq. (3.7) it is clear that if we had a lower-bound ϑ on the mutual information terms, $I[X_j(t)\,;\,\pi_j(t)]$, it would imply an upper-bound Δ_ϑ on the KL divergence $D_{KL}(P||Q)$ between $P[\cdot]$ and $Q[\cdot]$. In other words, a good DBN-based approximation of the underlying stochastics (i.e. one with small KL divergence) can be achieved by picking, for each node, a parent-set whose corresponding mutual information exceeds a user-defined threshold.

Remark 3.2.1. *Consider a sequence $\mathbf{X}(t) : t = 1, \ldots, T$, of time-evolving indicators for an event stream s over an alphabet of size M. Under assumptions* **A1** *and* **A2**, *the optimal DBN (one that minimizes KL divergence with the true data distribution) can be obtained by independently picking for each $X_j(t)$, $1 \leq j \leq M$ (and for some fixed $t > W$) a parent set $\pi_j(t)$ which maximizes mutual information $I[X_j(t)\,;\,\pi_j(t)]$. Moreover, picking $\pi_j(t)$ such that $I[X_j(t)\,;\,\pi_j(t)] > \vartheta$ implies a corresponding upper-bound Δ_ϑ on the KL divergence between the DBN and the true data distribution.*

However, picking such sets with high mutual information (while yields a good approximation) falls short of unearthing useful dependencies among the random variables. This is because mutual information is non-decreasing as more random variables are added to a parent-set (leading to a fully-connected network always being optimal). For a parent-set to be interesting, it should not only exhibit sufficient correlation (or mutual information) with the corresponding child-node, but should also successfully encode the conditional independencies among random variables in the system. This can be done by checking if, *conditioned* on a candidate parent-set, the mutual information between the corresponding child-node and all its non-descendants is always close to zero. (We provide more details later in Sec. 3.5.3).

3.3 Excitatory dynamic network (EDN)

The structure-learning approach described in Sec. 3.2 is applicable to any general DBN that satisfies **A1** and **A2**. We now introduce a specialized class of networks, which we call as *Excitatory Dynamic Networks* (or EDNs) where only certain kinds of conditional dependencies among nodes are permitted. Each event-type is assumed to occur with probability *less*

Table 3.1: CPT structure for EDNs. $\Pi = \{X_B, X_C, X_D\}$ denotes the set of parents for node X_A. The table lists excitatory constraints on the probability of observing $[X_A = 1]$ conditioned on the different value-assignments for X_B, X_C and X_D.

	Π			$P[X_A = 1 \mid \mathbf{a}_j]$
	X_B	X_C	X_D	
$\mathbf{a}_0$	0	0	0	$\epsilon < \frac{1}{2}$
$\mathbf{a}_1$	0	0	1	$\epsilon_1 \geq \epsilon$
$\mathbf{a}_2$	0	1	0	$\epsilon_2 \geq \epsilon$
$\mathbf{a}_3$	0	1	1	$\epsilon_3 \geq \epsilon, \epsilon_1, \epsilon_2$
$\mathbf{a}_4$	1	0	0	$\epsilon_4 \geq \epsilon$
$\mathbf{a}_5$	1	0	1	$\epsilon_5 \geq \epsilon, \epsilon_1, \epsilon_4$
$\mathbf{a}_6$	1	1	0	$\epsilon_6 \geq \epsilon, \epsilon_2, \epsilon_4$
$\mathbf{a}_7(\mathbf{a}^*)$	1	1	1	$\phi > \epsilon, \frac{1}{2}, \epsilon_j \forall j$

than 0.5 in the data. This corresponds to a sparse-data assumption. A collection, Π, of random variables is said to have an *excitatory* influence on an event-type, $A \in \mathcal{E}$, if occurrence of events corresponding to the variables in Π, increases the probability of A to *greater than 0.5*. We define an Excitatory Dynamic Network as a DBN in which *all* parent nodes can only exert excitatory influences on corresponding child nodes. For example, EDNs will allow relationships like "if B, C and D occur (say) 2 time-ticks apart, the probability of A increases." However, EDNs cannot encode excitations-in-absence like "when A does not occur, the probability of B occurring 3 time-ticks later increases" or inhibitions like "when A occurs, the probability of B occurring 3 time-ticks later decreases." Excitatory networks are natural in neuroscience, where one is interested in unearthing conditional dependency relationships among neuron spiking patterns. Several regions in the brain are known to exhibit predominantly excitatory relationships [85] and our model is targeted towards unearthing these.

The excitatory assumption manifests as constraints on the conditional probability tables associated with the DBN. Consider a node* X_A (an indicator variable for event-type $A \in \mathcal{E}$) and let Π denote a parent-set for X_A. In excitatory networks, the probability of A conditioned on the occurrence of all event-types in Π, should be *at least as high as* the corresponding conditional probability when only some (though not all) of the event-types of Π occur. Further, the probability of A is less than 0.5 when none of the event-types of Π occur and greater than 0.5 when all the event-types of Π occur. For example, let $\Pi = \{X_B, X_C, X_D\}$. The conditional probability table (or CPT) for A given Π is shown in Table 3.1. The different conditioning contexts come about by the occurrence or otherwise of each of the events in Π. These are denoted by $\mathbf{a}_j,\ j = 0, \ldots, 7$. So while $\mathbf{a}_0$ represents the all-zero assignment (i.e. none of the events B, C or D occur), $\mathbf{a}_7$ (or $\mathbf{a}^*$) denotes the

*To facilitate simple exposition, time-stamps of random-variables are dropped from the notation in this discussion.

all-ones assignment (i.e. all the events B, C and D occur). The last column of the table lists the corresponding conditional probabilities along with the associated excitatory constraints. The baseline propensity of A is denoted by ϵ and the only constraint on it is that it must be less than $\frac{1}{2}$. Conditioned on the occurrence of any event of Π, the propensity of A can only increase, and hence, $\epsilon_j \geq \epsilon \ \forall j$ and $\phi \geq \epsilon$. Similarly, when both C and D occur, the probability must be at least as high as that when either C or D occur alone (i.e. we must have $\epsilon_3 \geq \epsilon_1$ and $\epsilon_3 \geq \epsilon_2$). Finally, for the all-ones case, denoted by $\Pi = \mathbf{a}^*$, the conditional probability must be greater than $\frac{1}{2}$ and must also satisfy $\phi \geq \epsilon_j \ \forall j$.

In the context of DBN learning, an excitatory assumption on the data implies that event-types will occur *frequently* after their respective parents (with suitable delays). This will allow us to estimate EDN structures using frequent pattern discovery algorithms (which have been a mainstay in data mining for many years). We now present a simple necessary condition on the probability (or frequency) of parent-sets in an excitatory network.

Theorem 3.3.1. *Let X_A denote a node corresponding to the event-type $A \in \mathcal{E}$ in an Excitatory Dynamic Network (EDN). Let Π denote the parent-set for X_A in the EDN. Let ϵ^* be an upper-bound on the conditional probabilities $P[X_A = 1 \mid \Pi = \mathbf{a}]$ for all $\mathbf{a} \neq \mathbf{a}^*$ (i.e. for all but the all-ones assignment in Π). If the mutual information $I[X_A\,;\,\Pi]$ exceeds $\vartheta(> 0)$, then the joint probability of an occurrence of A along with all events of Π satisfies $P[X_A = 1, \Pi = \mathbf{a}^*] \geq P_{min}\Phi_{min}$, where*

$$P_{min} = \frac{P[X_A = 1] - \epsilon^*}{1 - \epsilon^*} \tag{3.8}$$

$$\Phi_{min} = h^{-1}\left[\min\left(1, \frac{h(P[X_A = 1]) - \vartheta}{P_{min}}\right)\right] \tag{3.9}$$

and where $h(\cdot)$ denotes the binary entropy function $h(q) = -q\log q - (1-q)\log(1-q)$, $0 < q < 1$ and $h^{-1}[\cdot]$ denotes its pre-image greater than $\frac{1}{2}$.

Proof: Under the excitatory model we have $P[X_A = 1 \,|\, \Pi = \mathbf{a}^*] > P[X_A = 1 \,|\, \Pi = \mathbf{a}] \ \forall \mathbf{a} \neq \mathbf{a}^*$. First we apply ϵ^* to terms in the expression for $P[X_A = 1]$:

$$\begin{aligned} P[X_A = 1] &= P[\Pi = \mathbf{a}^*]P[X_A = 1 \mid \Pi = \mathbf{a}^*] \\ &+ \sum_{\mathbf{a} \neq \mathbf{a}^*} P[\Pi = \mathbf{a}]P[X_A = 1 \mid \Pi = \mathbf{a}] \\ &\leq P[\Pi = \mathbf{a}^*] + (1 - P[\Pi = \mathbf{a}^*])\epsilon^* \end{aligned}$$

This gives us $P[\Pi = \mathbf{a}^*] \geq P_{min}$ (see Fig. 3.2). Next, since we are given that mutual information $I[X_A\,;\,\Pi]$ exceeds ϑ, the corresponding conditional entropy must satisfy:

$$\begin{aligned} H[X_A \mid \Pi] &= P[\Pi = \mathbf{a}^*]h(P[X_A = 1 \mid \Pi = \mathbf{a}^*]) \\ &+ \sum_{\mathbf{a} \neq \mathbf{a}^*} P[\Pi = \mathbf{a}]h(P[X_A = 1 \mid \Pi = \mathbf{a}]) \\ &< H[X_A] - \vartheta = h(P[X_A = 1]) - \vartheta \end{aligned}$$

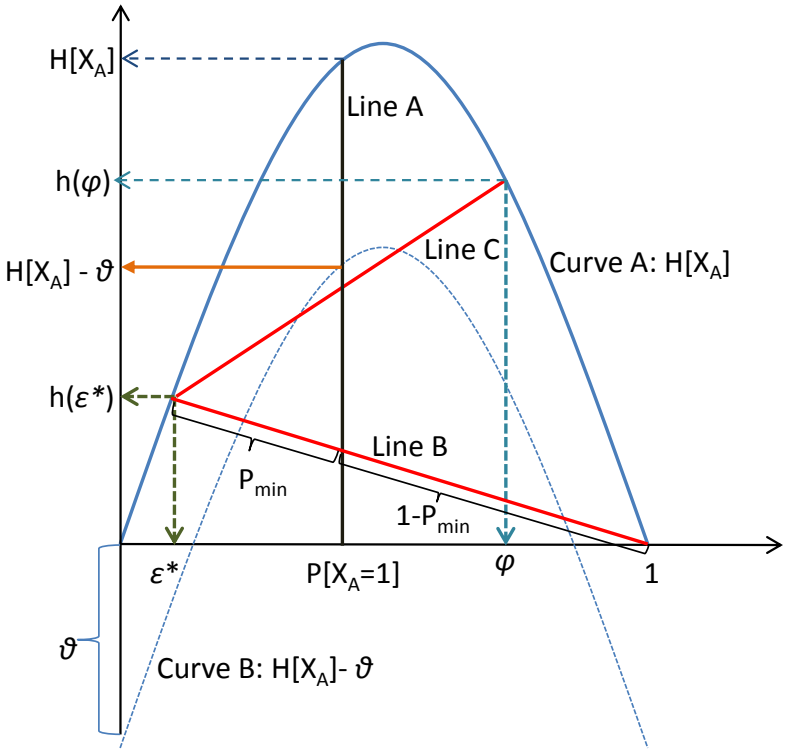


Figure 3.2: An illustration of the results obtained in Theorem 3.3.1. x-axis of the plot is the range of $P[X_A = 1]$ and y-axis is the corresponding range of entropy $H[X_A]$. For the required mutual information criteria, the conditional entropy must lie below $H[X_A] - \vartheta$ and on line A. At the boundary condition $[\phi = 1]$, Line A splits Line B in the ratio $P_{min} : (1 - P_{min})$. This gives the expression for P_{min}.

Every term in the expression for $H[X_A \mid \Pi]$ is non-negative, and hence, each term (including the first one) must be less than $(h(P[X_A = 1]) - \vartheta)$. Using $(P[\Pi = \mathbf{a}^*] \geq P_{min})$ in the above inequality and observing that $P[X_A = 1 \mid \Pi = \mathbf{a}^*]$ must be greater than 0.5 for an excitatory network, we now get $(P[X_A = 1 \mid \Pi = \mathbf{a}^*] > \Phi_{min})$. This completes the proof. □

3.3.1 Utility of EDNs

In Sec. 3.2, we derived conditions for optimal DBN approximations based on mutual information criteria. We showed that if each node was associated with a parent set that had high mutual information with the node, then the resulting DBN would have small KL divergence with respect to the true data distribution. The difficulty, however, is that searching for these sets with high mutual information is a computational expensive proposition. Excitatory Dynamic Networks, introduced in Sec. 3.3, alleviates this problem by focussing an a subclass

of DBNs that only encode excitatory relationships among nodes.

The main result about EDNs that we can exploit to develop efficient algorithms is *Theorem 3.3.1.* The theorem essentially translates a lower-bound ϑ on the mutual information $I[X_A\,;\,\Pi]$ between a node X_A and its parents Π, to a lower-bound $P_{min}\Phi_{min}$ on the joint probability $P[X_A = 1\,,\,\Pi = \mathbf{a}^*]$ of the occurrence of events corresponding to the node and its parents. Given a node X_A, identifying all sets, Π, for which the joint probabilities of the special form $P[X_A = 1\,,\Pi = \mathbf{a}^*]$ exceed a threshold falls in the realm of frequent pattern discovery in data mining. It is well-known in data mining literature that, if the data is sparse, frequent pattern discovery algorithms constitute a very efficient and effective class of methods for unearthing strong correlations in the data. In this work, we exploit these methods for efficiently learning optimal DBNs from event streams. Specifically, we define correlations of the form $[X_A = 1\,,\,\Pi = \mathbf{a}^*]$ as a new class of patterns called *fixed-delay episodes* and develop fast algorithms for discovering all fixed-delay episodes in the data whose frequencies exceed some threshold (See Sec. 3.4 for details). *Theorem 3.3.1* shows that a minimum mutual information constraint for a node and its parents translates to a (minimum) frequency threshold for discovering frequent fixed-delay episodes.

The lower-bound on joint probabilities $P[X_A\,,\,\Pi = \mathbf{a}^*])$ as per *Theorem 3.3.1* is $P_{min}\Phi_{min}$, where P_{min} and Φ_{min} depend on two user-defined quantities (cf. Eqs. 3.8-3.9): (i) the minimum mutual information threshold ϑ, and (ii) the upper-bound ϵ^* on the conditional probabilities in all-but-the-last-row of the CPT for X_A. It is easy to see that as the ϑ increases Φ_{min} increases and consequently, the threshold $P_{min}\Phi_{min}$ also increases. Similarly, as ϵ^* increases P_{min} decreases, as does Φ_{min}. Thus, as ϵ^* increases the threshold $P_{min}\Phi_{min}$ decreases. These relationships facilitate the use of ϵ^* and ϑ as exploratory 'knobs' for EDN learning. In general, using a lower threshold (i.e. a lower $P_{min}\Phi_{min}$) will allow us to detect even the weaker correlations, which results in estimation of a *more complete* network. However, lower thresholds translate to higher run-times, and as is well-known in pattern discovery literature, at some sufficiently low threshold the computation can become infeasible. The theoretical results allow us to systematically explore this trade-off between inferring weaker dependencies and requiring higher run-times. In Sec. 3.6, we describe experiments that demonstrate the effect of ϵ^* and ϑ on the performance of our algorithms. Finally, note that $P_{min}\Phi_{min}$ also depends on the marginal probability of A in the data, namely $P[X_A = 1]$. Unlike ϵ^* and ϑ (which are user-defined parameters) $P[X_A = 1]$ is estimated from the data. The relationship of $P_{min}\Phi_{min}$ with $P[X_A = 1]$, however, is a bit more complicated – the threshold can increase or decrease with $P[X_A = 1]$ depending on the values chosen for ϵ^* and ϑ.

3.4 Fixed delay episodes

In Secs. 3.2-3.3 we developed the theoretical framework for estimating optimal Excitatory Dynamic Networks from time-stamped event streams. We pointed out that we can reduce the search space for the parent sets of a node by constructing candidate parent sets out of

frequent fixed-delay episodes in the data (cf. Sec. 3.3.1). In this section, we formally define fixed-delay episodes and show how to use the frequencies of fixed-delay episodes to compute corresponding joint probabilities and mutual information quantities.

In the framework of frequent episode discovery [8] the data is a single long stream of events over a finite alphabet (cf. Sec. 3.2 and *Example 3.2.1*). An ℓ-node (serial) episode, α, is defined as a tuple, $(V_\alpha, <_\alpha, g_\alpha)$, where $V_\alpha = \{v_1, \ldots, v_\ell\}$ denotes a collection of nodes, $<_\alpha$ denotes a *total order*[†] such that $v_i <_\alpha v_{i+1},\ i = 1, \ldots, (\ell - 1)$. If $g_\alpha(v_j) = A_{i_j}, A_{i_j} \in \mathcal{E}, j = 1, \ldots, \ell$, we use the graphical notation $(A_{i_1} \rightarrow \cdots \rightarrow A_{i_\ell})$ to represent α. An occurrence of α in event stream, $s = \langle (E_1, \tau_1), (E_2, \tau_2), \ldots, (E_n, \tau_n) \rangle$, is a map $h : V_\alpha \rightarrow \{1, \ldots, n\}$ such that (i) $E_{h(v_j)} = g(v_j)\ \forall v_j \in V_\alpha$, and (ii) for all $v_i <_\alpha v_j$ in V_α, the times of occurrence of the i^{th} and j^{th} events in the occurrence satisfy $\tau_{h(v_i)} \leq \tau_{h(v_j)}$ in s.

Example 3.4.1. *Consider a 3-node episode $\alpha = (V_\alpha, <_\alpha, g_\alpha)$, such that, $V_\alpha = \{v_1, v_2, v_3\}$, $v_1 <_\alpha v_2$, $v_2 <_\alpha v_3$ and $v_1 <_\alpha v_3$, and $g_\alpha(v_1) = A$, $g_\alpha(v_2) = B$ and $g_\alpha(v_3) = C$. The graphical representation for this episode is $\alpha = (A \rightarrow B \rightarrow C)$, indicating that in every occurrence of α, an event of type A must appear before an event of type B, and the B must appear before an event of type C. For example, in sequence (3.1), the subsequence $\langle (A, 1), (B, 3), (C, 9) \rangle$ constitutes an occurrence of $(A \rightarrow B \rightarrow C)$. For this occurrence, the corresponding h-map is given by, $h(v_1) = 1$, $h(v_2) = 2$ and $h(v_3) = 5$.*

There are many ways to incorporate explicit time constraints in episode occurrences like the windows-width constraint of [8]. Episodes with inter-event *gap* constraints were introduced in [19]. For example, the framework of [19] can express the temporal pattern "B must follow A *within* 5 time-ticks and C must follow B *within* 10 time-ticks." Such a pattern is represented using the graphical notation, $(A \xrightarrow{[0-5]} B \xrightarrow{[0-10]} C)$. We use a simple sub-case of the inter-event gap constraints, in the form of *fixed* inter-event time-delays. For example, $(A \xrightarrow{5} B \xrightarrow{10} C)$ represents a fixed-delay episode, every occurrence of which must comprise an A, followed by a B *exactly* 5 time-ticks later, which in-turn is followed by a C *exactly* 10 time-ticks later.

Definition 3.4.1. *An ℓ-node* fixed-delay episode *is defined as a pair, $(\alpha, \mathcal{D})$, where $\alpha = (V_\alpha, <_\alpha, g_\alpha)$ is the usual (serial) episode of [8], and $\mathcal{D} = (\delta_1, \ldots, \delta_{\ell-1})$ is a sequence of $(\ell - 1)$ non-negative delays. Every occurrence, h, of the fixed-delay episode in an event sequence s must satisfy the inter-event constraints, $\delta_j = (\tau_{h(v_{j+1})} - \tau_{h(v_j)}),\ j = 1, \ldots, (\ell - 1)$. $(A_{i_1} \xrightarrow{\delta_1} \cdots \xrightarrow{\delta_{\ell-1}} A_{i_\ell})$ is our graphical notation for the inter-event episode, $(\alpha, \mathcal{D})$, where $A_{i_j} = g_\alpha(v_j),\ j = 1, \ldots, \ell$.*

Definition 3.4.2. *Two occurrences, h_1 and h_2, of a fixed-delay episode, $(\alpha, \mathcal{D})$, are said to be* distinct*, if they do not share any events in the data stream, s. Given a user-defined,*

[†]In general, $<_\alpha$ can be any partial order over V_α. We focus on only total orders here and show how multiple total orders can be used to model DBNs of arbitrary -arity. In [8], such total orders are referred to as *serial* episodes.

$W > 0$, frequency of $(\alpha, \mathcal{D})$ in s, denoted $f_s(\alpha, \mathcal{D}, W)$, is defined as the total number of distinct occurrences of $(\alpha, \mathcal{D})$ in s that terminate strictly after W.

In general, counting distinct occurrences of episodes suffers from computational inefficiencies [34]. Each occurrence of an episode $(A \rightarrow B \rightarrow C)$ is a substring that looks like $A * B * C$, where $*$ denotes a *variable-length* don't-care, and hence, counting all distinct occurrences in the data stream can require memory of the same order as the data sequence which typically runs very long. However, in case of fixed-delay episodes, it is easy to track distinct occurrences efficiently. For example, when counting frequency of $(A \xrightarrow{3} B \xrightarrow{5} C)$, if we encounter an A at time t, to recognize an occurrence involving this A we only need to check for a B at time $(t+3)$ and for a C at time $(t+8)$. In addition to being attractive from an efficiency point-of-view, we show next in Sec. 3.4.1 that the distinct occurrences-based frequency count for fixed-delay episodes will allow us to interpret relative frequencies as appropriate DBN marginals. (Note that the W in *Definition 3.4.2* is same as the length of the history window used in the constraint **A1**. Skipping occurrences terminating in the first W time-ticks makes it easy to normalize the frequency count into a probability measure).

3.4.1 Marginals from episode frequencies

In this section, we describe how to compute mutual information from the frequency counts of fixed-delay episodes. For this, every subset of event-indicators in the network is associated with a fixed-delay episode.

Definition 3.4.3. *Let $\{X_j(t) : j = 1, \ldots, M;\ t = 1, \ldots, T\}$ denote the collection of event-indicators used to model event stream, $s = \langle (E_1, \tau_1), \ldots (E_n, \tau_n) \rangle$, over alphabet, $\mathcal{E} = \{A_1, \ldots, A_M\}$. Consider an ℓ-size subset, $\mathcal{X} = \{X_{i_1}(t_1), \ldots, X_{i_\ell}(t_\ell)\}$, of these indicators, and without loss of generality, assume $t_1 \leq \cdots \leq t_\ell$. Define the $(\ell - 1)$ inter-event delays in $\mathcal{X}$ as follows: $\delta_j = (t_{j+1} - t_j)$, $j = 1, \ldots, (\ell - 1)$. The fixed-delay episode, $(\alpha(\mathcal{X}), \mathcal{D}(\mathcal{X}))$, that is associated with the subset, $\mathcal{X}$, of event-indicators is defined by $\alpha(\mathcal{X}) = (A_{i_1} \rightarrow \cdots \rightarrow A_{i_\ell})$, and $\mathcal{D}(\mathcal{X}) = \{\delta_1, \ldots, \delta_{\ell-1}\}$. In graphical notation, the fixed-delay episode associated with $\mathcal{X}$ can be represented as follows:*

$$(\alpha(\mathcal{X}), \mathcal{D}(\mathcal{X})) = (A_{i_1} \xrightarrow{\delta_1} \cdots \xrightarrow{\delta_{\ell-1}} A_{i_\ell}) \tag{3.10}$$

For computing mutual information we need the marginals of various subsets of event-indicators in the network. Given a subset like $\mathcal{X} = \{X_{i_1}(t_1), \ldots, X_{i_\ell}(t_\ell)\}$, we need estimates for probabilities of the form, $P[X_{i_1}(t_1) = a_1, \ldots, X_{i_\ell}(t_\ell) = a_\ell]$, where $a_j \in \{0, 1\}$, $j = 1, \ldots, \ell$. The fixed-delay episode, $(\alpha(\mathcal{X}), \mathcal{D}(\mathcal{X}))$, that is associated with $\mathcal{X}$ is given by *Definition 3.4.3* and its frequency in the data stream, s, is denoted by $f_s(\alpha(\mathcal{X}), \mathcal{D}(\mathcal{X}), W)$ (as per *Definition 3.4.2*) where W denotes the length of history window as per **A1**. Since an

occurrence of the fixed-delay episode, $(\alpha(\mathcal{X}), \mathcal{D}(\mathcal{X}))$, can terminate in each of the $(T-W)$ time-ticks in s, the probability of an all-ones assignment for the random variables in $\mathcal{X}$ is given by:

$$P[X_{i_1}(t_1) = 1, \ldots, X_{i_\ell}(t_\ell) = 1] = \frac{f_s(\alpha(\mathcal{X}), \mathcal{D}(\mathcal{X}), W)}{T - W} \tag{3.11}$$

For all other assignments (i.e. for assignments that are not all-ones) we use inclusion-exclusion to obtain corresponding probabilities. Inclusion-exclusion has been used before in data mining, e.g., in [86], to obtain exact or approximate frequency counts for arbitrary boolean queries using only counts of *frequent* itemsets in the data. In our case, counting distinct occurrences of fixed-delay episodes facilitates use of the inclusion-exclusion formula for obtaining the probabilities needed for computing mutual information of different candidate parent-sets. Consider the set, $\mathcal{X} = \{X_{i_1}(t_1), \ldots, X_{i_\ell}(t_\ell)\}$, of ℓ event-indicators, and let $\mathcal{A} = (a_1, \ldots, a_\ell)$, $a_j \in \{0, 1\}$, $j = 1, \ldots, \ell$, be an assignment for the event-indicators in $\mathcal{X}$. Let $\mathcal{U} \subset \mathcal{X}$ denote the subset of indicators out of $\mathcal{X}$ for which corresponding assignments (in $\mathcal{A}$) are 1's, i. e. $\mathcal{U} = \{X_{i_j}(t_j) \in \mathcal{X} \ : \ j \text{ s.t. } \ a_j = 1 \text{ in } \mathcal{A}, 1 \leq j \leq \ell\}$. Inclusion-exclusion is used to compute the probabilities as follows:

$$\begin{aligned} & P[X_{i_1}(t_1) = a_1, \ldots, X_{i_\ell}(t_\ell) = a_\ell] \\ & \quad = \sum_{\substack{\mathcal{Y} \text{ s.t.} \\ \mathcal{U} \subseteq \mathcal{Y} \subseteq \mathcal{X}}} (-1)^{|\mathcal{Y} \setminus \mathcal{U}|} \left(\frac{f_s(\mathcal{Y})}{T - W} \right) \end{aligned} \tag{3.12}$$

where $f_s(\mathcal{Y})$ is short-hand for $f_s(\alpha(\mathcal{Y}), \mathcal{D}(\mathcal{Y}), W)$, the frequency (cf. *Definition 3.4.2*) of the fixed-delay episode, $(\alpha(\mathcal{Y}), \mathcal{D}(\mathcal{Y}))$.

Finally, a note on mutual information computation. Consider a node $X_{i_\ell}(t_\ell)$ and its candidate parent set $\Pi = \{X_{i_1}(t_1), \ldots, X_{i_{\ell-1}}(t_{\ell-1})\}$. Construct the set $\mathcal{X} = \{X_{i_\ell}(t_\ell)\} \bigcup \Pi$. After computing all the necessary joint probabilities involving $X_{i_\ell}(t_\ell)$ and Π based on Eqs. (3.11)-(3.12) we can compute the mutual information $I[X_{i_\ell} \, ; \, \Pi]$ using the following expression:

$$\begin{aligned} I[X_{i_\ell}(t_\ell) \, ; \, \Pi] = & \sum P[X_{i_\ell}(t_\ell) = a_\ell, X_{i_1}(t_1) = a_1, \ldots, X_{i_{\ell-1}}(t_{\ell-1}) = a_{\ell-1}] \\ & \times \log_2 \left[\frac{P[X_{i_\ell} = a_\ell, \ X_{i_1}(t_1) = a_1, \ldots, \ X_{i_{\ell-1}}(t_{\ell-1}) = a_{\ell-1}]}{P[X_{i_\ell}(t_\ell) = a_\ell] P[X_{i_1}(t_1) = a_1, \ldots, \ X_{i_{\ell-1}}(t_{\ell-1}) = a_{\ell-1}]} \right] \end{aligned} \tag{3.13}$$

where $(a_1, \ldots, a_\ell) \in \{0, 1\}^\ell$ and the summation is over all possible values for $(a_1, \ldots, a_\ell)$. This way, all mutual information quantities that are needed for EDN learning can be computed using just the frequencies of appropriate fixed-delay episodes in the data.

Procedure 2 Overall Procedure

Input: Alphabet $\mathcal{E}$, event stream $s = \langle (E_1, \tau_1), \ldots, (E_n, \tau_n = T) \rangle$, length W of history window, conditional probability upper-bound ϵ^*, mutual information threshold, ϑ

Output: DBN structure (parent-set for each node in the network)

1: **for all** $A \in \mathcal{E}$ **do**
2: $X_A :=$ event-indicator of A at any time $t > W$
3: Set $f_{min} = (T - W) P_{min} \Phi_{min}$, using Eqs. (3.8)-(3.9)
4: Obtain set, $\mathcal{C}$, of fixed-delay episodes ending in A, with frequencies greater than f_{min} (cf. Sec. 3.5.2, *Procedure 3*)
5: **for all** fixed-delay episodes $(\alpha, \mathcal{D}) \in \mathcal{C}$ **do**
6: $\mathcal{X}_{(\alpha,\mathcal{D})} :=$ event-indicators corresponding to $(\alpha, \mathcal{D})$
7: Compute mutual information $I[X_A \, ; \, \mathcal{X}_{(\alpha,\mathcal{D})}]$
8: Remove $(\alpha, \mathcal{D})$ from $\mathcal{C}$ if $I[X_A \, ; \, \mathcal{X}_{(\alpha,\mathcal{D})}] < \vartheta$
9: Prune $\mathcal{C}$ using conditional mutual information criteria to distinguish direct from indirect influences (cf. Sec. 3.5.3)
10: Return (as parent-set for X_A) event-indicators corresponding to episodes in $\mathcal{C}$

3.5 Algorithms

3.5.1 Overall approach

In Secs. 3.2-3.4, we developed the formalism for learning an optimal DBN structure from event streams by using distinct occurrences-based counts of fixed-delay episodes to compute the DBN marginal probabilities. The top-level algorithm (cf. Sec. 3.2) for discovering the network is to fix any time $t > W$, to consider each $X_j(t)$, $j = 1, \ldots, M$, in-turn, and to find its set of parents in the network. Due to the translation invariance assumption **A2**, we need to do this *only once for each event-type* in the alphabet.

The algorithm is outlined in *Procedure 2*. For each $A \in \mathcal{E}$, we first compute the minimum frequency for episodes ending in A based on the relationship between mutual information and joint probabilities as per *Theorem 3.3.1* (line 3, *Procedure 1*). Then we use a pattern-growth approach (see *Procedure 3*) to discover all patterns terminating in A (line 4, *Procedure 1*). Each frequent pattern corresponds to a set of event-indicators (line 6, *Procedure 1*). The mutual information between this set of indicators and the node X_A is computed using inclusion-exclusion formula and only sets for which this mutual information exceeds ϑ are retained as candidate parent-sets (lines 5-8, *Procedure 1*). Finally, we prune out candidate parent-sets which have only *indirect influences* on A and return the final parent-sets for nodes corresponding to event-type A (lines 9-10, *Procedure 1*). This pruning step is based on conditional mutual information criteria (to be described later in Sec. 3.5.3).

Procedure 3 *pattern_grow*$(\alpha, \mathcal{D}, \mathcal{L}_{(\alpha,\mathcal{D})})$

Input: ℓ-node episode $(\alpha, \mathcal{D}) = (A_{j_1} \xrightarrow{\delta_1} \cdots \xrightarrow{\delta_{\ell-1}} A_{j\ell})$ and event sequence $s = \langle (E_1, \tau_1), \ldots, (E_n, \tau_n = T) \rangle$, Length of history window W, Frequency threshold f_{min}.

1: $\Delta = W - span(\alpha, \mathcal{D})$
2: **for all** $A \in \mathcal{E}$ **do**
3: **for** $\delta = 0$ to Δ **do**
4: **if** $\delta = 0$ **and** $(A_{j_1} > A$ **or** $\ell = 1)$ **then**
5: **continue**
6: $(\alpha', \mathcal{D}') = A \xrightarrow{\delta} \alpha$; $\mathcal{L}_{(\alpha',\mathcal{D}')} = \{\}$; $f_s(\alpha', \mathcal{D}') = 0$
7: **for all** $\tau_i \in \mathcal{L}_{(\alpha,\mathcal{D})}$ **do**
8: **if** $\exists (E_j, \tau_j)$ such that $E_j = A$ and $\tau_i - \tau_j = \delta$ **then**
9: Increment $f_s(\alpha', \mathcal{D}')$
10: $\mathcal{L}_{(\alpha',\mathcal{D}')} = \mathcal{L}_{(\alpha',\mathcal{D}')} \cup \{\tau_j\}$
11: **if** $f_s(\alpha', \mathcal{D}') \geq f_{min}$ **then**
12: Add $(\alpha', \mathcal{D}')$ to output set $\mathcal{C}$
13: **if** $span(\alpha', \mathcal{D}') \leq W$ **then**
14: *pattern_grow*$(\alpha', \mathcal{D}', \mathcal{L}_{(\alpha',\mathcal{D}')})$

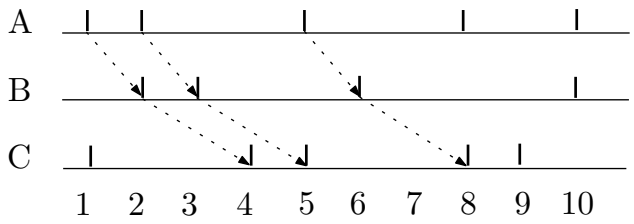


Figure 3.3: An event sequence showing 3 distinct occurrences of the episode $A \xrightarrow{1} B \xrightarrow{2} C$.

3.5.2 Discovering fixed delay episodes

We employ a pattern-growth algorithm (Procedure 3) for mining frequent fixed-delay episodes because, unlike *Apriori*-style algorithms, pattern-growth procedures allow use of different frequency thresholds for episodes ending in different alphabets. This is needed in our case, since, in general, *Theorem 3.3.1* prescribes different frequency thresholds for nodes in the network corresponding to different alphabets. The recursive procedure is invoked with $(\alpha, \mathcal{D}) = (A, \phi)$ and frequency threshold $f_{min} = (T - W)P_{min}\phi_{min}$ (Recall that in the main loop of *Procedure 1*, we look for parents of nodes corresponding to event-type $A \in \mathcal{E}$).

The pattern-growth algorithm listed in *Procedure 3* takes as input, an episode $(\alpha, \mathcal{D})$, a set of start times $\mathcal{L}_{(\alpha,\mathcal{D})}$, and the event sequence s. $\mathcal{L}_{(\alpha,\mathcal{D})}$ is a set of time stamps τ_i such that there is an occurrence of $(\alpha, \mathcal{D})$ starting at τ_i in s. For example, if at level 1 we have $(\alpha, \mathcal{D}) = (C, \phi)$, then $\mathcal{L}_{(C,\phi)} = \{1, 4, 5, 8, 9\}$ in the event sequence s shown in Fig 3.3. The algorithm obtains counts for all episodes like $(\alpha', \mathcal{D}')$ generated by extending $(\alpha, \mathcal{D})$ e.g. $B \xrightarrow{1} C$, ..., $A \xrightarrow{5} C$

etc. For an episode say $(\alpha', \mathcal{D}') = B \xrightarrow{2} C$, the count is obtained by looking for occurrences of event B at times $\tau_j = \tau_i - 2$ where $\tau_i \in \mathcal{L}_{(C,\phi)}$. In the example such B's at $\tau_j \in \mathcal{L}_{B \xrightarrow{2} C} = \{2, 3, 6\}$. The number of such occurrences (= 3) gives the count of $B \xrightarrow{2} C$. At every step the algorithm tries to grow an episode with count $f_s > f_{min}$ otherwise stops.

3.5.3 Conditional MI criteria

The final step in determining the parents of a node X_A involves testing of conditional mutual information criteria (cf. line 10, *Procedure 1*). The input to this step is a set, $\mathcal{C}_A$, of candidate parent sets for each node X_A in the network, such that each candidate set $\mathcal{Y}$ has high mutual information with X_A i.e. we have $I[X_A\,;\,\mathcal{Y}] > \vartheta\ \forall \mathcal{Y} \in \mathcal{C}_A$. Consider two such sets $\mathcal{Y}$ and $\mathcal{Z}$, each having sufficient mutual information with X_A. Our conditional mutual information criterion is: *remove $\mathcal{Y}$ from the set of candidate parents (of X_A), if $I[X_A\,;\,\mathcal{Y} \mid \mathcal{Z}] = 0$*[‡]. We repeat this test for every pair of episodes in $\mathcal{C}$.

To understand the utility of this criterion, there are two cases to consider: (i) either $\mathcal{Y} \subset \mathcal{Z}$ or $\mathcal{Z} \subset \mathcal{Y}$, and (ii) both $\mathcal{Y} \not\subset \mathcal{Z}$ and $\mathcal{Z} \not\subset \mathcal{Y}$. In the first case, our conditional mutual criterion will ensure that we pick the larger set as a parent only if it brings more information about X_A than the smaller set. In the second case, we are interested in eliminating sets which have a high mutual information with X_A because of indirect influences. For example, if the network were such that C excites B and B excites A, then X_C can have high mutual information with X_A, but we do not want to report X_C as a parent of X_A, since C influences A only *through* B. Our conditional mutual information criterion will detect this and eliminate X_C from the set of candidate parents (of X_A) because it will detect $I[X_A\,;\,X_C \mid X_B] = 0$.

We now summarize the conditional mutual information step in our algorithm. We pick candidates out of $\mathcal{C}_A$ in decreasing order of their respective mutual informations with X_A. A candidate set, say $\mathcal{Y}$, is declared to belong to the final parent set Π only if $I[X_A\,;\,\mathcal{Y} \mid \mathcal{Z}] = 0$ for all $\mathcal{Z} \in \mathcal{C}_A,\ \mathcal{Z} \neq \mathcal{Y}$. Note that the time needed for pruning, which is quadratic in the size of $\mathcal{C}_A$, is typically small since the number of frequent fixed-delay episodes ending in A is typically small.

3.6 Results

In this section, we present results both on data gathered from mathematical models of spiking neurons and real neuroscience experiments.

[‡]We use a small threshold parameter to ascertain this equality.

3.6.1 Spiking neuronal network models

Several mathematical models of spiking neurons have been proposed and studied in neuroscience [87, 88, 38, 19, 89, 90]. For example, [87] employ a linear Poisson model where spike train signals are generated using an inhomogeneous Poisson process, while [88] study a point process observation model of neural spiking activity. In [38] the spike train of a neuron is expressed as a conditionally Poisson point process. A network of interacting neurons, each modeled as an inhomogeneous Poisson process was introduced in [19] and this model was extended to incorporate higher-order interactions in [90]. These models based on inter-dependent inhomogeneous Poisson processes can incorporate sophisticated relationships between neurons in the network and constitute a rich source of data for testing our DBN learning models. We briefly introduce the spiking neuronal model of [90] which we use in our experimental work.

The data generation model[§] takes as input an inter-connected set of neurons. The spike train of each neuron is modeled as an inhomogeneous Poisson process and interactions with other neurons are introduced by controlling the firing rate of the neuron as a function of the input it receives from others. This rate is updated every ΔT time units (We can think of ΔT as the resolution of each time-tick). The firing rate $\lambda_i(t)$ of the i^{th} neuron at time instant $t\Delta T$ (or t^{th} time-tick) is given by

$$\lambda_i(t) = \frac{\widehat{\lambda}_1}{1 + exp(-I_i(t) + d)} \tag{3.14}$$

where $I_i(t)$ denotes the input received by the i^{th} neuron at time $t\Delta T$, $\widehat{\lambda}_1$ determines the (high) firing rate of the neuron in the excited state (when sufficient potential accumulates at its input) and d denotes an offset parameter that is fixed based on the user-defined rest (or quiescent) firing rate $\widehat{\lambda}_0$ of the neuron (when the input is *zero*). The simulator determines the high firing rate $\widehat{\lambda}_1$ based on the desired conditional probability ρ of firing of the neuron when it receives its full expected input (ρ is a user-defined parameter). The network inter-connect model gives it the sophistication needed for simulating higher-order interactions. The model allows for variable delays which mimic the delays in conduction pathways of real neurons. The total input $I_i(t)$ received by the i^{th} neuron at time-tick t is given by

$$I_i(t) = \sum_j \beta_{ij} Y_{j(t-\tau_{ij})} + \ldots + \sum_{j \ldots \ell} \beta_{ij \ldots l} Y_{j(t-\tau_{ij})} \ldots Y_{\ell(t-\tau_{i\ell})} \tag{3.15}$$

where $Y_{j(t-\tau_{ij})}$ is the indicator of a spike on j^{th} neuron τ_{ij} time-ticks before t and $\beta_{(.)}$'s are the weights of the corresponding synaptic interactions. The first summation in Eq. (3.15) models first-order interactions, while the subsequent summations model progressively higher-order interactions. For example, to model a strong first-order connection from neuron j to neuron i with appropriate synaptic delays (i.e. to obtain a high conditional probability for the firing

[§]Simulator courtesy Raajay Viswanathan, Electrical Engineering, Indian Institute of Science, Bangalore.

of i given that j has fired at an appropriate time in the past) the simulator would set a high value for the weight β_{ij}. Similarly, to model a higher-order interaction such as one that sets a high conditional probability of firing for neuron i given that neurons j, k and ℓ fired (at times determined by the corresponding synaptic delays) the simulator assigns a high value for the weight β_{ijkl}. Finally, the simulator also incorporates a short refractory period for the neurons (which is the time for which a neuron does not respond to any stimulus after it has just spiked).

Types of networks

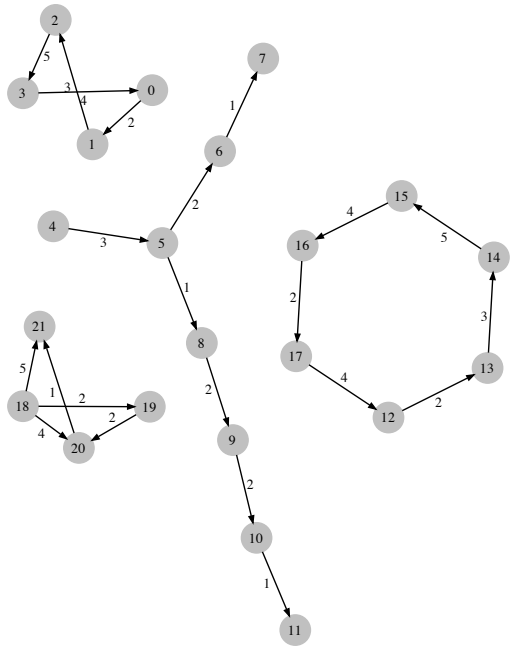
We generate spike train data from a wide range of networks by embedding several different kinds of neuronal interactions and by varying the different user-defined parameters in the simulator.

1. **Causative chains and higher-order structures**: A higher-order chain is one where parent sets are not restricted to be of cardinality one. In the example network of Fig. 3.4(a), there are four disconnected components with two of them having cycles. (Recall this would be 'illegal' in a static Bayesian network formulation). Also the component consisting of nodes $\{18, 19, 20, 21\}$ exhibits higher-order interactions. The node 20 fires with high probability when node 18 has fired 4 ms before and node 19 has fired 2 ms before. Similarly node 21 is activated by nodes $\{18, 20\}$ firing at respective delays.

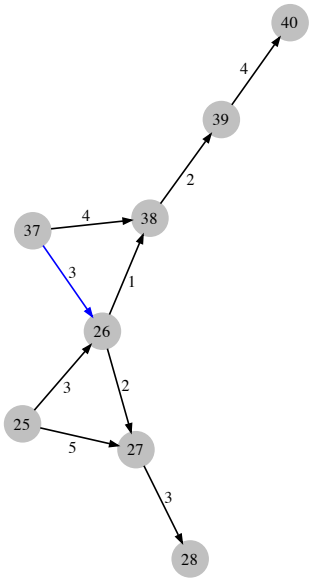
2. **Overlapping causative chains**: The graph shown in Fig. 3.4(b) has two chains $\{25, 26, 27, 28\}$ and $\{37, 26, 38, 39, 40\}$. Node 26 is common to both chains and can be independently excited by 25 or 37. Also 27 is activated by $\{25, 26\}$ together and 38 is activated by $\{26, 37\}$. Thus a firing event on 25 excites one chain while a firing event on 37 excites the other chain. This shows one possible way in which neurons can participate in multiple circuits at the same time (e.g. polychronous circuits [91]). Depending on the stimulus sequence, the same neurons can participate in different cascades of firing events (encoding completely different pieces of information).

3. **Syn-fire chains**: Another important class of patterns studied in neuron spike trains is called synfire chains. This consists of groups of synchronously firing neurons strung together repeating over time. An example of a syn-fire chain is given in Fig. 3.4(c). In [19], it was noted that discovering such patterns required a combination of serial and parallel episode mining. But the DBN approach applies more naturally to mining such network structures.

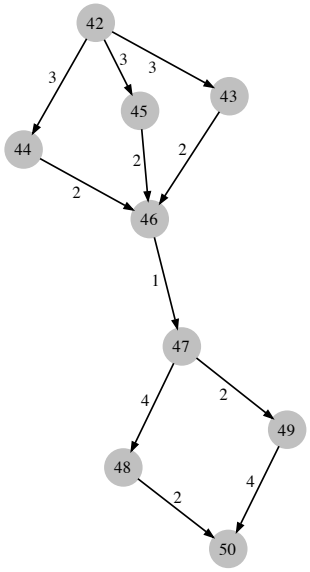
4. **Polychronous circuits**: Groups of neurons that fire in a time-locked manner with respect to each other are referred to as polychronous groups. This notion was introduced



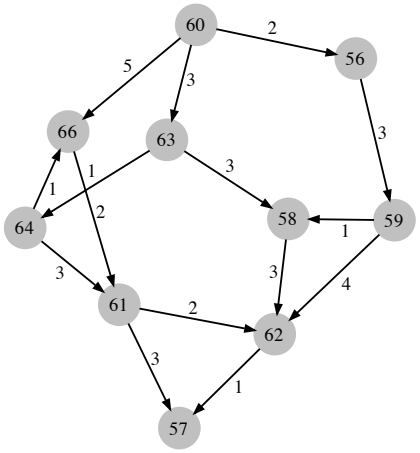
(a) Higher-order causative chains



(b) Overlapping causative chains



(c) Syn-fire Chains



(d) Polychronous Circuits

Figure 3.4: Four classes of DBNs investigated in our experiments.

Table 3.2: Synthetic datasets used in EDN evaluation.

Dataset Name	Data length	Alphabet Size	Rest Firing Rate $\widehat{\lambda}_0$	Conditional Probability ρ
A1	60000	100	0.02	0.9
A2	60000	125	0.02	0.9
A3	60000	150	0.02	0.9
A4	60000	175	0.02	0.9
B5	60000	100	0.01	0.9
B6	60000	100	0.015	0.9
B7	60000	100	0.02	0.9
B8	60000	100	0.025	0.9
C9	60000	100	0.02	0.8
C10	60000	100	0.02	0.85
C11	60000	100	0.02	0.9
C12	60000	100	0.02	0.95
D13	60000	100	0.02	0.9
D14	90000	100	0.02	0.9
D15	120000	100	0.02	0.9

in [91] and gives rise to an important class of patterns. Once again, our DBN formulation is a natural fit for discovering such groups from spike train data. A polychronous circuit is shown in Fig 3.4(d).

Synthetic datasets

The data sets we used for our experiments are listed in Table 3.2. The name of the data set is listed in Column 1, the length of the data set (or number of time-slices in the data sequence) in Column 2, the size of the alphabet (or total number of neurons in the network) in Column 3, the rest firing rate $\widehat{\lambda}_0$ in Column 4 and the conditional probability ρ (that a neuron will fire when it receives its full expected input) in Column 5. All the substructures listed in Figs. 3.4(a)-3.4(d) were inserted into all the data sets. The data sets are arranged into 4 groups. In the A-group (A1-A4) the data length is 60000 events, the rest firing rate $\widehat{\lambda}_0$ is 0.02, the conditional probability ρ is 0.9 and the size of alphabet is varied from 100 to 175. In the B-group (B5-B8) the rest firing rate $\widehat{\lambda}_0$ is varied from 0.01 to 0.025 keeping everything else constant. Similarly, in the C-group (C9-C12) the conditional probability ρ is varied from 0.8 to 0.95. Finally, in the D-group (D13-D15) the data lengths are varied from 60000-120000 events.

3.6.2 Competing methods

Most of the existing DBN literature on structure learning is focused on networks with first order Markov assumption and use hidden variables to circumvent this limitation [35, 84]. In contrast, our formulation allows variable order structure learning controlled by the parameters k and w in our algorithm. This makes baseline comparison with existing methods difficult. There is also a limit on the number of variables in the data for such algorithms [92, 93]. We found it most natural to compare our method with an implementation of Sparse Candidate algorithm [58] extended for learning DBN structures and the Simulated Annealing-based DBN learning proposed in [38]. (We use the shorthands SC and SA to denote the Sparse Candidate algorithm and the Simulated Annealing-based DBN respectively). SC was originally proposed for structure learning of regular Bayesian Networks. We use a natural extension of SC for DBNs based on our assumptions in Section 3.2. In the Dynamic Probabilistic Networks formulation [35], two scoring functions were proposed, namely BIC (cf. Eq. (3.16)) and BDe (cf. Eq. (3.17)) (The latter assumes Dirichlet priors).

$$BIC = \sum_{i,j_i,k_i} N_{i,j_i,k_i} \log \frac{N_{i,j_i,k_i}}{\sum_{k_i} N_{i,j_i,k_i}} - \frac{\log N}{2} \#G \tag{3.16}$$

where N_{i,j_i,k_i} is number of times $X_i(t)$ takes the value k_i and its parent-set Π_i takes the value j_i, $\#G$ is the dimension of the bayesian network G (which in this case is the number of parameters in the conditional probability tables) and N is the number of time slices in the data sequence.

$$BDe = \prod_{i,j_i} \frac{\Gamma\left(\sum_{k_i} N'_{i,j_i,k_i}\right)}{\Gamma\left(\sum_{k_i} N'_{i,j_i,k_i} + N_{i,j_i,k_i}\right)} \prod_{k_i} \frac{\Gamma\left(N'_{i,j_i,k_i} + N_{i,j_i,k_i}\right)}{\Gamma\left(N'_{i,j_i,k_i}\right)} \tag{3.17}$$

where N_{i,j_i,k_i} is same as before, and the hyper-parameters N'_{i,j_i,k_i} are selected by assuming an empty prior network: $N'_{i,j_i,k_i} = \hat{N} \times P(X_i(t) = k_i)$ where $\hat{N}$ is called the effective data length of the prior and we set it at 10% of the actual data length. We experimented with both these scoring functions and the results were comparable in terms of quality as well as run-times. Hence, we quote SC results only for BIC. However we use BDe score for SA (as suggested by [38]). The SA method simply uses the BDe score along with simulated annealing for searching through the space of candidates. The main tunable parameter in simulated annealing is the temperature or cooling schedule. We experimented with many schedules in our implementation: $T = T_0(0.99)^k$ for $T_0 = 10^2, 10^4, 10^6$ and $T = \frac{T_0}{1+\log C.k}$ for $T_0 = 10^4, 10^6$ and $C = 1, 10, 100$. Further, at each iteration of SA we remembered the best state so far, and used the best state after around 2.5 hours as the final answer.

3.6.3 Performance

First we summarize the performance of SA on the data sets of Table 3.2. The table quotes the average precision and recall values obtained across all 15 data sets along with the corre-

Table 3.3: Summary of results of SA with BDe score over the data sets of Table 3.2.

Parameters	Precision (%)	Recall (%)	Run-time (in secs.)
k=5,w=5	17.1 ± 6.2	42.5 ± 6.9	9197.0
k=5,w=10	8.7 ± 3.3	23.9 ± 4.8	9760.9
k=10,w=5	12.2 ± 3.7	39.0 ± 8.5	9522.7
k=10,w=10	7.5 ± 3.6	25.9 ± 10.0	9816.1

sponding average run-times. The results are reported in (mean $\pm$ std. dev.) format. These are the best results we could achieve with SA in reasonable time (about 2.5 hours). For this we used the exponentially decreasing cooling schedule with parameter $T_0 = 10^6$. We were unable to extract any meaningful performance on these data sets using SA (despite trying a wide range of parameter tuning schedules). The precision and recall almost never exceeded 20% and 50% respectively even after letting each optimization run for more than 2.5 hours. These run-times were typically 5-10 times longer than the corresponding run-times for SC and EDN (which also delivered substantially better precision and recall performance). Based on our experiments, we concluded that SA was unsuitable for learning structure from our inter-dependent inhomogeneous poisson model of spiking neuronal networks. For these reasons, in this section, we provide full experimental results and comparisons only for SC and EDN based methods and omit the detailed results of SA to avoid clutter.

Quality of inferred networks

Fig. 3.5 shows a comparison of the performance of EDN and SC on the 15 data sets of Table 3.2 for different choices of k (maximum number of parents) and w (size of history window). The precision and recall for a given data set are represented as bars along the same horizontal line (the light color bar represents SC and the dark color bar represents EDN). For example, for the $k = 5$ $w = 5$ case, both precision and recall of EDN were 100% for 10 (out of the 15) data sets. In case of SC, although precision was 100% in 11 data sets, the recall was between 95% and 98% for all the data sets. From the results reported for different k and w values, we can see that (on most data sets) EDN consistently outperformed SC in terms of precision and recall. This is not surprising because the network structures injected into these data sets were predominantly excitatory. The results also highlight the bias of EDN towards high precision performance (sometimes, at the cost of some loss in recall). This is primarily due to the conditional mutual information checks (which, in case of EDNs, is feasible because of the typically small number of frequent episodes ending in the same event-type).

Table 3.4 presents a comparison of SC and EDN for the different groups of networks defined in Sec. 3.6.1. We consider each of the network groups described in Figs. 3.4(a)-3.4(d) separately and report the average precision and recall values obtained by SC and EDN across all the 15 data sets of Table 3.2 (for $k = 5$, $w = 5$). The results show that the few edges that EDN

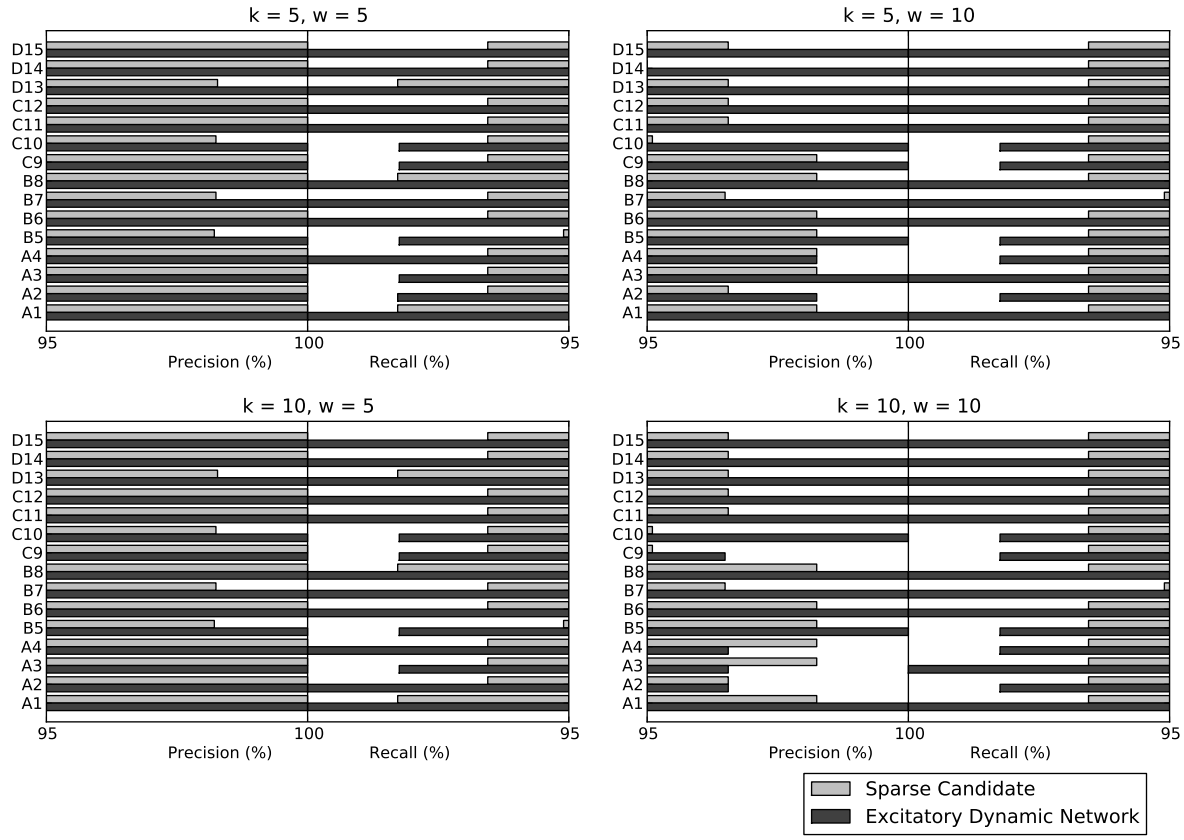


Figure 3.5: Precision and recall comparison of Excitatory Dynamic Network discovery algorithm and Sparse Candidate algorithm for different window sizes, w and max. no. of parents, k.

Table 3.4: Comparison of quality of subnetworks recovered by Excitatory Dynamic Network algorithm and Sparse Candidate algorithm. (k=5,w=5)

	Excitatory Dynamic Network		Sparse Candidate	
Network group	Precision(%)	Recall(%)	Precision(%)	Recall(%)
Causative chains & Higher order structures	100.0	100.0	100.0	99.39
Overlapping causative chains	100.0	99.26	98.42	92.59
Syn-fire chains	100.0	100.0	100.0	99.39
Polychronous circuits	100.0	98.22	99.06	93.77

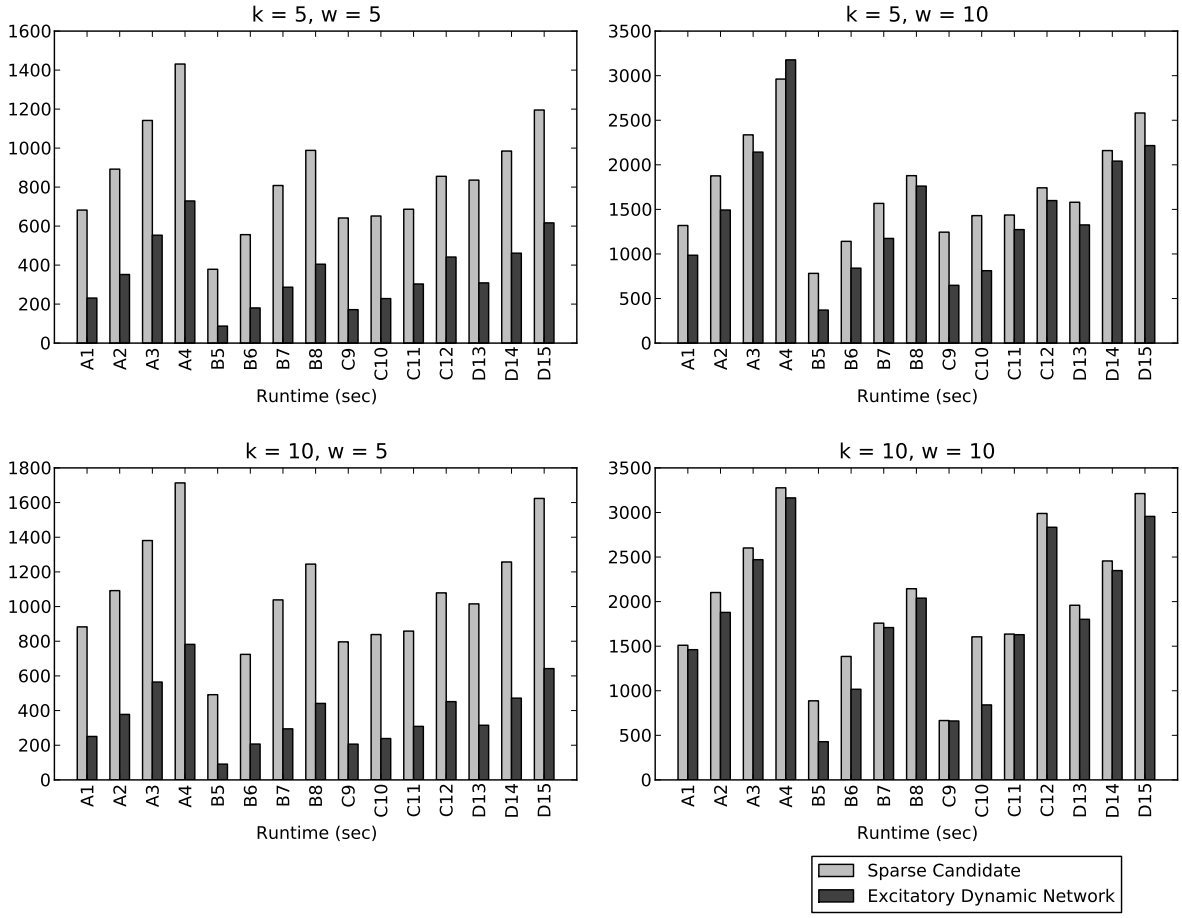


Figure 3.6: Run-time comparison of Excitatory Dynamic Network discovery algorithm and Sparse Candidate algorithm for different window sizes, w and max. no. of parents, k.

missed were from the more complex network structures, namely, overlapping causative chains and polychronous circuits (while causative chains, higher-order structures and syn-fire chains were detected with 100% precision and recall by EDN). The same trends can be observed for SC also, although the deterioration for overlapping causative chains and polychronous circuits is more severe here than in EDN. Similar trends were observed for the other choices of k and w as well.

Run-times

Fig. 3.6 shows the run-time comparisons for EDN and SC on the 15 data sets of Table 3.2. The run-time bars for $(k = 5, w = 5)$ and for $(k = 10, w = 5)$ show a substantial run-time advantage for EDN over SC (by a factor of at least two or more) in all the data sets. For $(k = 5, w = 10)$ and $(k = 10, w = 10)$ also, we observe that EDN is consistently faster than

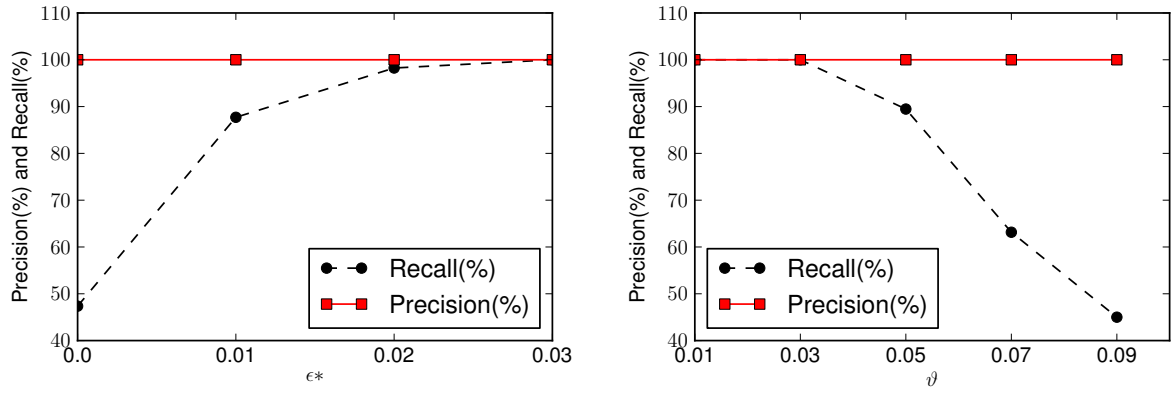


Figure 3.7: Effect of $\epsilon*$ and ϑ on precision and recall of EDN method applied to dataset A1.

SC (although the improvement is much less). This shows that if we use a w value that is near the true size of history window in the network generating the data, EDN has a considerable computational advantage over SC (and this advantage reduces for larger values of w).

Effect of parameters

In Sec. 3.3.1, we discussed the role of ϑ and ϵ^* in fixing the frequency threshold ($P_{min}\Phi_{min}$) for frequent episode discovery. We argued that while the threshold decreases as ϵ^* increases, it increases as ϑ increases. In Fig. 3.7 we plot the effect of changing ϵ^* and ϑ on the performance of our EDN learning algorithms. The figure show results obtained on the A1 data set. Here again, we see the bias of EDN toward high precision – the plots show the precision staying at 100% even as ϵ^* and ϑ are varied. The recall however is more sensitive to the choice of parameters, showing an increasing trend with ϵ^* and a decreasing trend with ϑ. These results are consistent with our theoretical understanding of how frequency threshold changes with ϵ^* and ϑ. Eventually, for sufficiently high ϵ^* and for sufficiently low ϑ the recall starts to approach 100%. (Similar trends were observed in other data sets as well).

Finally, we describe our choice of EDN parameters that were used to generate the results reported in Figs. 3.5-3.6 and in Table 3.4. For A-group and D-group data sets we used $\epsilon^* = 0.03$ and $\vartheta = 0.03$. The conditional probability ρ varies in the C-group data sets, and hence, for these data sets, we used a lower mutual information threshold $\vartheta = 0.02$ and kept $\epsilon^* = 0.03$ (as before). The B-group data sets were the more tricky cases for our EDN algorithm since the base firing rates vary in this group. For B5-B6 (which have lower base firing rates) we used $\epsilon^* = 0.02$ and for B7-B8 (which have higher base firing rates) we used $\epsilon^* = 0.03$. The mutual information threshold for all the B-group data sets was set at $\vartheta = 0.03$.

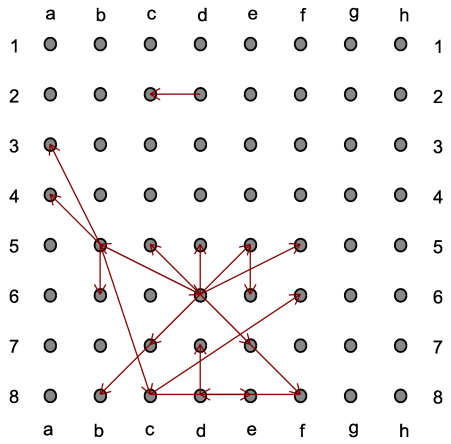


Figure 3.8: DBN structure discovered from first 10 min of spike train recording on day 35 of culture 2-1 [1].

3.6.4 Results on MEA data

Multi-electrode arrays provide high throughput recordings of the spiking activity in neuronal tissue and are hence rich sources of event data where events correspond to specific neurons being activated. We used data from dissociated cortical cultures gathered by Steve Potter's laboratory at Georgia Tech [1] over several days.

In our first experiment, we estimated EDNs from 10 minute-chunks of recordings using parameters $k = 2$, $w = 2$, $\vartheta = 0.001$ and $\epsilon^* = 0.02$. By way of example, we visualize one such network that we estimated in Fig. 3.8. This network obtained from the first 10 minutes of the spike train recording on day 35 of culture 2-1 reflects the sustained bursts observed in this culture by [1]. In order to establish the significance of the networks discovered we ran our algorithm on several surrogate spike trains generated by replacing the neuron labels of spikes in the real data with randomly chosen labels. These surrogates break the temporal correlations in the data while still preserving the overall summary statistics. No network structures were found in such surrogate sequences. We are currently in the process of characterizing and interpreting the usefulness of such networks found in real data.

In our second experiment using recordings from real cortical cultures, we compared the networks estimated using SC and EDN on different slices of data. The results obtained on 4 such slices are reported in Table 3.5. For each data slice, Column 1 of the table reports the number of edges that were common in the outputs of EDN and SC. The number of unique edges found by each of the methods is listed in columns 3 and 4. The figures in brackets describe the corresponding number of unique edges computed as a *percentage* of the total edges found by EDN and SC respectively. The results show that considerable portions of the networks discovered (30% or more of edges found by either method) were actually common under EDN and SC. However, about 50-70% of the edges reported by EDN were

Table 3.5: Comparison of networks discovered by EDN algorithm and SC algorithm on 4 slices of spike train recording (each 10 min long) taken from datasets 2-1-32 to 2-1-35 [1]. Parameters $k = 2$ and $w = 2$ used for both EDN and SC.

	No. of common edges	No. of unique EDN edges	No. of unique SC edges
Data Slice 1	25	59 (70.24%)	30 (54.55%)
Data Slice 2	24	53 (68.83%)	31 (56.36%)
Data Slice 3	30	45 (60.00%)	26 (46.43%)
Data Slice 4	29	41 (58.57%)	27 (48.21%)

unique, and similarly, 45-55% of the edges reported by SC were unique. This suggests that EDN and SC discover different kinds of networks from the same data sets (which is not surprising since EDN and SC restrict the network topologies in different ways and optimize for different criteria). The common edges correspond to *strong excitatory* connections, the unique edges in EDN represent relatively *weaker excitatory* connections and the unique edges in SC correspond to other *non-excitatory* connections (such as inhibitions or excitation-in-absence). Thus, EDN and SC infer complemntary relationships from time-stamped event sequences.

Next, we assess the quality of excitatory networks discovered by EDN and SC in the MEA data. Toward this end, we use a standard scoring function (such as BIC) to compare the parent sets returned by EDN and SC. For each data set, we compute the aggregate BIC score over all nodes for which the EDN returned non-empty parent sets and compare it with the corresponding BIC score for the same nodes but with parents as per the SC network. The comparison is reported in Fig. 3.9. In Fig. 3.9(b) observe that the run-times of SC are roughly 2-6 times worse than for EDN, while the strengths of the excitatory connections discovered (in terms of aggregate BIC scores) are within 10% of the corresponding scores in SC (see Fig. 3.9(a)). Recall that our theoretical results guarantee that the EDN algorithm finds the *optimal* network, over the class of all excitatory networks. Our results here show that, these excitatory networks discovered are also comparable in terms of connections strengths with the output of SC (which searches over non-excitatory networks as well).

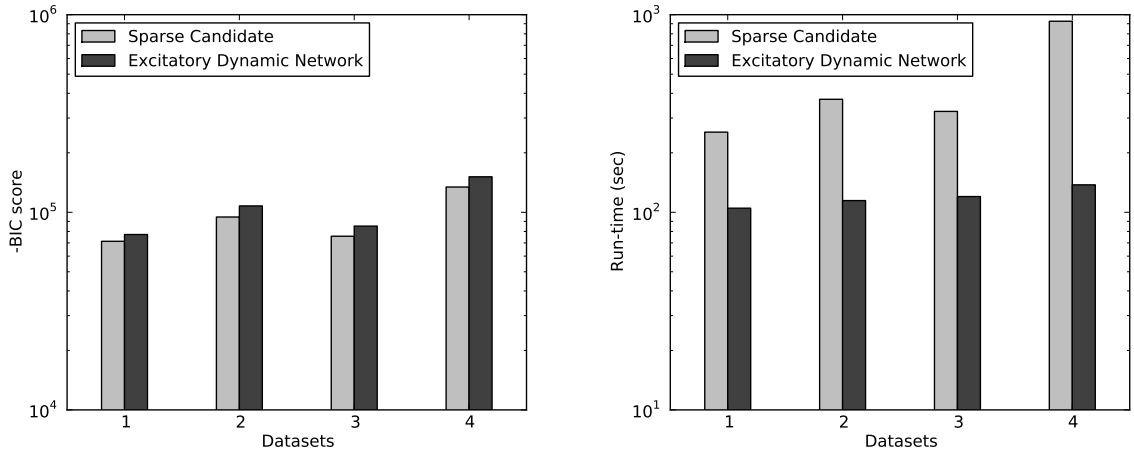
3.6.5 Results on fMCI data

Another relatively new technique of recording activity of multiple neurons simultaneously is functional Multi-neuron Calcium Imaging (fMCI). fMCI is an imaging technique where the neurons are loaded with calcium fluorophores. Their electrical activity causes fluorescence which is recorded using confocal microscopy. In the second set of experiments we analyze the recordings from rat hippocampal slices made publicly available by Yuji Ikegaya's group [2].

In Table 3.6, we present a comparison of the networks discovered by EDN and SC. Like

Table 3.6: Comparison of networks discovered by EDN algorithm and SC algorithm in calcium imaging spike train data (Source: [2]). Parameters: Mutual information threshold used for EDN = 0.0005 and for both EDN and SC, k=5 and w=5 were used.

	No. of common edges	No. of unique EDN edges	No. of unique SC edges
DATA_000	12	24 (66.67%)	1 (7.69%)
DATA_001	8	11 (57.89%)	10 (55.56%)
DATA_002	30	14 (31.82%)	43 (58.90%)
DATA_003	8	20 (71.43%)	5 (38.46%)
DATA_004	-	-	-
DATA_005	12	24 (66.67%)	1 (7.69%)
DATA_006	1	3 (75.00%)	1 (50.00%)
DATA_007	50	32 (39.02%)	57 (53.27%)
DATA_008	50	8 (13.79%)	49 (49.49%)
DATA_009	1	1 (50.00%)	0 (0.00%)
DATA_010	5	9 (64.29%)	3 (37.50%)
DATA_011	15	21 (58.33%)	9 (37.50%)
DATA_012	3	5 (62.50%)	1 (25.00%)
DATA_013	8	2 (20.00%)	1 (11.11%)
DATA_014	19	24 (55.81%)	13 (40.62%)
DATA_015	3	6 (66.67%)	0 (0.00%)
DATA_016	3	3 (50.00%)	0 (0.00%)
DATA_017	26	29 (52.73%)	9 (25.71%)
DATA_018	48	42 (46.67%)	46 (48.94%)
DATA_019	43	32 (42.67%)	34 (44.16%)
DATA_020	18	12 (40.00%)	7 (28.00%)
DATA_021	9	3 (25.00%)	1 (10.00%)
DATA_022	6	15 (71.43%)	2 (25.00%)
DATA_023	49	48 (49.48%)	53 (51.96%)
DATA_024	11	21 (65.62%)	1 (8.33%)
DATA_025	46	51 (52.58%)	23 (33.33%)
DATA_026	62	86 (58.11%)	82 (56.94%)
DATA_027	41	40 (49.38%)	45 (52.33%)
DATA_028	-	-	-
DATA_029	13	24 (64.86%)	4 (23.53%)
DATA_030	32	13 (28.89%)	25 (43.86%)



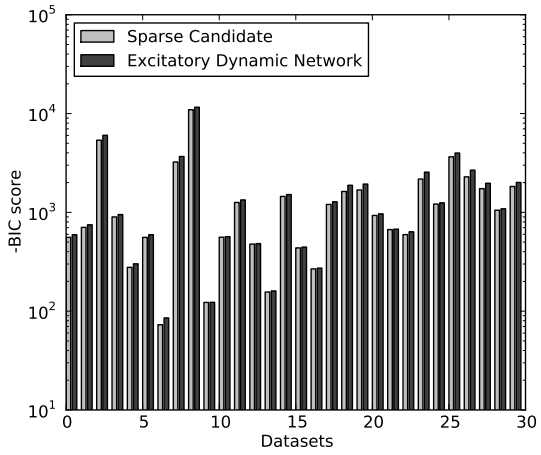
(a) BIC score comparison

(b) Runtime comparison

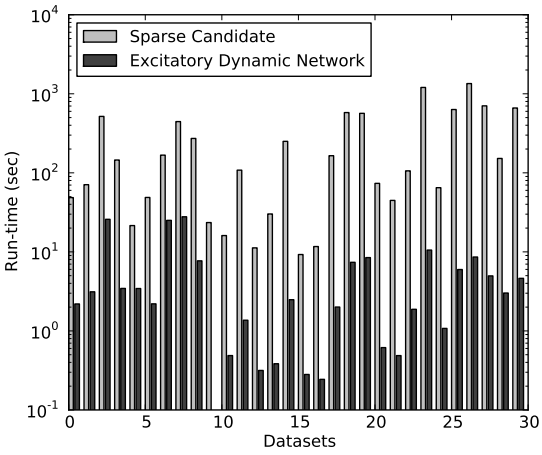
Figure 3.9: Comparison of networks discovered by EDN algorithm and SC algorithm in 4 slices of spike train recording (each 10 min long) taken from datasets 2-1-32 to 2-1-35 [1]. Parameters used: Mutual information threshold for EDN = 0.01 and for both EDN and SC, $k = 2$ and $w = 2$ were used.

the MEA results presented earlier (in Table 3.5), Table 3.6 has 4 columns – the first column describes the name of the data slice, the second column shows the number of edges commonly discovered by SC and EDN, the third column lists the number of unique EDN edges and the fourth column lists the number of unique SC edges. In columns 3 and 4, the numbers in brackets describe the corresponding quantities as percentages of total number of EDN and SC edges. There is no clear pattern in the result – in some data sets, like in DATA_007 and DATA_026, the proportion of common edges is reasonably high, while in some others, like in DATA_001, this proportion is small (Results are not reported for DATA_004 and DATA_028 since, for these data sets, neither algorithm converged even after running for over three hours). These results again show that EDN and SC often yield different Bayesian networks. While SC discovers strong connections between nodes over an unconstrained class of networks, EDN focusses attention only on excitatory networks (a restriction that is natural in applications like neuroscience).

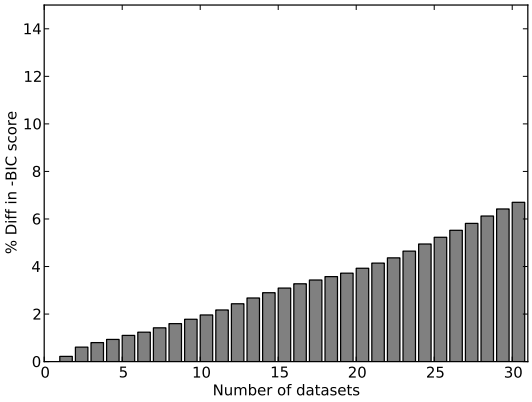
Finally, we compare strengths of the connections discovered as we have done in case of MEA data. For this, we compare aggregate BIC scores (and run-times) for the excitatory networks (under EDN) with the scores (and run-times) for corresponding subnetworks under SC. The results are shown in Fig. 3.10 for an MI threshold of 0.0005 (for EDN). Fig. 3.10(a) shows a bar graph for the corresponding BIC scores under both methods, while Fig. 3.10(c) shows the corresponding cumulative average plot. The graphs show that the BIC scores achieved under EDN and SC are consistently very close for all data sets. For example, in atleast 25 (out of 29) data sets the BIC scores differ by less than 5%. This shows that the excitatory networks



(a) BIC score comparison



(b) Run-time comparison



(c) % Diff. in BIC score

Figure 3.10: Comparison of networks discovered by EDN algorithm and SC algorithm in calcium imaging spike train data [2]. Parameters used: Mutual information threshold for EDN = 0.0005 and for both EDN and SC, k=5 and w=5 were used.

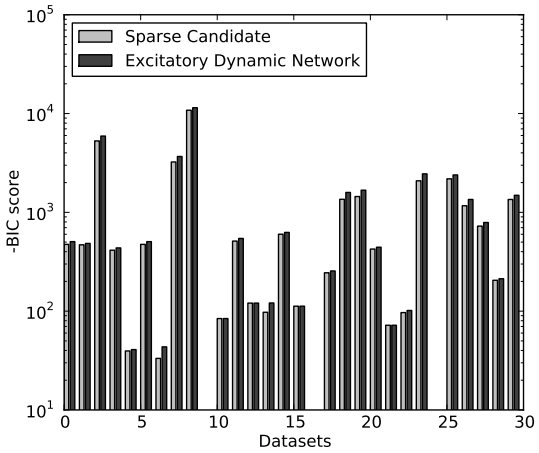
discovered by EDN, although different from the networks obtained through SC, consist of connections that are of roughly the same strength (in terms of BIC scores). Moreover, these strong excitatory circuits are discovered very fast through EDN (compared to the run-times in SC shown in Fig. 3.10(b)). On the average, EDN runs more than 70 times faster than SC (with the minimum and maximum run-time advantages being 6 times and 270 times respectively).

To illustrate the effect of MI threshold on the EDN algorithm, we repeated our experiment for two higher thresholds of 0.001 and 0.002. The corresponding results are shown in Figs. 3.11 and 3.12. First, as can be expected, with increasing MI threshold, excitatory networks are discovered in fewer data sets. EDN found excitatory networks in 27 data sets at a threshold of 0.001 and in 13 data sets at a threshold of 0.002 (compared to finding excitatory networks in 29 datasets at a threshold of 0.0005). Second, the aggregate BIC scores for EDN became marginally poorer at higher thresholds. For example, the number of data sets on which BIC scores for EDN and SC differed by less than 5% fell to 22 at a threshold of 0.001 (see Fig. 3.11(c)) and to 6 at a threshold of 0.002 (see Fig. 3.12(c)). However, the corresponding run-time advantage of EDN became more exaggerated (with EDN being close to 90 times faster, on average, for both thresholds 0.001 and 0.002). All these results demonstrate that learning optimal excitatory networks is considerably more efficient than learning unrestricted DBNs of comparable connection-strengths. Thus, either when the application itself motivates use of excitatory networks (as is the case in neuroscience), or when learning unrestricted DBNs requires more computation than is acceptable (or both), excitatory networks (or EDNs) become relevant for learning optimal temporal networks from time-stamped event sequences.

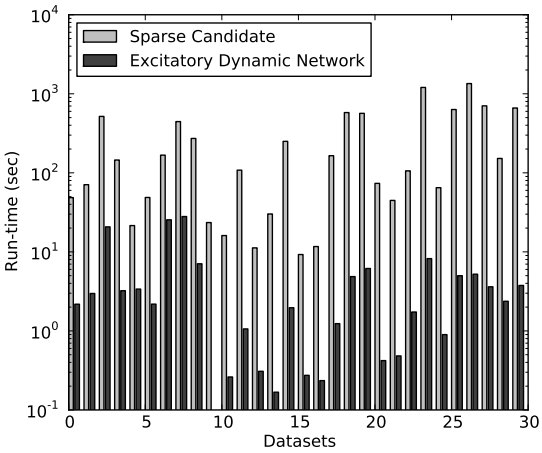
In all of our experiments above, although we have diligently compared our approach to the Sparse Candidate algorithm, the latter is by no means the ground truth for the underlying network. On the contrary, it is interesting that we achieve comparable values for BIC (as the Sparse Candidate algorithm) without directly optimizing for it. These results lead to the radical suggestion that perhaps our approach can supplant SC as a good starting point to find DBNs in general, inspite of its seemingly narrow focus on EDNs, and with significant gains in runtime. Further work is necessary to explore this possibility, especially into the sufficiency of the model class of EDNs for practical application problems and the constraints imposed by the CPTs.

3.7 Discussion

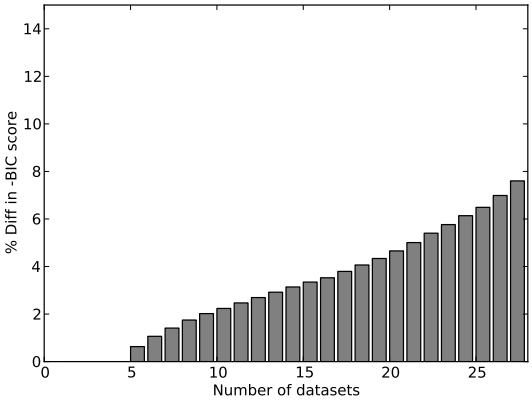
Our work marries frequent pattern mining with probabilistic modeling for analyzing discrete event stream datasets. DBNs provide a formal probabilistic basis to model relationships between time-indexed random variables but are intractable to learn in the general case. Conversely, frequent episode mining is scalable to large datasets but does not exhibit the rigorous probabilistic interpretations that are the mainstay of the graphical models liter-



(a) BIC score comparison

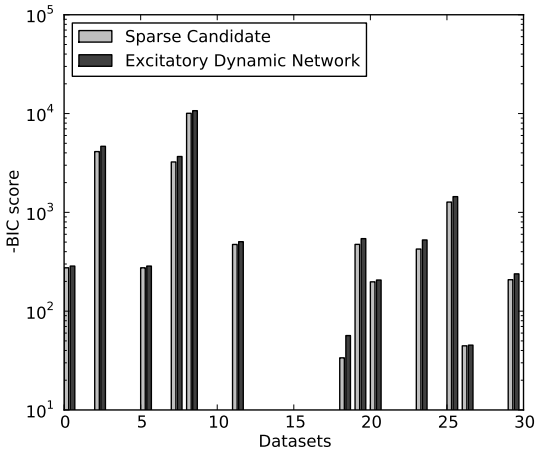


(b) Run-time comparison

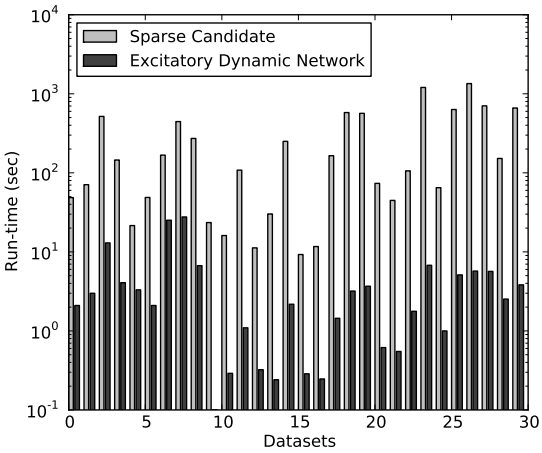


(c) % Diff. in BIC score

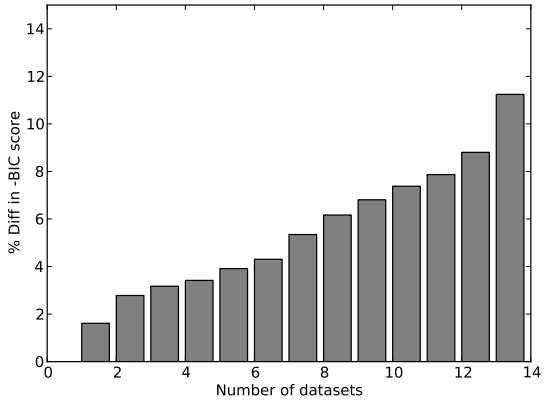
Figure 3.11: Comparison of networks discovered by EDN algorithm and SC algorithm in calcium imaging spike train data [2]. Parameters used: Mutual information threshold for EDN = 0.001 and for both EDN and SC, k=5 and w=5 were used.



(a) BIC score comparison



(b) Run-time comparison



(c) % Diff. in BIC score

Figure 3.12: Comparison of networks discovered by EDN algorithm and SC algorithm in calcium imaging spike train data [2]. Parameters used: Mutual information threshold for EDN = 0.002 and for both EDN and SC, k=5 and w=5 were used.

ature. We have presented the beginnings of research to relate these two diverse threads and demonstrated its potential to mine excitatory networks with applications in spike train analysis.

Two key directions of future work are being explored. The excitatory assumption as modeled here posits an order over the entries of the conditional probability table but does not impose strict distinctions of magnitude over these entries. This suggests that, besides the conditional independencies inferred by our approach, there could potentially be additional 'structural' constraints masquerading inside the conditional probability tables. We seek to tease out these relationships further. A second, more open, question is whether there are other useful classes of DBNs that have both practical relevance (like excitatory circuits) and which also can be tractably inferred using sufficient statistics of the form studied here.

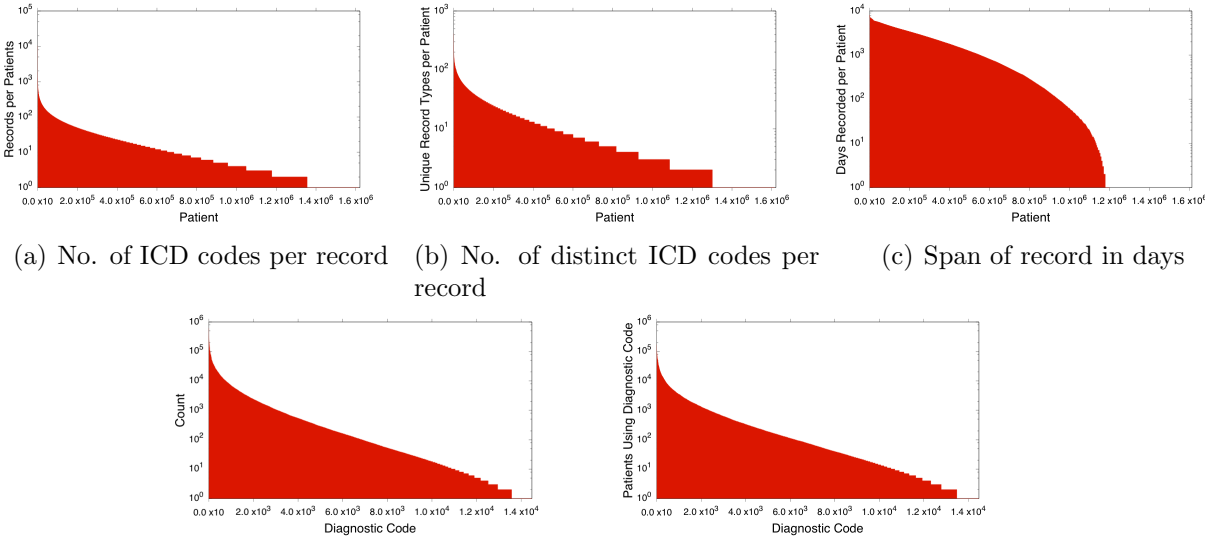
Chapter 4

Mining temporal event sequences from electronic medical records

The increased use of electronic medical records (EMRs) to capture standardized patient information creates an opportunity to better understand both patterns of illness among similar patients, as well as patterns of care within and across medical systems. While the first allows us to identify "fingerprints" of clinical symptoms that may help us capture the progression toward catastrophic clinical transitions, the second can help us understand how diagnoses and procedures are applied within and across different clinical settings.

Two primary categories of information embedded in an EMR are referred to as ICD and CPT codes. ICD 9 (International Classification of Diseases and Related Health Problems, v9) is a popular classification system used in health care which is also heavily used for a wide variety of research activities [94]. This system is primarily intended to code signs, symptoms, injuries, diseases, and conditions. CPT (Current Procedural Terminology) refers to a similar categorization of medical procedures performed on a patient. CPT codes are 5 digit numeric codes, which are published by the American Medical Association. The purpose of this coding system is to provide uniform language that accurately describes medical, surgical, and diagnostic services (including radiology, anesthesiology, and evaluation/management services of physicians, hospitals, and other health care providers). There are about 20,000 distinct ICD 9 codes and about 10,000 CPT codes in use today. Thus an electronic medical record consists, in its most basic sense, of an interleaved sequence of diagnostic (ICD) and procedure (PCT) codes assigned to the patient every time he/she received care along with other associated test reports and doctor's notes.

An EMR is intrinsically temporal in nature yet research on temporal data mining in medical records is scarce. This work seeks to fill this void. Our goal is to extract clinically relevant *sequences* of event codes embedded in the patient histories of more than 1.6 million subjects of the University of Michigan's health system. A straightforward implementation of sequential mining approaches [8] does help provide one level of data reduction, but the resulting



(a) No. of ICD codes per record (b) No. of distinct ICD codes per record (c) Span of record in days

(d) No. of occurrences of an ICD code (e) No. of patients assigned an ICD code

Figure 4.1: Plots of various distributions seen in the real EMR data.

sequences are still too many for human consumption because they are permutations or nearly permutations of each other representing different serializations of the same sets of codes. For instance, it is more likely to see a diagnosis of cardiac ischemia before an aortic coronary bypass, but there are patients for which these occur in the opposite order. It is also not satisfactory to simply lump together codes on the evidence of multiple permutations, because not all sequences are equally likely. We therefore present an approach to derive a partial order from frequent sequences of codes, along with a tool called EMRView that supports exploration of these orders based on a physician's selections of supporting sequences.

4.1 Initial Exploration

After obtaining approval from the Institutional Review Board at the University of Michigan Medical School, we organized a dataset of information about 1.6 million patients in the health system. The actual medical records contained about 100 million time stamped ICD 9 and CPT 4 codes and also other categories of data such as doctor's notes, test results, prescriptions and x-ray images. There are two main concerns in using these other categories of data. The first is to ensure the privacy of the patients concerned. With freeform text this is a very hard problem to address. The second concern pertains to transcribing the handwritten notes into electronic form before they can be analyzed. Hence we began our initial focus on the encoded procedure and diagnostic codes.

First, we replaced patient medical record numbers with artificial research identifiers not tied to the medical record. Time intervals were then used to replace specific dates (thus,

resetting the start of a patient's record to time zero). See Table 4.1 for an example. These codes represented over 10 years of clinical encounters in our electronic medical record system. Data was securely transferred to Virginia Tech for analysis. Specific details of the data are shown in Table 4.2. The range of the number of entries/codes per patient varies from 1 to 10K, i.e., there were many patients with few isolated visits and also patients who have received their entire lifetime of care from the particular healthcare provider. These and other power law behaviors are shown in Fig. 4.1. Some patient records extend over 20 years! In terms of the diagnostic and procedure codes, there are over 40 million and 38 million entries respectively of each type of code in the data.

After a preliminary analysis, we decided to focus exclusively on the diagnostic (ICD) codes. The reason for this decision is that most of the procedure codes (CPT) appear to have been initiated in response to a diagnosis and hence actually served as proxy for conditions. This aided in simplifying the search for patterns downstream although we run the risk of missing infrequent classes of relationships (e.g., the administration of a procedure causing a complication down the road).

Let us denote the medical record of the i^{th} patient as an ordered sequence of diagnostic codes:

$$S_i = \langle (E_1, t_1), \ldots, (E_j, t_j), \ldots, (E_{|S_i|}, t_{|S_i|}) \rangle$$

where E_j is the j^{th} diagnostic code and t_j is the associated time stamp. An entire EMR database consisting of several patient records is denoted by $\mathcal{D} = \{S_1, \ldots, S_{|\mathcal{D}|}\}$.

Table 4.1: Example EMR.

Patient ID	Code	Type	Desc.	Timestamp (in days)
149970	99243	CPT	Office consultation	0
149970	145.9	ICD	Malignant neoplasm of mouth, unspecified.	0
149970	88321	CPT	Microslide consultation.	1
149970	792.9	ICD	Other nonspecific abnormal findings in body substances.	1

Table 4.2: Characteristics of the EMR database.

Property	Value
Number of patients	1,620,681
Number of diagnostic (ICD) codes	41,186,511
Number of procedure (CPT) codes	38,942,605
Max. number of codes in a record	10,430
Min. number of codes in a record	1
Max. span of a record in days	8202 days$\approx$ 22.5 years
Min. span of a record in days	1

One interesting characteristic of the data was observed when we plotted the distribution of time differences between consecutive diagnostic codes across all records (see Fig. 4.2). Note the different modes at multiples of 7 days, an artifact of how followup visits to the health care provider are typically scheduled.

Finally, just as stopword elimination is employed in information retrieval, we conducted 'stopcode elimination' to remove diagnostic codes that are too frequently used. Specifically, codes that surfaced in more than 0.49% of patient records were eliminated (the most frequent code surfaced in 9% of patients).

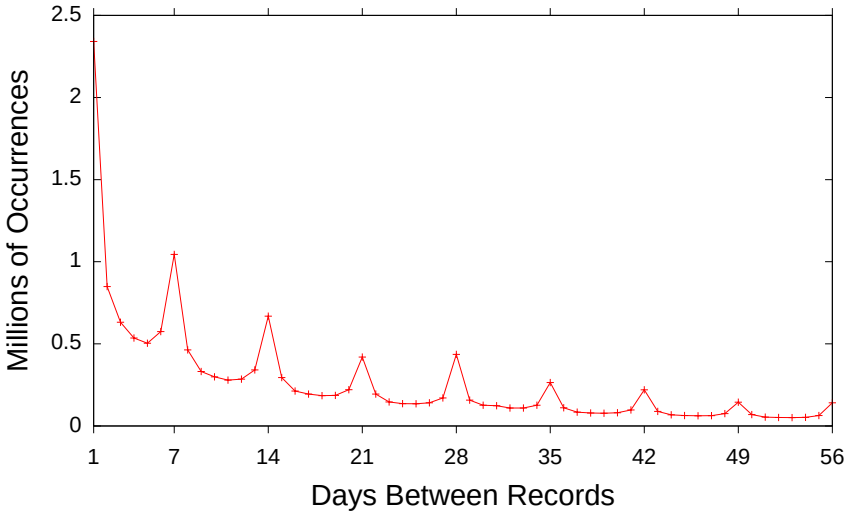


Figure 4.2: Distribution of interarrival times.

4.2 Method overview

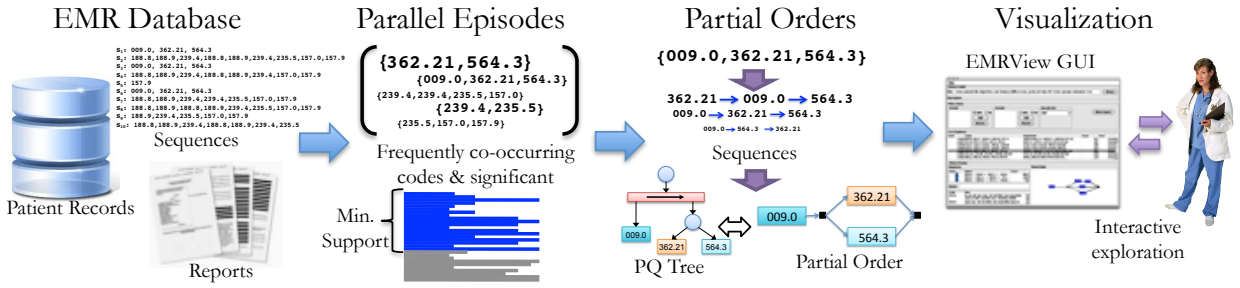


Figure 4.3: Overview of the proposed EMR analysis methodology.

Figure 4.3 provides an overview of our proposed methodology for unearthing partial orders from EMR data. There are three broad stages: i) mining parallel episodes, ii) tracking serial extensions, and iii) learning partial orders.

Mining parallel episodes: Parallel episodes are very similar to itemsets but take temporal information into account in the form of expiry constraints. We present a counting algorithm in Section 4.3.1 that discovers parallel episodes which are frequent above a given threshold. It is well known that patterns with support higher than a threshold may not always be interesting because certain high frequency items can randomly co-occur. Hence, we apply the maximum entropy principle[95] to predict the support of an episode and examine the actual support to impart a score of significance. This score is used to order episodes and is carried out offline.

Tracking serial extensions: After mining a set of frequent parallel episodes, we find order information embedded in these frequently co-occurring sets of diagnostic codes. The intuition behind searching for order is to unearth potential evidence (or lack thereof) for sequentiality in the data. For a n sized frequent parallel episode, there could potentially be $n!$ permutations exhibited in the data.

Learning partial orders: The set of all serial extensions determined in the previous step is compacted into a partial order using a special PQ tree algorithm. PQ trees[96] enable an expressive class of partial orders to be learnt and the user has the flexibility to interactively aid the search for partial orders by selecting the set of extensions to be modeled. This part of the procedure is thus interactive so that the medical professional can iteratively generalize and specialize partial orders in order to arrive at an understanding of the underlying orders.

4.3 Frequent episode mining

In this section we briefly review the frequent episode mining framework and related terminology.

Definition 4.3.1 (General Episode)**.** *An episode α is defined as a tuple $\{V_\alpha, \leq_\alpha\}$, where $V_\alpha \subseteq \mathcal{A}$, $\mathcal{A}$ is the set all symbols (i.e. diagnostic codes). The relation $\leq_\alpha$ defines an ordering of the codes in V_α. A general episode is a partial order over V_α.*

For example, consider an episode α given by $V_\alpha = \{A, B, C\}$ with $A \leq_\alpha B$ and $A \leq_\alpha C$. This defines an episode where A must occur before B and C but (B, C) can occur in any order and can be denoted as $A \rightarrow (BC)$.

Definition 4.3.2 (Serial Episode)**.** *A serial episode is an ordered sequence of symbols/codes. It represents a total order where for any pair $a_i, a_j \in V_\alpha$ and $a_i \neq a_j$, either $(a_i \leq_\alpha a_j)$ or $(a_j \leq_\alpha a_i)$ holds.*

An example of a serial episode is $A \rightarrow B \rightarrow C$ where the relation $\leq_\alpha$ is such that $A \leq_\alpha B$, $A \leq_\alpha C$ and $B \leq_\alpha C$.

Definition 4.3.3 (Parallel Episode)**.** *A parallel episode is an unordered set of symbols/codes i.e. $\leq_v = \phi$.*

A sequence S supports an episode α, if $V_\alpha \subseteq \{A_i : A_i \in S\}$ and for no pair A_j preceding A_i in S, $A_i \leq_\alpha A_j$ holds. In other words, the order of the symbols in S does not violate the precedence order encoded by $\leq_\alpha$. The support of α is the fraction of sequences in D that support it. In addition, an expiry constraint T_X can be specified for episodes which requires that the symbols in S that constitute an occurrence of the episode occur no further than T_X time units apart from each other.

Example 4.3.1. *Consider an EMR dataset of 4 sequences (Fig. 4.4). The highlighted codes constitute occurrences of the parallel episode (ABC) (in S_1,S_2,S_3 and S_4), serial episode $A \rightarrow B \rightarrow C$ (in S_2 and S_3), and the partial order $A \rightarrow (BC)$. Later we shall use this example to illustrate our method.*

S_1: W,U,C,I,B,C,K,A,Y,E,K,J,H,A
S_2: K,J,A,D,K,E,W,K,B,C,R,H
S_3: Q,T,B,J,A,C,O,J,B,A,C,K,F,N,F
S_4: L,M,A,N,C,V,J,H,B,I,U,W,G,B,G,G

Figure 4.4: Database of four EMR sequences S_1, S_2, S_3 and S_4. The time-stamps are not shown. Note that $support(ABC) = 4/4$, $support(A \rightarrow B \rightarrow C) = 2/4$ and $support(A \rightarrow (BC) = 3/4$.

4.3.1 Parallel episodes with expiry

Frequent episode mining follows the Apriori-style itemset mining algorithm: the set of n-size frequent episodes are used to generate the $(n+1)$-size candidate episodes. The counting algorithm then determines the support of each candidate episode over the database of ordered sequences D. Typically, frequent episode mining is carried out over one long sequence of time-stamped events. In a database of ordered sequences, episodes essentially reduce to sequential patterns [14] so that support is defined as the fraction of sequences that have the pattern as a subsequence. Our expiry constraint gives us an additional handle to control the explosion in candidates besides the support threshold.

Algorithm 4 sketches the counting algorithm for parallel episodes with expiry time constraint. The algorithm makes one pass of the database to determine the support of all episodes in the set of candidates. It uses a hash-map data structure to efficiently reference episodes that need to be updated for each code seen in a sequence. The data structure $wait\_count[.]$ tracks the codes that are remaining to complete an occurrence and $T[.]$ stores the latest time stamp of each code. Note that tracking the latest time-stamp ensures that the span of the resulting occurrence is minimum. Hence we correctly track all episode occurrences that satisfy the expiry constraint.

Algorithm 4 Count parallel episodes of size-k.

Input: Set of candidate k size parallel episodes C, Sequence database D, Expiry time constraint T_X, Min support θ.

Output: Set of frequent parallel of episodes with counts.

```
1: A = Set of all codes in episodes in C
2: Initialize waits[A] = φ, ∀A ∈ A
3: for all α ∈ C do
4:    Set count[α] = 0
5:    Set wait_count[α, A] = 1 and T[α, A] = φ,    ∀A ∈ α
6:    Add α to waits[A]
7: for all S ∈ D do
8:    for all (E, t) ∈ S do
9:       for all α ∈ waits[E] do
10:         Update wait_count[α, E] = 0 and T[α, E] = t
11:         for all A ∈ α do
12:            if (t − T[α, A] > T_X) or sid(A)! = S.id then
13:               wait_count[α, E] = 1
14:         if (Σ_{A∈α} wait_count[α, A]) = 0 then
15:            count[α] = count[α] + 1
16:            Reset wait_count(α, A) = 1, ∀A ∈ α, ∀α ∈ C
17: return {α, count(α) : count(α) ≥ θ|D|}
```

4.3.2 Significance of parallel episodes

We utilize the maximum entropy formulation introduced in [97] to assess significance of our episodes. Consider an n-size parallel episode $\alpha = \{E_1, \ldots, E_n\}$, where E_i's are the diagnostic codes. Let Ω be the sample space of all n-size binary vectors and has a distribution $p : \Omega \rightarrow [0, 1]$. In the context of the database D, $p(\omega)$ gives the fraction of the sequences which support the binary vector ω where each 0/1 represents the presence or absence of the corresponding code E_i in a sequence. We compute the empirical distribution q_α over ω using the support of α and those of its sub-episodes. Example 4.3.2 illustrates the use of inclusion-exclusion principle to obtain q_α.

Example 4.3.2 (Empirical distribution)**.** *Consider a 3-size parallel episode ABC. If ABC is frequent, all the sub-episode supports are known. Table 4.3 illustrates the computation of the empirical distribution q_{ABC} using the inclusion-exclusion principle over support (σ) of subepisodes.*

We estimate the distribution p for α using only the support of its sub-episodes. This is done by estimating the maximum entropy distribution p_α^{ME} (defined in Eq 4.1) that represents a log-linear model.

$$p_\alpha^{ME}(\omega) = \frac{1}{Z_\alpha} \exp\Big(\sum \lambda_i f_i(\omega)\Big) \tag{4.1}$$

Table 4.3: Computing the empirical distribution q_α

A B C	Distribution q_{ABC}
0 0 0	$\sigma(\phi) - \sigma(A) - \sigma(B) + \sigma(AB) - \sigma(C) + \sigma(AC) + \sigma(BC) - \sigma(ABC)$
0 0 1	$\sigma(C) - \sigma(AC) - \sigma(BC) + \sigma(ABC)$
0 1 0	$\sigma(B) - \sigma(AB) - \sigma(BC) + \sigma(ABC)$
0 1 1	$\sigma(BC) - \sigma(ABC)$
1 0 0	$\sigma(A) - \sigma(AB) - \sigma(AC) + \sigma(ABC)$
1 0 1	$\sigma(AC) - \sigma(ABC)$
1 1 0	$\sigma(AB) - \sigma(ABC)$
1 1 1	$\sigma(ABC)$

Here each factor $f_i(w)$ is an indicator function:

$$f_i(w) = I(\omega \text{ covers } i^{th} \text{ subepisode of } \alpha)$$

We learn the maximum entropy model under the constraint that the expectations $\mathbb{E}\langle f_i \rangle = \langle \text{support of } i^{th} \text{ proper subepisode of } \alpha \rangle$. We use the KL-divergence measure between the empirical distribution and the maximum entropy distribution estimated using only subepisodes to score the significance of α (Eq 4.2).

$$score(\alpha) = \sum_{\omega} q_\alpha(\omega) \log \frac{q_\alpha(\omega)}{p_\alpha^{ME}(\omega)} \tag{4.2}$$

A higher score implies that the support of the episode is more surprising given the subepisodes. This significance score is used to rank the discovered patterns.

4.3.3 Tracking sequential extensions

After mining the lattice of frequent parallel episodes, we make another pass through the data to record the number of times different linear (serial) extensions of each frequent parallel episode occur in the data. This is illustrated below for the sequences in Example 4.3.1.

Example 4.3.3 (Sequential Extensions)**.** *Consider the parallel episode* (ABC)*. There are* $3! = 6$ *possible permutations or serial extensions of this episode. Now consider the sequences shown in Example 4.3.1, where only 3 of the 6 permutations of* (ABC) *are seen in the data. The supports of these sequences is noted below.*

$$A \to B \to C : 2/4; \qquad A \to C \to B : 1/4; \qquad B \to C \to A : 1/4$$

4.4 Learning partial orders

At this stage, we have mined sets of frequent episodes that are permutations of each other, e.g. $A \to B \to C$, $A \to C \to B$ and $B \to C \to A$. Since the parallel episode (ABC) and

each of these serial episodes is known to be frequent in the data, we aim to find a partial order that describes these sequences (and as few other sequences as possible). A trivial solution is the already obtained null partial order: (ABC) which, besides covering the above three extensions, also covers $B \to A \to C$, $C \to A \to B$, and $C \to B \to A$. A better solution is the partial order $(A(BC))$ which describes only one other sequence in addition to above: $C \to B \to A$.

Mining partial orders is a difficult problem in general because the space of partial orders grows very quickly with alphabet size. As a result, researchers have proposed focusing on special cases of partial orders (e.g., series-parallel) to make the problem tractable [98]. Here, we present a PQ tree approach[99] for finding a recursive partial order over the set of frequent serial episodes over the same set of symbols.

In the context of sequential data, a sequence S is *compatible* with a partial order α, if for no pair a_2 preceding a_1 in S, $a_1 \leq_\alpha a_2$ holds in α. In other words, the order of the elements in S does not violate the precedence order encoded by $\leq_\alpha$. A compatible S is also called an *extension* of $\leq_\alpha$. Sequence S is called a complete extension of $\leq_\alpha$ when S contains all the elements of V_α. In the episodes framework, only a complete extension is considered to support the episodes α. We shall define S' as the subsequence of S that is still a complete extension of α but contains only the elements of V_α. Subsequently, we shall refer to S' as the serial extension of α in the sequence S and denote it as $S' \in \alpha$.

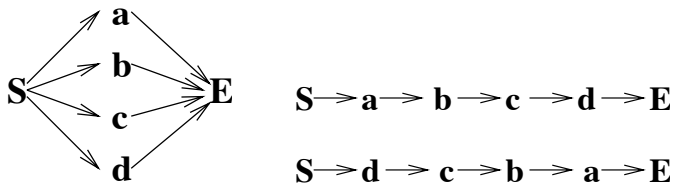
A partial order can be represented by a directed graph where each edge (a_1, a_2) represents the relation $a_1 \leq_\alpha a_2$. Since the relation $\leq_\alpha$ is antisymmetric, it is easy to see that this graph is acyclic (i.e., a DAG). Further, the transitive reduction G'_α of the graph G_α also encodes the partial order α such that when $a_1 \leq_\alpha a_2$, there is a directed path from a_1 to a_2 in G'_α.

4.4.1 Maximal partial order problem

It is easy to see that S is a serial extension of the parallel episode α where $\leq_\alpha = \phi$ for any sequence S on the alphabet V_α. Also if $S \in \alpha(V_\alpha, \leq_\alpha)$, then $S \in \beta(V_\alpha, \leq_\beta)$ where $(\leq_\beta \subset \leq_\alpha)$. Thus it is natural to talk about a *maximal* partial order. We identify the following problem:

Problem 4.4.1. *Given a set $\mathcal{S}$ of sequences S_i, $1 \leq i \leq m$, each of length n defined over the same alphabet, find a maximal partial order α such that all $S_i \in \alpha$ and this does not hold for any partial order β with $V_\alpha = V_\beta$ and $(\leq_\beta \supset \leq_\alpha)$.*

Here maximality implies that: given a pair (a_i, a_j), if a_i precedes d_j in *all* the given m sequences then $(a_i \leq_\alpha a_j)$ must hold. A straightforward approach to solving the problem is to collect all pairs (a_1, a_2) such that a_1 precedes a_2 in *all* sequences $s_i \in \mathcal{S}$ and then construct a (transitive reduction) DAG from these pairs.



(a) α_0 (b) α_1 and α_2

Figure 4.5: Excess in partial orders: (a) One partial order, α_0 and (b) two partial orders, α_1, α_2 representing $s_1 = \langle a\, b\, c\, d \rangle$ and $s_2 = \langle d\, c\, b\, a \rangle$.

4.4.2 Excess in partial orders

Consider $s_1 = \langle a\, b\, c\, d \rangle$ and $s_2 = \langle d\, c\, b\, a \rangle$. Notice that no pair of characters preserve the order in s_1 and s_2, hence the only partial order that s_1 and s_2 do not violate is shown in Figure 4.5(a). However, any permutation of the characters $\{a, b, c, d\}$ also does not violate this partial order.

Let $A(\alpha)$ be the set of all complete extensions $s \in \alpha$. Thus it is clear that the partial order captures some but not all the subtle ordering information in the data and the maximal partial order α of $\mathcal{S}$ is such that $A(\alpha) \supseteq \mathcal{S}$. The gap, $A(\alpha) \setminus \mathcal{S}$ is termed *excess*. One way to handle excess in partial order is allowing for multiple partial orders, say α_1 and α_2 in Figure 4.5(b). Clearly, given S, there has to be a balance between the number of partial orders and the excess in the single maximal partial order α_0.

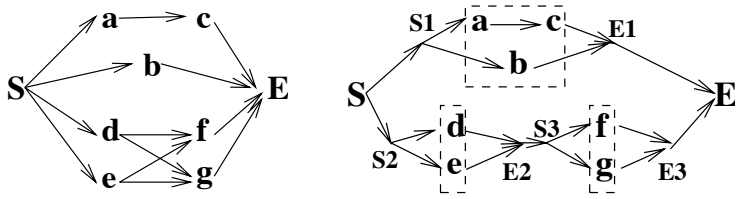
It is easy to see that given n distinct sequences, n partial orders (chains or total orders) give zero excess. Obviously this is not an acceptable solution, and a trade off between excess and the number of multiple partial orders needs to be made. However getting an algorithmic handle on this combinatorial problem, using excess, is non trivial. Other researchers have assumed further restrictions, e.g., Mannila and Meek [98] restrict their partial orders to have an MSVP (minimal vertex series-parallel) DAG. Their aim is to count the size of of $A(\alpha)$ (number of complete extensions of α) since they are looking for a probabilistic model of an underlying generator.

4.4.3 Handling excess with PQ structures

We illustrate this approach starting with an example.

Example 4.4.1. *Let $\mathcal{S} = \{\langle abcdegf \rangle, \langle bacedfg \rangle, \langle acbdefg \rangle, \langle edgfabc \rangle, \langle degfbac \rangle\}$. It is easy to see that the maximal partial order is α_0 shown in Figure 4.6. In α_1, the alphabet is augmented with $\{\mathbf{S_1}, \mathbf{E_1}, \mathbf{S_2}, \mathbf{E_2}, \mathbf{S_3}, \mathbf{E_3}\}$ to obtain a tighter description of $\mathcal{S}$.*

Here the solution to handling excess is to augment the alphabet V_α with $\mathcal{O}(|V_\alpha|)$ characters in the maximal partial order DAG, G_α. Let this new set be V'_α. All complete extension S'



(a) Partial order, α_0 (b) Augmented partial order, α_1

Figure 4.6: Handling excess with PQ structures. Note that α_1 is a specialization of α_0 i.e. if $S \in \alpha_1$, then $S \in \alpha_0$, but not *vice-versa*. If $S = \langle abdefcg \rangle$, then $S \in \alpha_0$ but $S \notin \alpha_1$.

are now on $(V_\alpha \cup V'_\alpha)$ and can be easily converted to S on V_α by simply removing all the symbols $\mathbf{S}_k, \mathbf{E}_k \in V'_\alpha$.

The boxed elements in Figure 4.6 are clusters of symbols that *always* appear together, i.e., are uninterrupted in $\mathcal{S}$. Our scheme exploits this property to reduce excess. We use the systematic representation of these recursive clusters in reducing excess in partial orders. The recursive (order) notation of clusters is naturally encoded by a data structure called the PQ tree [96]. The equivalence of this order notation to a PQ structure, denoted by T_c, along with efficient algorithms is discussed in [99].

A PQ tree is a rooted tree whose internal nodes are of two types: P and Q. The children of a P node occur in no particular order while those of a Q node must appear either in a strict left to right or right to left order. We designate a P node by a circle and a Q node by a rectangle. The leaves of T_c are labeled bijectively by the elements of $\mathcal{S}$. The *frontier* of T_c, denoted by $Fr(T_c)$, is the permutation of the elements of $\mathcal{S}$ obtained by reading the labels of the leaves from left to right. Two PQ trees T_c and $T_{c'}$ are equivalent, denoted $T_c \equiv T_{c'}$, if one can be obtained from the other by applying a sequence of the following transformation rules: (1) Arbitrarily permute the children of a P node, and (2) Reverse the children of a Q node. $\mathcal{C}(T_c)$ is defined as follows: $\mathcal{C}(T_c) = \{Fr(T_{c'}) | T_{c'} \equiv T_c\}$. An *oriented PQ tree* is a tree where all the Q nodes are oriented similarly (in our case left-to-right).

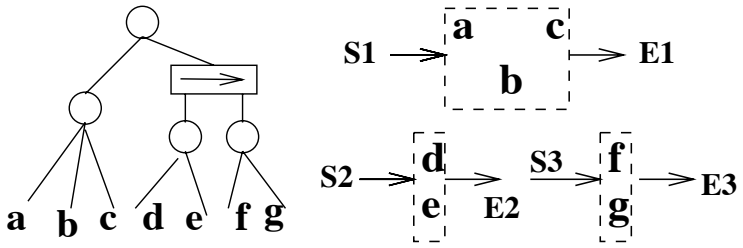
Given $\mathcal{S}$, the *minimal consensus PQ Tree* is a structure that captures all the common clusters that appear in each $s \in \mathcal{S}$. A linear time algorithm, $\mathcal{O}(|\mathcal{S}|\,|V_\alpha|)$, to compute this tree is discussed in [99]. We make a simple modification (not discussed here) in the algorithm to give oriented PQ (oPQ) trees where each Q node follows the strict left-to-right order. The number of augmented alphabet pairs $\mathbf{S}_k, \mathbf{E}_k$, in the augmented DAG correspond to the number of internal nodes in a PQ tree.

To summarize, given $\mathcal{S}$, our scheme works as follows: (1) We first construct the minimal consensus oriented PQ tree T_c of $\mathcal{S}$ using Algorithm 5. (2) For each internal P node we construct an instance of Problem 4.4.1 and construct the maximal partial order. (3) Using the PQ structure of step (1) we combine all the maximal partial order DAGs of (step 2), with augmented characters $\mathbf{S}_k, \mathbf{E}_k$ for each to obtain the augmented maximal partial order.

Figure 4.7 shows the steps on the data of Example 4.4.1. (a) shows the minimal consensus

Algorithm 5 PQ Tree construction

Input: A set of sequences $\mathcal{S}$ each consisting of all symbols in V.
Output: A minimal consensus PQ Tree T_c for $\mathcal{S}$.
1: Compute common intervals in sequences $\mathcal{S} = C_\Pi$ using algorithm in [100].
2: Initialize $T_c = \{\text{Universal tree}\}$
3: **for all** $c \in C_\Pi$ **do**
4: $\quad T_c = REDUCE(T_c, c)$ {See reference [96]}
5: **return** T_c



(a) (b)

Figure 4.7: PQ tree and augmented partial order: (a) The minimal consensus oriented PQ structure for the data in Example 4.4.1. (b) The P nodes as placed in parallel and the Q nodes are placed serially.

oriented PQ tree for the input data. Then the P nodes are arranged in parallel and the Q nodes are arranged serially as shown in (b). For each cluster, the partial order problem is solved and the resulting DAG is shown in Figure 4.6(b).

4.5 EMRView

In this section we describe the features of the EMRView graphical user interface (GUI) software that serves as a partial order exploration tool. The parallel episode mining algorithm runs offline to generate a set of frequent parallel episodes for a user-defined support threshold. The parallel episodes are ordered in decreasing order of the significance score discussed earlier. This allows the user to focus his attention on the most surprising patterns in the data first. The offline mining algorithm also determines the support of the sub-sequences in the EMR sequences that constitute the occurrences of the given parallel episode. We learn a partial order over a subset of these sequences chosen so as to account for a sufficient fraction of the support of the parallel episode. This partial order brings out the ordering information embedded in the EMR data and is displayed via its Hasse diagram.

Pattern loader (Panel 1): The offline result of parallel episode mining together with counts of the sequences constituting the occurrences of each parallel episode is loaded into the visualization tool. The tool also allows reading data in compressed/zip format.

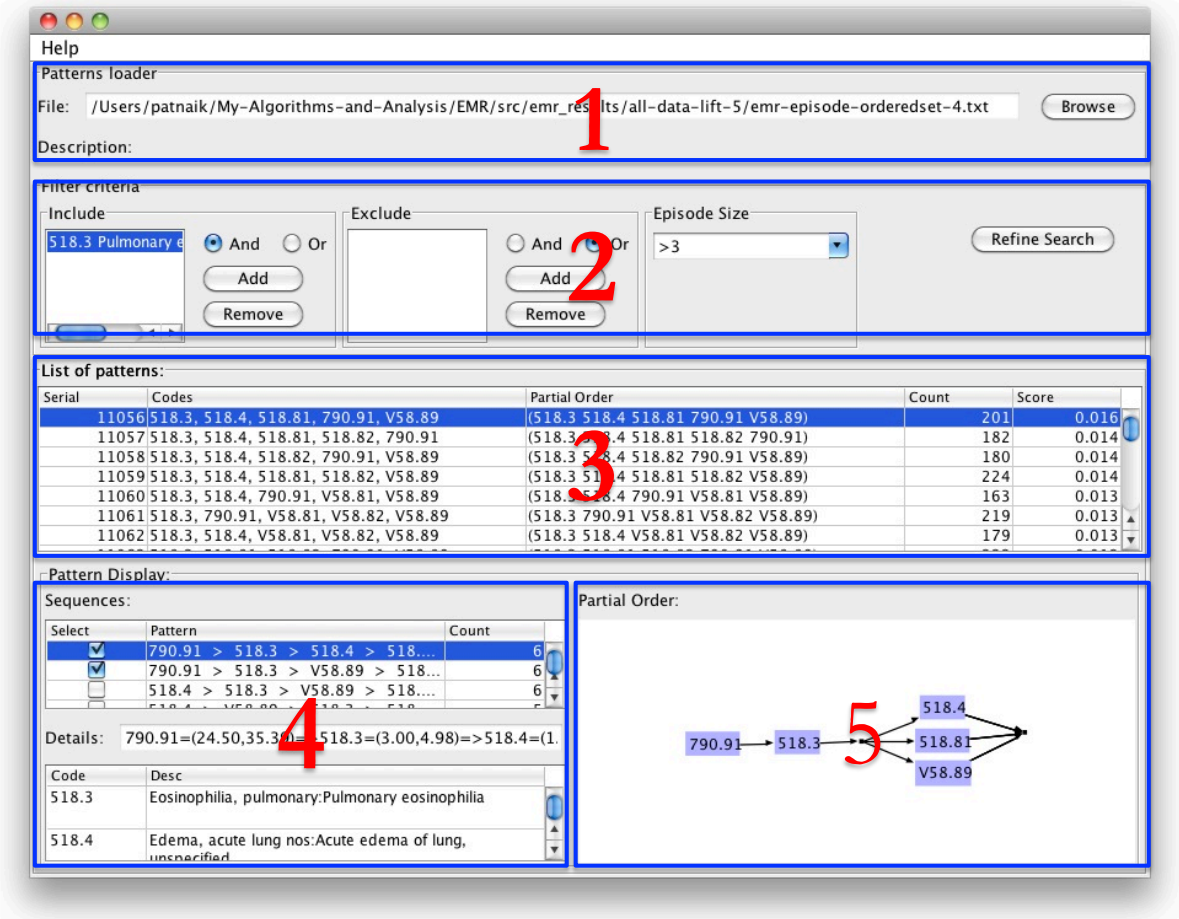


Figure 4.8: EMRView tool showing different panels of the interface.

Filtering results (Panel 2): For the parameters noted in the result section, the number of frequent parallel episodes is over 10,000. Looking through each of the parallel episodes and its supporting subsequences can be a daunting task. The EMRView tool allows the user to filter the list of episodes based on multiple criteria. The user can specify a set of diagnostic codes that must be present in the reported episodes with the option for enforcing all or any of the codes. Similarly the user can specify a set of diagnostic codes that must not be present. Further the user can specify the size of episodes that must be displayed.

A specially designed search dialog is presented to the user to select diagnostic codes. This dialog allows the user to search for the codes by entering either text describing the diagnostic code or the code if available. One could also pick codes by browsing through the hierarchy of the ICD codes. Figure 4.9 shows the search dialog illustrating the different options available for finding and adding diagnostic codes.

Result display (Panel 3): The filtered parallel episodes are displayed here. The first column gives an identifier for the row. The second column shows the list of codes in the

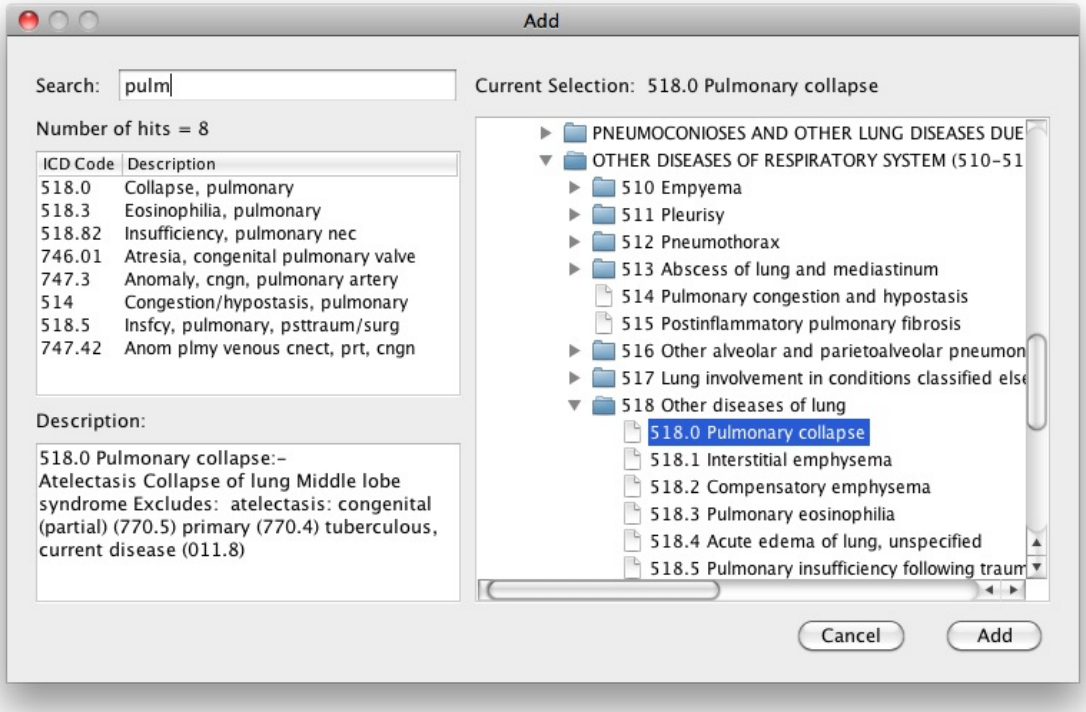


Figure 4.9: EMRView diagnostic code lookup interface.

parallel episode. Since there is no order imposed on the codes, we sort them in alphabetical order. The third column displays a partial order learnt from the set of sequences which together account for 75% of the support of the parallel episode. The same set of sequences are shown selected by default in Panel 4. Note that these sequences are ordered in descending order of support. The forth and fifth columns present the support (or count) and significance scores respectively. The rows are presorted in decreasing order of the significance score.

Pattern display (Panel 4): This panel lists the sequences (more appropriately subsequence of the EMR sequences) that constitute the occurrences of the parallel episode. This panel is populated when the user selects a parallel episode in Panel 3. The sequences are listed in decreasing order of support. By default, sequences are selected until the sum of their support values exceeds a threshold of 75% of the total support of the parallel episode. The rationale is that these are likely the most important orders encountered in the data for the parallel episode under consideration. The partial order learnt from the selected sequences is displayed in the adjacent panel. In addition the user can select/deselect sequences. The partial order is updated online to reflect the selection. This puts the user in control of selecting sequences he deems useful and the algorithm summarizes them into a compatible partial order.

4.6 Results and clinical relevance

For the current study we chose to discover temporal patterns that occurred within a consecutive 200 day (6.5 month) period at support level $\geq 10^{-4}$. We did not identify any truly novel patterns but were able to discover many that reflect the capabilities of our algorithm and the power of our partial order discovery methodology.

Figure 4.10 shows the progression from a general diagnosis of hip pain to a subsequent diagnosis of osteoarthritis and a femur fracture, ultimately requiring a hip replacement. The code 719.68 is rather nonspecific but is typical of diagnosis codes used for routine care, especially when uncertainty exists in the final diagnosis.

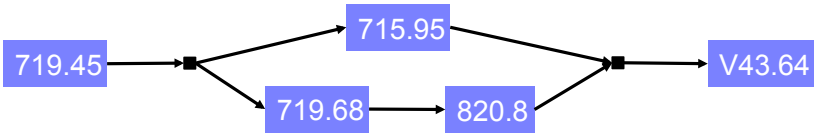


Figure 4.10: A five-sequence pattern demonstrating an initial diagnosis of pelvic pain (719.45) followed by the variable sequence of (a) a poorly defined joint symptom (719.68); (b) a femoral neck fracture (820.8); and (c) pelvic osteoarthritis (715.95), with a final common pathway converging on a hip joint replacement (V43.64).

Another set of patterns related to sinusitis are shown in Figures 4.11(a) and 4.11(b). In both of these sequential patterns, an initial diagnosis (deviated septum or nasal polyps) leads to various chronic sinusitis diagnoses. The sequence in Figure 4.11(a) occurred 102 times among the patient population, whereas the two sequences shown in Figure 4.11(b) occurred collectively 27 times. Similar sequences could also be found in the dataset with varying frequencies.

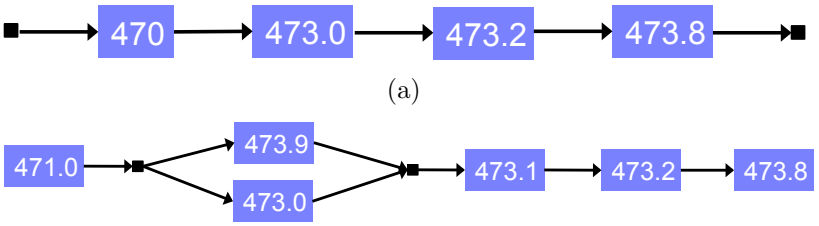


Figure 4.11: Sequences showing events leading to a sinusitis diagnosis. In (a) a deviated septum (470) leads to maxillary sinusitis (473.0), followed by ethmoidal sinusitis (473.2), and then a non-specific sinusitis diagnosis (473.8). In (b) nasal polyps (471.0) leads to maxillary sinusitis (473.0) or a non-specific sinusitis (473.9), followed by frontal sinusitis (473.1), then ethmoidal sinusitis (473.2) and finally another non-specific sinusitis (473.8).

Some sequences converged on a diagnosis or event, whereas others diverged. This can be

seen in Figures 4.12(a) and 4.12(b). Figure 4.12(a) shows events related to arm fractures at a school playground (representing 84 distinct patients), whereas Figure 4.12(b) shows events related to an initial pylenonephritis (kidney infection) event with subsequent diagnoses of calculi (stones) in the kidneys and urinary tracts. In this context, a convergence means requiring a particular diagnosis before continuing to make other diagnoses.

(a) (b)

Figure 4.12: Sequences that converge onto a common event or diverge from a common event. The elements stacked vertically were found to occur interchangeably in the patterns found, suggesting that the specific ordering does not matter. In (a) A fall from playground equipment (E884.0) was associated in time with a lower humerus fracture(812.40) as well as a supracondylar humerus fracture (812.41), but all ended up with a follow-up exam for the healed fracture (V67.4). The diagram in (b) shows an initial pylenonephritis event (590.00) occurring, followed by the diagnoses of kidney calculus (e.g., stone) (592.0), ureter calculus (592.1), and non-specific urinary calculus (592.9).

The clinical relevance of the above patterns is quite immediate. For instance, the presence of osteoarthritis is a recommended, although controversial, indication for performing a hip joint replacement after the occurrence of a femoral neck fracture[101, 102, 103]. It is possible to imagine that someday clinicians may wish to interrogate their local data with EMRView to determine local patterns of care in order to answer questions such as "At our institution, what is our routine process of care for patients with a finding of osteoarthritis who subsequently develop a femoral neck fracture?" A key component in such a query would be to include the temporal concept of "subsequently" so that the timing of the fracture in relation to the other relevant diagnoses can be taken into account.

The observations regarding sinusitis are also interesting, and support the use of EMRView for hypothesis generation. It is well known that maxillary sinus is the most common site for such an infection to occur, followed by the ethmoid sinus (87% vs. 65%, respectively) [104], but it is worthwhile to note that in our observations, for the sequences that contained the codes for both maxillary (473.0) and ethmoidal sinusitis (473.2) the maxillary preceded the ethmoidal in the vast majority of cases. It may be that a severe maxillary sinusitis can often lead to an ethmoid infection with the reverse sequence being less common, or it may simply be that maxillary sinusitis is easier to diagnose and only later is an ethmoid sinusitis diagnosed in the same patient. It is also noteworthy that this temporal pattern occurred when the initial diagnoses was a deviated septum as well as with nasal polyps. It is unclear why this temporal pattern emerged. At least one source has reported that about 75% of all polyps are located in the ethmoid sinuses [105], which raises the question why the sinusitis

was still primarily occurring in the maxillary sinuses first.

Among the other events we reported, the relationship between arm/humerus fractures and falls from school playground equipment is well documented in the literature [106, 107]. Nevertheless, monitoring how such patterns over time change at a specific medical center might allow for the detection of playgrounds that are especially dangerous, or perhaps the success of safety programs implemented to prevent such accidents. The relationship between pyelonephritis (kidney infections) and urinary system calculi also has important implications. As outlined in a recent report, using ultrasound to discover the calculi may be a useful tool when such infections are diagnosed since calculi may be a complicating factor in the management of the infections [108].

4.7 Related work

Many studies have explored temporal patterns in clinical data: some have focused on extracting temporal patterns from natural language descriptions in clinical documents [109, 110] whereas others have used coded administrative data, similar to the data we used [111, 112]. Reis et al. [113] used Bayesian models to predict future risks of abuse based on past episodes. They theorized that such a system could provide an early warning for clinicians to identify patients at high risk for future abuse episodes. Others have used temporal data mining to explore patterns related to specific diseases such as diabetes [114] or specific time frames such as adverse events following immunizations in a large cohort of nearly a million children in Denmark [115].

Plaisant and colleagues [116] built into the Microsoft Amalga software a user interface to help visualize patient specific temporal patterns. This did not involve data mining and the interface itself was significantly different from EMRView. More closely related, however, was the methodology and interface built by Moskovitch and Shahar [117] to identify and visualize temporal patterns in clinical data which was applied to a dataset of about 2000 patients with diabetes. More broadly, several published papers discuss the challenges faced in analyzing EMR data starting from data integration aspects [118] to privacy issues [119]. Interest in the KDD community has blossomed recently, e.g., in [120], a multi label classifier is proposed that learns inter-code relationships from clinical free text. In another study it has been shown that using the entropy of ICD9 codes it is possible to categorize diseases as chronic or acute [121]. Our work is novel in the application to a large scale database of EMRs to mine frequent and significant episodes, in the new partial order discovery algorithm, and in the development of EMRView based on user needs.

4.8 Discussion

Our exercise in using temporal data mining to identify clinically relevant patterns in medical record data did demonstrate the feasibility of the approach. We were able to find hundreds of patterns using real world data that had been collected as a part of routine care. The clinical relevance of the patterns helps to confirm the validity of our mining algorithms.

Our current analysis brings about many future challenges for temporal mining in EMRs. For instance, thus far we have limited the time frame for analysis to 200 days. While this has the advantage, at least theoretically, of yielding greater clinical relevance between events that are temporally correlated, the disadvantage is the danger of missing correlated events that are far apart. Asbestos exposure and subsequent development of lung diseases is one such example [122]. Automatically identifying intervals for analysis is one area of research.

A second major issue with our analysis is that it is easier to obtain the data then it is to explore or assign meaning to the results. There is nothing intrinsic in the data itself to inform if a temporal sequence is novel, important, or even clinically valid. From our initial analysis the results do seem to be valid, but there is no existing database of clinical medicine with which to computationally compare our findings. The more frequently occurring patterns are probably a reflection of an event occurring more frequently among our population. But it may very well be that the less frequently occurring events are the ones that are not well described in the literature and may represent novel discoveries. Finding a scalable approach to explore this space is still an open question.

Last, while we did have nearly 100 million events with which to detect patterns, these codes still represent only a tiny fraction of the clinical parameters that are relevant to each patient. Many clinical details were not present in the dataset that might have a significant impact on the interpretation of the results. It will be important to develop methods to incorporate these additional data into the data mining algorithms to extract more meaningful results in the future.

Chapter 5

Mining frequent patterns in streaming data

As mentioned in the introduction chapter, the problem of discovering interesting patterns from large datasets has been well studied in the form of pattern classes such as itemsets, sequential patterns, and episodes with temporal constraints. However, most of these techniques deal with static datasets, over which multiple passes are performed.

In many domains like telecommunication and computer security, it is becoming increasingly difficult to store and process data at speeds comparable to their generation rate. A few minutes of call logs data in a telecommunication network can easily run into millions of records. Such data are referred to as *data streams* [123]. A *data stream* is an unbounded sequence where new data points or events arrive continuously and often at very high rates. Many traditional data mining algorithms are rendered useless in this context as one cannot hope to store the entire data and then process it. Any method for data streams must thus operate under the constraints of limited memory and processing time. In addition, the data must be processed faster than it is being generated. In this chapter, we investigate the problem of mining frequent patterns under these constraints and demonstrate the effectiveness of our proposed method for a specific pattern class, namely, serial episodes.

In several applications where frequent episodes have been found to be useful, share the streaming data characteristics. In neuroscience, multi electrode arrays are being used as implants to control artificial prosthetics. These interfaces interpret commands from the brain and direct external devices. Identifying controlling signals from brain is much like finding a needle in the hay stack. Large volumes of data need to be processed in real time to be able to solve this problem. Similar situations exist in telecom and computer networks where the network traffic and call logs must be analyzed to detect attacks or fraudulent activity.

5.1 Problem definition

In this section we present the streaming problem more formally. A data stream $\mathcal{D}$ is an infinite sequence of data points. In the case of episodes and partial orders it is an event stream of the form $\mathcal{D} = \langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_i, \tau_i), \ldots, (e_n, \tau_n), \ldots \rangle$, where each event is a pair (e_i, τ_i): e_i is the event type and τ_i is the time of occurrence of the event. (e_n, τ_n) is then the most recent event in the stream. For sequential patterns, the data is an infinite sequence of transactions whereas for motifs it is a continuous time series.

One of the key data mining tasks over a data stream is to find all frequent patterns *in the recent past*. The notion of a recent past can be defined in many different ways. Most studies in data streams focus on one of the following 'window' models: i) the landmark model, ii) the time fading model, or iii) the sliding window model. In the landmark model, one end of the window is fixed at a landmark event like the start of the sequence and the window grows with new arrivals. In this model, all patterns, irrespective of where they are concentrated (in time), receive equal importance. In most applications, however, it is desirable to assign higher importance to patterns that are more recent in time. The time fading model achieves this by assigning higher weight to more recent events. The sliding window model is one in which older data points are discarded as new data arrive, and hence this approach also allows one to focus on the latest patterns. In this work, we focus on a variant of the sliding window model.

Typically in the sliding window model, data in an entire window is loaded into memory and processed. Next, some of the old data is replaced with new arrivals and the processing is repeated on the data in memory. There are efficient ways of avoiding recomputing the results from scratch by using the ones from the previous window. But nevertheless there must be enough space to hold an entire window in memory and thus the findings are limited to this window. In many real applications, we are interested in finding patterns over windows larger than what can fit in memory or be processed at a rate faster than that of data generation. For example, in the telecom setting, we may want to find the most frequent patterns or episodes in the last several weeks while only a few hours of call log data can be stored and processes at a time. Further in the neuroscience application, typical firing rates of neurons observed in the data are about 25-30 Hz. In a network of 1000 neurons, an hour of data can easily run into 100 million spikes ($\approx 1000 \times 30 \times 3600$). From our experiments we see that 1 million spikes together with associated data structures take up roughly 60-70 MB of storage. Extrapolating, an hour of spike train recording would require about 6-7 GB of memory. Although not unrealistic this memory requirement is fairly steep. Mining for frequent patterns in 10 hrs of this data would be infeasible without the use of very powerful computing resources. Finally, runtime limitations further restrict the amount of data that can be processed in real time.

We propose a sliding window model where the window of interest is larger than the space available. Only a portion of the window called a batch can be processed at a time. This

notion is illustrated in Figure 5.1. The window of interest is denoted by W_s. A batch of data B_s, consists of consecutive data points in $\mathcal{D}$ starting at time τ_i and ending at τ_j such that,

$$(s-1)T_b \leq \tau_i \leq \tau_j < sT_b \tag{5.1}$$

where T_b is the span of a batch. Our definition of a batch is time-based. If time is considered to be integer and at most one event is allowed to occur at any time tick then there can be at most T_b events in any batch. For our purposes, we will only assume that the number of events in any batch is bounded and that we have sufficient memory to store all events that occur in a batch.

We now define the window of interest W_s to consist of m consecutive batches ending in the batch B_s, i.e.

$$W_s = \langle B_{s-m+1}, B_{s-m+2}, \ldots, B_s \rangle \tag{5.2}$$

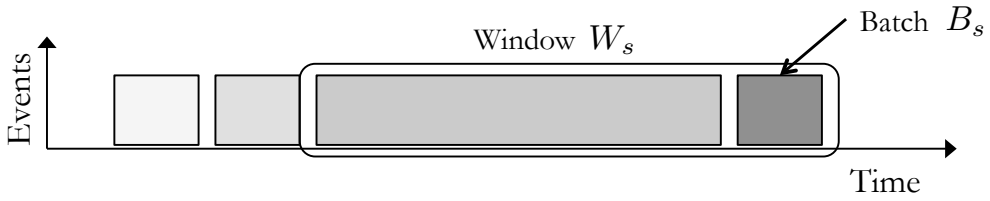


Figure 5.1: Illustration of the proposed sliding window model. In this model only one batch B_s is available for processing and results are desired over the entire window W_s.

We are interested in finding frequent patterns in the most recent window W_s while having access to only the most recent batch B_s. Although our interest is in episodes, we will formulate the problem in a more general pattern mining setting, with two very loose requirements, viz. the i) frequency of a pattern in a window is the sum of its frequencies in each batch in the window and ii) the frequency measure obeys the anti-monotonicity property. Nevertheless, our motivation for the streaming mining setting arises from serial episodes and their application to computational neuroscience datasets. In this light, the neuroscience application will be the driving force behind the different proposed formulations below.

5.2 Related work

Most prior work in streaming pattern mining is related to frequent itemsets and sequential patterns [124, 9, 125, 126]. Some interesting algorithms have also been proposed for streaming motif mining in time-series data [127]. But these methods do not easily extend to other pattern classes like episodes and partial orders. To our knowledge, there has been very little

work in the area of mining patterns in discrete event streams. In this section we discuss some of the existing methods for itemsets, sequential patterns, and motifs.

Karp *et al* proposed a one pass streaming algorithm for finding frequent events in an item sequence [124]. The algorithm, at any given time, maintains a set K of event types and their corresponding counts. Initially, this set is empty. When an event is read from the input sequence, if the event type exists in the set then its count is incremented. Otherwise the event type is inserted into the set K with count 1. When the size of the set K exceeds $\lfloor 1/\theta \rfloor$, the count of each event type in the set is decremented by 1 (and deleted from the set if count drops to zero). The key property is that any event type that occurs at least $n\theta$ times in the sequence is in the set K. Consider an event type that occurs f times in the sequence, but is not in K. Each occurrence of this event type is eliminated together with more than $\lfloor 1/\theta \rfloor - 1$ occurrences of other event types achieved by decrementing all counts by 1. Thus, at least a total of f/θ elements are eliminated. Thus $f/\theta < n$, where n is the number of events in the sequences and hence, $f < n\theta$. This method guarantees no false negatives for a given support threshold. But the space and time complexity of this algorithm varies inversely with the support threshold chosen by the user. This can be a problem when operating at low support thresholds. In [9], this approach was extended to mine frequent itemsets.

Lossy counting constitutes another important class of streaming algorithms proposed by Manku and Motwani in 2002 [125]. In this work an approximate counting algorithm for itemsets is described. The algorithm stores a list of tuples which comprise an item or itemset, a lower bound on its count, and a maximum error term (Δ). When processing the i^{th} item, if it is currently stored then its count is incremented by one; otherwise, a new tuple is created with the lower bound set to one, and Δ set to $\lfloor i\epsilon \rfloor$. Periodically, each tuple whose upper bound is less than $\lfloor i\epsilon \rfloor$ is deleted. This technique guarantees that the percentage error in reported counts is no more than ϵ and it is also shown that the space used by this algorithm is $O(\frac{1}{\epsilon} \log \epsilon n)$ for itemsets. Unfortunately, this method requires operating at very low support threshold ϵ in order to provide small enough error bounds. In [16], the pattern growth algorithm - PrefixSpan [15] for mining sequential patterns was extended to incorporate the idea of lossy counting.

In [126], the authors propose a new frequency measure for itemsets over data streams. The frequency of an itemset in a stream is defined as its maximal frequency over all windows in the stream from any point in the past until the current time that satisfy a minimal length constraint. They present an incremental algorithm that produces the current frequencies of all frequent itemsets in the data stream. The focus of this work is on the new frequency measure and its unique properties.

In [127] an online algorithm for mining time series motifs was proposed. The algorithm uses an interesting data structure to find a pair of approximately repeating subsequences in a window. The Euclidean distance measure is used to measure the similarity of the motif sequences in the window. Unfortunately this notion does not extend naturally to discrete patterns. Further, this motif mining formulation does not explicitly make use of a support

or frequency threshold and returns exactly one pair of motifs that are found to be the closest in terms of distance.

A particular sore point in pattern mining is coming up with a frequency threshold for the mining process. Choice of this parameter is key to the success of any effective strategy for pruning the exponential search space of patterns. Mining the top-k most frequent patterns has been proposed in the literature as a more intuitive formulation for the end user. In Chapter 3, we proposed an information theoretic principle for determining the frequency threshold that is ultimately used in learning a dynamic Bayesian network model for the data. In both cases the idea is to mine patterns at the highest possible support threshold to either outputs the top-k patterns or patterns that satisfy a minimum mutual information criteria. This is different from the approach adopted, for example, in lossy counting where the mining algorithm operates at support threshold proportional to the error bound. Therefore, in order to guarantee low errors, the algorithm tries to operate at the lowest possible threshold.

An episode or a general partial order pattern can be thought of as a generalization of itemsets where each item in the set is not confined to occur within the same transaction (i.e. at the same time tick) and there is additional structure in the form of ordering of events or items. In serial episodes, events must occur in exactly one particular order. Partial order patterns allow multiple orderings. In addition there could be repeated event types in an episode. The loosely coupled structure of events in an episode results in narrower separation between the frequencies of true and noisy patterns (i.e. resulting from random co-occurrences of events) and quickly leads to combinatorial explosion of candidates when mining at low frequency thresholds. Most of the itemset literature does not deal with the problem of candidate generation. The focus is on counting and not so much on efficient candidate generation schemes. In this work we explore ways of doing both counting and candidate generation efficiently. Our goal is to devise algorithms that can operate at as high frequency thresholds as possible and yet give certain guarantees about the output patterns.

5.3 Pattern mining in event streams

We present the notation used in our formulation in Table 5.1. The pattern mining problem we aim to address is defined formally in this section. We begin our discussion with the more general top-k frequent pattern mining problem and then specialize it based on the constraints of streaming setting.

Definition 5.3.1 (Top-k frequent patterns)**.** *This is the set of patterns in which the frequency of each pattern is greater than or equal to the frequency of the k^{th} most frequent pattern (denoted by $f_k^{W_s}$ for a window W_s or f_k^s for a batch B_s). The pattern frequencies are defined over the corresponding window W_s or batch B_s.*

Note that this definition allows that there can be more than k patterns in the top-k set.

Table 5.1: Notation used in streaming problem formulation.

Notation	Definition
(e_i, τ_i)	An event of type e_i that occurred at time τ_i
$\mathcal{D}$	The data stream $\mathcal{D} = \langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_i, \tau_i), \ldots, (e_n, \tau_n), \ldots \rangle$
B_s	A batch of data, start time: τ_i, end time: τ_j, s.t. $(s-1)T_b \leq \tau_i \leq \tau_j < sT_b$
W_s	A window consisting of m batches. $W_s = \langle B_{s-m+1}, B_{s-m+2}, \ldots, B_s \rangle$
m	No. of batches in a window
T_b	Span of a batch measured in units of time
k	The k for top-k mining
ℓ	The size of the pattern reported. For example (A, B, C, D) is a size-4 itemset
$f^s(\alpha)$	Frequency of a pattern α in the batch B_s
$f^{W_s}(\alpha)$	Frequency of a pattern α in the window W_s
f_k^s	Frequency of k^{th} most frequent pattern in the batch B_s
$f_k^{W_s}$	Frequency of k^{th} most frequent pattern in the window W_s
δ	Parameter in bounded rate change assumption
v	Parameter in (v, k) persistence formulation, $1 \leq v < m$

Problem 5.3.1 (Find the top-k over window W_s)**.** *Mine top-k most frequent patterns of size ℓ in a window W_s of the data stream where $k(>0)$ and $\ell(>0)$ are user defined parameters.*

This problem can be solved exactly if we have sufficient memory to store and process (i.e., make multiple passes over) all the events that arrive within a window W_s. However in our proposed sliding window model we can store and process only one batch of events at a time. The goal is to be able to answer questions about much longer window while requiring only a fraction of the memory needed to store all events in that interval.

The methods discussed in Section 5.2 propose mining patterns in batches with very low frequency thresholds (also referred to as the error parameter) and provide a bound on the total error in the frequencies of the reported patterns. As already noted, mining episodes with very low frequency thresholds quickly results in an explosion in the number of candidates. Therefore we need to operate at fairly high frequency thresholds. Mining top-k most frequent patterns *plus a few more in each batch* achieves our goal and therefore makes the solution feasible. Next we explore the guarantees we can have about the top-k patterns reported over the entire window of interest.

5.3.1 Mining top-k frequent patterns

In the sliding window setting, we are restricted access to just the most recent batch of data, while we are required to report the top-k patterns in the most recent window. In general the top-k patterns in the window may be entirely different from the top-k patterns of all batches constituting the window put together. This is illustrated with help of an example below.

Example 5.3.1 (Worst case top-k)**.** *Let W be a window consisting of four batches $B_1, \ldots, B_4$. Let $ABCD$, $EFGH$, $IJKL$, $MNOP$, $PQRS$ and $WXYZ$ be six patterns seen in the data. The patterns in each batch with corresponding counts are listed in Figure 5.2. The count of a pattern in the window W is given by the sum of counts in the batches. The window counts of all patterns are shown in Table 5.2.*

Pattern	Count
WXYZ	12
PQRS	10
ABCD	8
MNOP	8
EFGH	0
IJKL	0

B_1

Pattern	Count
EFGH	15
IJKL	12
MNOP	10
ABCD	9
PQRS	0
WXYZ	0

B_2

Pattern	Count
WXYZ	12
EFGH	10
ABCD	10
MNOP	8
PQRS	0
IJKL	0

B_3

Pattern	Count
IJKL	11
PQRS	9
MNOP	8
ABCD	8
EFGH	0
WXYZ	0

B_4

Figure 5.2: An example where the batch wise top-k (k=2) sets are disjoint from the top-k of the window.

Table 5.2: Window counts of all patterns.

Pattern	Count
$ABCD$	**35**
$EFGH$	25
$IJKL$	23
$MNOP$	**34**
$PQRS$	19
$WXYZ$	25

In this example the top-2 patterns in batch B_1 are $PQRS$ and $WXYZ$. Similarly $EFGH$ and $IJKL$ in B_2, $EFGH$ and $WXYZ$ in B_3 and, $IJKL$ and $PQRS$ in B_4. The patterns $ABCD$ and $MNOP$ have higher window counts than any other pattern but never appear in the top-2 of any of the batches. In other words, the patterns that finally turn out to be frequent over the window are 'flying under the radar' in each batch. This example can be easily generalized to any number of batches and any k.

Example 5.3.1 shows that in the worst case there exists no relationship between window top-k and batch wise top-k and hence not much can be inferred about the top-k patterns in the window by looking at just the frequent patterns of each batch. In order to establish a connection between batches and the window, we need an assumption on the data generation process. This assumption must be sufficiently general to be applicable to many different data sources. Let the frequency of a pattern α in a batch B_s be denoted by $f^s(\alpha)$. We make the following assumption about the way frequency of patterns change over time.

Assumption 5.3.1 (Bounded rate change)**.** *The change in frequency of any pattern α across consecutive batches B_s and B_{s+1} is bounded, i.e*

$$|f^{s+1}(\alpha) - f^s(\alpha)| \leq \delta \tag{5.3}$$

where δ is a positive number much less than $f^s(\alpha)$ and $f^{s+1}(\alpha)$.

This assumption leads to some interesting properties and allows theoretical guarantees for obtaining top-k most frequent patterns in the window (while looking at atmost one batch in the window at a time).

Table 5.3 gives evidence to support our bounded rate change assumption. Here we consider four multi-neuronal spike train recordings reported in [1]. In the experimental setup, the spiking activity of neurons was recorded in milliseconds accuracy. The average firing rate of cortical neurons vary between 20 Hz to 40 Hz. Each dataset was mined for top-k (k=50) episodes of size 4 in batches of span 150 sec. The second and third columns in Table 5.3 list the average and standard deviation of change in frequencies of top-k episodes in consecutive batches. The last two columns give the average batch-wise frequency and standard deviation of these episodes. We see that the change in frequency across batches is one order of magnitude less than the absolute frequencies of the episodes. This validates that bounded rate change assumption is realistic. Ideally we would like to compute these statistics for all possible episodes. But achieving this goal is clearly infeasible.

Table 5.3: Statistics of top-50 episode frequencies and δ - change in frequency across batches. Batchsize = 150 sec.

Dataset	Avg. δ	Std. δ	Avg. Freq.	Std. Freq.
6-3-22	115.01	95.35	5064.1	123.15
6-3-24	78.53	51.53	8804.68	217.99
6-3-25	64	45.51	4021.1	98.5
6-3-10	19.60	17.57	1054.1	45.0

Now we describe a few interesting properties resulting from the bounded rate change assumption (Assumption 5.3.1). These properties will be later used to provide theoretical guarantees for the proposed algorithms.

Let the frequency of the k^{th} most frequent pattern in B_s be denoted by f_k^s. All patterns in the top-k set exceed f_k^s in frequency.

Lemma 5.3.1. *Consider two consecutive batches B_s and B_{s+1} with f_k^s and f_k^{s+1} as the frequencies of their respective k^{th} most frequent pattern. Under Assumption 5.3.1 we have (5.4).*

$$|f_k^{s+1} - f_k^s| \leq \delta \tag{5.4}$$

Proof. There exist at least k patterns with frequency $\geq f_k^s$ in B_s (by definition). Hence, there exist at least k patterns with frequency $\geq (f_k^s - \delta)$ in B_{s+1} (since frequency of a pattern can decrease by at most δ between B_s and B_{s+1}).

Similarly, there can be atmost k patterns with frequency $\geq (f_k^s + \delta)$, since frequency of a pattern can increase by atmost δ. No other pattern outside the top-k set can exceed the frequency $(f_k^s + \delta)$. □

Next we show that if the frequency of a pattern is known relative to f_k^s in the current batch B_s, we can bound its frequency in a later batch.

Theorem 5.3.1. *If $f^s(\alpha) \geq f_k^s$, then*

$$f^{s+r}(\alpha) \geq f_k^{s+r} - 2|r|\delta$$

Proof. We know from Property 5.3.1 that $f^{s+r}(\alpha)$ is at least $(f^s(\alpha) - |r|\delta)$ and from Lemma 5.3.1, f_k^{s+r} is at most $(f_k^s + |r|\delta)$. Therefore, for any pattern for which $f^s(\alpha) \geq f_k^s$, we have

$$f^{s+r}(\alpha) + |r|\delta \geq f^s(\alpha) \geq f_k^s \geq f_k^{s+r} - |r|\delta$$

Rearranging, we get $f^{s+r}(\alpha) \geq f_k^{s+r} - 2|r|\delta$. □

Theorem 5.3.2. *If $f^s(\alpha) < f_k^s$, then*

$$f^{s+r}(\alpha) < f_k^{s+r} + 2|r|\delta$$

Proof. Analogously, we have

$$f^{s+r}(\alpha) - |r|\delta \leq f^s(\alpha) < f_k^s \leq f_k^{s+r} + |r|\delta$$

Rearranging we get $f^{s+r}(\alpha) < f_k^{s+r} + 2|r|\delta$. □

Corollary 5.3.1. *Under Assumption 5.3.1, the top-k patterns in a batch $B_{s'+r}$ belong to the set of patterns with frequency greater than $(f_k^{s'} - 2|r|\delta)$ in the current batch $B_{s'}$, where r is any integer.*

Proof. Assume that we know $f_k^{s'}$ for the batch $B_{s'}$. If a pattern α is to belong to the set of top-k frequent patterns in the batch $B_{s'+r}$, then $f^{s'+r}(\alpha) \geq f_k^{s'+r}$ (where $f_{s'+r}(\alpha)$ and $f_k^{s'+r}$ are unknown for $r > 0$).

Substituting $s = s' + r$ in Theorem 5.3.1, we get $f^{s'}(\alpha) \geq f_k^{s'} - 2|r|\delta$. □

Corollary 5.3.2. *Top-k patterns in batch $B_{s'}$ have frequency greater than $(f_k^{s'-1} - 2\delta)$ in the previous batch $B_{s'-1}$. Therefore frequent patterns in $B_{s'-1}$ at frequency threshold $(f_k^{s'-1} - 2\delta)$ contain all the patterns that can be in the top-k of the next batch.*

5.3.2 Mining top-k patterns

In this section we present the sketch of an algorithm for mining patterns in a batch-wise manner and generating the top-k patterns over a window. We begin by describing the theory that guarantees finding all top-k patterns in window under the bounded rate change assumption.

Lemma 5.3.2 (Completeness of Top-k)**.** *A necessary condition for a pattern to be in the top-k set of the window W_s is that the frequency of the pattern $\geq (f_k^{s'} - 2(m-1)\delta)$ in every batch $B_{s'} \in W_s$.*

Proof. We prove this by showing that if a pattern fails this threshold in even one batch of W_s, then there exist k or more patterns with window frequency greater than the pattern under consideration.

Consider two patterns α and β such that in a batch $B_{s'}$ $f^{s'}(\alpha) \geq f_k^{s'}$ and $f^{s'}(\beta) < (f_k^{s'} - 2(m-1)\delta)$. In any batch B_p, we have

$$\begin{aligned} f^p(\alpha) &\geq f^{s'}(\alpha) - |p - s'|\delta \\ &\geq f_k^{s'} - |p - s'|\delta \end{aligned} \tag{5.5}$$

and

$$\begin{aligned} f^p(\beta) &\leq f^{s'}(\beta) + |p - s'|\delta \\ &< (f_k^{s'} - 2(m-1)\delta) + |p - s'|\delta \end{aligned} \tag{5.6}$$

If the batches $B_{s'}$ and B_p belong to the same window W_s, then $|p - s'| \leq (m-1)$. Substituting $(m-1)$ for $|p - s'|$ in (5.5) and (5.6) we get:

$$f^p(\alpha) \geq f_k^{s'} - (m-1)\delta > f^p(\beta) \tag{5.7}$$

In each batch $B_p \in W_s$ we have $f^p(\beta) < f^p(\alpha)$. This implies $f^{W_s}(\beta) < f^{W_s}(\alpha)$. There are atleast k patterns in $B_{s'}$ with $f^{s'}(\alpha) \geq f_k^{s'}$ and each one has window frequency greater than that of β. Therefore β cannot be in the window-top-k of W_s. □

Mining all patterns in each batch $B_{s'}$ of W_s with frequency threshold of $(f_k^s - 2(m-1)\delta)$ gives us the complete set of patterns containing all the top-k patterns over the window W_s. Note that this is achieved without having to remember all events that constitute the window.

However in practice, this frequency threshold turns out to be low. The slow rate change assumption is not a strong enough condition to separate out the patterns of interest while looking at one batch of data at a time. In view of this, we propose a new pattern class with a stronger notion of persistence and argue that in many real applications it is an interesting class of patterns to mine for.

An important but often ignored issue with mining frequent patterns is the following. At times there can be bursty patterns [128] in the data. A bursty pattern is a pattern that appears suddenly in the event stream, with very high frequency for a short span of time and quickly disappears. If patterns are to be thought of as trends in the data, bursty patterns that last for short intervals of time do not characterize this property well. However, since top-k mining is defined solely based on frequency, the output can be overwhelmed by such non-persistent bursts of patterns. The notion of persistence looks for patterns that persist over several windows and finds them reliably while providing no guarantees for bursty patterns.

5.3.3 A new class of patterns: (v, k)-persistent

Definition 5.3.2 ((v, k)-persistent pattern)**.** *A pattern is said to be (v, k)-persistent in a window W_s if it appears as a top-k most frequent pattern in atleast v batches of W_s.*

We present a modified data mining problem using the notion of (v, k)-persistent episodes.

Problem 5.3.2 (Top k (v, k)-persistent over window W_s)**.** *Mine top-k most frequent and (v, k)-persistent patterns of size ℓ in the window W_s, where k, ℓ and v are user defined positive integers.*

We show next that the notion of persistence of a pattern can be exploited to operate the mining process at sufficiently high frequency thresholds.

Lemma 5.3.3. *Mining patterns with frequency threshold $(f_k^s - 2(m - v)\delta)$ in each batch B_s gives complete counts of all (v, k)-persistent patterns in the window $W_{s'}$, where s' is such that $B_s \in W_{s'}$.*

Proof. Consider a set of v batches, V_α, where a pattern α is in each of the batch-wise top-k most frequent patterns. Any batch that can be included together in the same window as the batches in V_α, is within $(m - v)$ batches of these v batches. This is illustrated in Figure 5.3 where the dot in each batch (represented by a rectangle) indicates a pattern α and the vertical position of the dot indicates its relative frequency. The pattern α is in the top-k of batches $\in V_\alpha$.

Let $B_q (\notin V_\alpha)$ be a batch that can be placed in the same window as the batches in V_α. Let $B_{\hat{p}(q)} (\in V_\alpha)$ be the batch nearest to B_q where α is in top-k. From Corollary 5.3.1, we have (5.8), since $|q - \hat{p}(q)| \leq (m - v)$.

$$f^q(\alpha) \geq f_k^q - 2(m - v)\delta \tag{5.8}$$

At a given batch B_s, in order to report the top-k (v, k)-persistent patterns in any window that includes B_s, we must track the frequencies of each pattern that can potentially be in

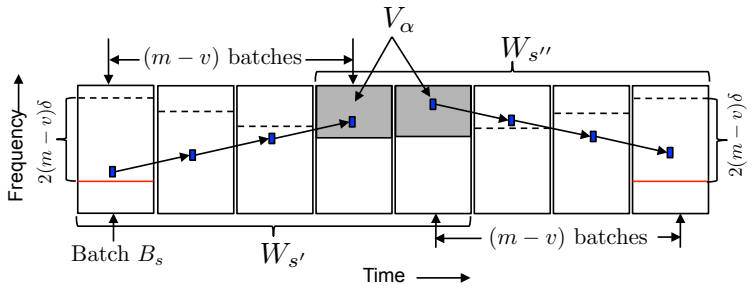


Figure 5.3: Illustration of the frequency threshold required for mining (v, k)-persistent patterns. $W_{s'}$ and $W_{s''}$ represent the two extreme windows where α is still a v-persistent top-k pattern ($v = 2$).

the top-k over atleast v-batches in the future (see $W_{s'}$ in Fig.5.3) or was already in top-k of v-batches but its batch wise frequency in all batches in $W_{s''}$ is required in order to report the top-k over the entire window (see $W_{s''}$ in Fig.5.3). The support threshold for both these cases is given by (5.8).

Therefore, if we mine with a frequency threshold of $f_k^s - 2(m-v)\delta$ in each batch B_s, we are guaranteed to have all the batch-wise frequencies of (v, k)-persistent patterns over any window containing the v batches. □

This is an important result. It enables mining of all (v, k)-persistent patterns at moderately high frequency thresholds (when v is chosen sufficiently close to m). The notion of persistence is of value in many applications including neuroscience and adds an extra dimension to frequent patterns beyond just the frequency measure. Mining with frequency threshold $(f_k^s - 2(m-v)\delta)$ can output patterns that are not (v, k)-persistent but they can be easily removed from the output with help of a little book keeping.

Next, we derive a lower bound for the frequency of a pattern that is in the top-k set of episodes in atleast v batches over the window W_s. This shows that (v, k)-persistence also guarantees a minimum window frequency for qualifying patterns.

Lemma 5.3.4. *The minimum frequency of a pattern α that is in the batch-wise top-k of atleast v batches in the window W_s is given by*

$$f^{W_s}(\alpha) \geq \left(\sum_{B_{s'} \in W_s} f_k^{s'} \right) - (m-v)(m-v+1)\delta \tag{5.9}$$

Proof. The total frequency of the pattern α over the window W_s can be written as (see

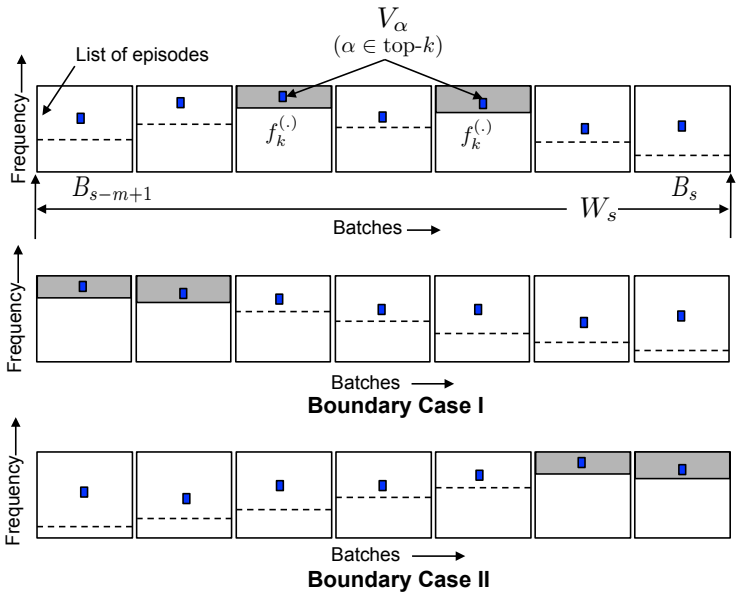


Figure 5.4: Illustration of batches in a window W_s. Each rectangle represents the list of patterns in a batch ordered in decreasing order of batch-wise frequency. The square dot indicates a pattern α that is top-k most frequent in v batches ($v = 2$) (marked in grey).

Figure 5.4)

$$f^{W^s}(\alpha) = \underbrace{\sum_{B_p \in V_\alpha} f^p(\alpha)}_{(I)} + \underbrace{\sum_{B_q \in W_s \setminus V_\alpha} f^q(\alpha)}_{(II)} \tag{5.10}$$

where the pattern α is in top-k of the batches V_α.

In Figure 5.4, each rectangle represents the list of all patterns in a batch ordered in decreasing order of frequency. The square dot indicates a pattern α that is in top-k in the V_α batches (marked in grey).

Let B_p be a batch in V_α. We have $f^p(\alpha) \geq f_k^p$ and the term in part (I) of (5.10) is bounded as follows:

$$\sum_{B_p \in V_\alpha} f^p(\alpha) \geq \sum_{B_p \in V_\alpha} f_k^p \tag{5.11}$$

In part (II) of (5.10), B_q is a batch where α is not in top-k, ($B_q \notin V_\alpha$). Let $B_{\hat{p}(q)}$ be the nearest batch to B_q in which α is in the top-k ($B_{\hat{p}(q)} \in V_\alpha$). $|\hat{p}(q) - q|$ gives the number of batches in between the two. From Theorem 5.3.1, we have $f^q(\alpha) \geq (f_k^q - 2|\hat{p}(q) - q|\delta)$ as $f^{\hat{p}(q)}(\alpha) \geq f_k^{\hat{p}(q)}$.

$$\sum_{B_q} f^q(\alpha) \geq \sum_{B_q} f_k^q - \sum_{B_q} 2|\hat{p}(q) - q|\delta \tag{5.12}$$

The second term in (5.12) is maximized when each batch B_q is as far away from $B_{\hat{p}(q)}$ as possible with pattern frequencies obeying Property 5.3.1. This condition is illustrated in Boundary Case I and II of Figure 5.4. Therefore, we have (5.13).

$$\begin{aligned}\sum_{B_q} f^q(\alpha) &\geq \left(\sum_{B_q} f_k^q\right) - 2[1 + 2 + \ldots + (m - v)]\delta \\ &= \left(\sum_{B_q} f_k^q\right) - (m - v)(m - v + 1)\delta \end{aligned} \tag{5.13}$$

Adding together the bounds of Part (I) and Part (II), we get the bound in (5.9). □

In practice this result can be used to filter the (v, k)-persistent patterns from other patterns found during the mining process.

5.3.4 A heuristic connection with top-k over the window

A (v, k)-persistent pattern under the bounded rate change assumption must have relatively high frequency in the window. That is, if the parameter v is suitably chosen, the lower bound for the window frequency of a pattern given by Lemma 5.3.4 can be fairly high. This is because each f_k^q is itself a high frequency threshold for any reasonable k. Therefore it is intuitive to believe that with high probability the (v, k)-persistent patterns will constitute a significant portion of the set of top-k patterns in a window. A natural question to ask is whether there exists $\bar{v} < m$ such that the top-k patterns in the window are a subset of the $(\bar{v}, k)$-persistent patterns. Similarly is there a $\bar{k}$ for which $(v, \bar{k})$-persistent patterns contain the true top-k in the window for a given v? Both are difficult questions to answer since there is no ordering of the persistent patterns possible. For example, we cannot say that all exactly 1-persistent patterns have window frequency strictly less that of 2-persistent patterns. Hence we cannot claim that if there are more than k (v, k)-persistent patterns, the top-k patterns will all be (v, k) persistent.

Now consider two patterns α and β. The frequency of a pattern in W_s can be expressed in terms of its batch-wise frequencies. Thus, we can express the difference between the window frequencies of α and β as follows:

$$f^{W_s}(\alpha) - f^{W_s}(\beta) = \sum_{B_{s'} \in W_s} \left(f^{s'}(\alpha) - f^{s'}(\beta)\right) \tag{5.14}$$

Let the batch frequency of α be greater than that of β in atmost v batches. In v terms of the summation we have $f^{s'}(\alpha) > f^{s'}(\beta)$ and $f^{s'}(\alpha) \leq f^{s'}(\beta)$ in the remaining $(m-v)$ terms. In order to minimize the difference, we need to maximize each term. This means setting $(m-v)$ terms to zero and assigning the remaining terms maximum possible values while ensuring that across consecutive batches the change in frequency is atmost δ. This condition is satisfied by the case shown in Figure 5.5.

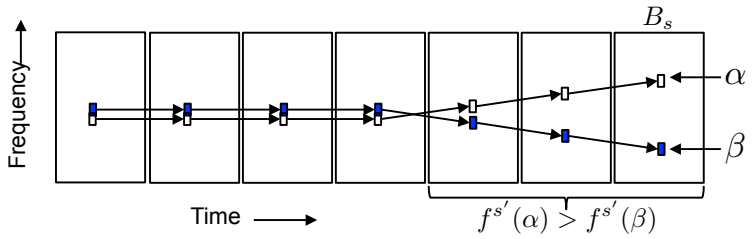


Figure 5.5: Illustration of the case that maximizes $f^{W_s}(\alpha) - f^{W_s}(\beta)$ with the batch frequency of α greater than that of β in atmost v-batches.

Therefore, $f^{W_s}(\alpha) - f^{W_s}(\beta)$ can now be written as:

$$\begin{aligned} f^{W_s}(\alpha) - f^{W_s}(\beta) = & \sum_{s'=(s-v+1)}^{s} \Big(f^{s'}(\alpha) - f^{s'}(\beta) \Big) \\ =& 2\delta[1 + 2 + \ldots + v] \\ =& \delta v(v+1) \end{aligned} \tag{5.15}$$

If α has batch frequency greater than β in atmost v-batches, the maximum difference between their window frequencies is given by

$$f^{W_s}(\alpha) - f^{W_s}(\beta) \leq v(v+1)\delta \tag{5.16}$$

Setting a frequency threshold for the window at $f_k^{W_s} - v(v+1)\delta$ while doing batch-wise mining can ensure that we find patterns in atleast v-batches. Since there is no direct way of setting the window level frequency threshold while doing batch wise mining we lower the batch frequencies by $\frac{v(v+1)\delta}{m}$. This gives a new frequency threshold $f_k^s - m\left[2 - \frac{v}{m} + (\frac{v}{m})^2\right]\delta$. In the result section we show that this frequency threshold improves precision and recall computed with respect to window-wise top-k without any persistence criteria.

5.3.5 Incremental algorithm

In this section we present an efficient algorithm for incrementally mining patterns with frequency $\geq (f_k^s - \theta)$. From our formalism, the value of θ is specified by the type of pattern we want to mine. For (v,k) persistence, $\theta = 2(m-v)\delta$ whereas for only top-k under the bounded rate change assumption (Assumption 5.3.1) it is $2(m-1)\delta$.

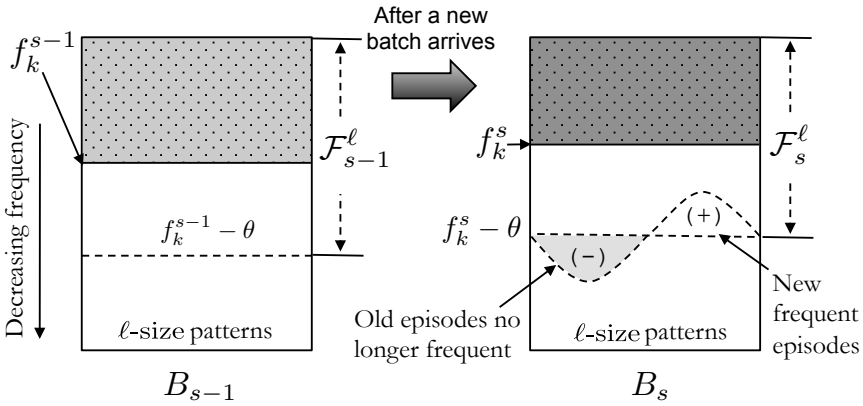


Figure 5.6: The set of frequent patterns can be incrementally updated as new batches arrive.

Recall that the goal of our mining task is to report frequent patterns of size ℓ. After processing the data in the batch B_{s-1}, we desire all patterns with frequency greater than $(f_k^{s-1} - \theta)$. Algorithmically this is achieved by first setting a high frequency threshold and mining for patterns using the classical level wise Apriori method [4]. If the number of patterns of size-ℓ is less than k, the support threshold is decreased and the mining repeated until atleast k ℓ-size patterns are found. At this point f_k^s is known. The mining process is repeated once more with the frequency threshold $(f_k^s - \theta)$. Doing this entire procedure for every new batch can be expensive and wasteful. After seeing the first batch of the data, whenever a new batch arrives we have information about the patterns that were frequent in the previous batch. This can be exploited to incrementally and efficiently update the set of frequent episodes in the new batch. The intuition behind this is that the frequencies of the majority of episodes do not change much from one batch to the next. As a result a small number of episode fall below the new support threshold in the new batch. There is also the possibility of some new episodes becoming frequent. This is illustrated in Figure 5.6. In order to efficiently find these sets of episodes, we need to maintain additional information that allows us to avoid full-blown candidate generation. We show that this state information is a by-product of Apriori algorithm and therefore any extra processing is unnecessary.

In the Apriori algorithm, frequent patterns are discovered iteratively, in ascending order of their size and it is often referred to as a levelwise procedure. The procedure alternates between counting and candidate generation. First a set C^i of candidate i-size patterns is created by joining the frequent $(i-1)$-size itemsets found in the previous iteration. Then the data is scanned for determining the frequency or count of each candidate pattern and the frequent i-size patterns are extracted from the candidates. An interesting observation is that all candidate episodes that are not frequent constitute the negative border of the frequent lattice. This is true because, in the Apriori algorithm, a candidate pattern is generated only when all its subpatterns are frequent. The usual approach is to discard the border. For our purposes, the patterns in the border contain the information required to identify the change in the frequent sets from one batch to the next.

The pseudocode for incrementally mining frequent patterns in batches is listed in Algorithm 6. Let the frequent episodes of size-i be denoted by $\mathcal{F}_s^i$. Similarly, the border episodes of size-i are denoted by $\mathcal{B}_s^i$. The frequency threshold used in each batch is $f_k^s - \theta$. In the first batch of data, the top-k patterns are found by progressively lowering the frequency threshold f_{min} by a small amount ϵ (Lines 1-8). Once atleast k patterns of size ℓ are found, f_k^s is determined and the mining procedure repeated with a threshold of $f_k^s - \theta$. The border patterns generated during level wise mining are retained.

For subsequent batches, first f_k^s is determined. As shown in Corollary 5.3.2, if $\theta \geq 2\delta$, then under the bounded rate change assumption the set of frequent patterns $\mathcal{F}_{s-1}^\ell$ in batch B_{s-1} contains all patterns that can be frequent in the next batch B_s. Therefore simply updating the counts of all patterns in $\mathcal{F}_{s-1}^\ell$ in the batch B_s and picking the k^{th} highest frequency gives f_k^s (Lines 10-11). The new frequency threshold f_{min} is set to be $f_k^s - \theta$. The procedure, starting from bottom (size-1 patterns) updates the lattice for B_s. The data is scanned to determine the frequency of new candidates together with the frequent and border patterns from the lattice (Line 15-18). In the first level (patterns of size 1), the candidate set is empty. After counting, the patterns from the frequent set $\mathcal{F}_{s-1}^\ell$ that continue to be frequent in the new batch are added to $\mathcal{F}_s^\ell$. But if a pattern is no longer frequent it is marked as a border set and all its super episodes are deleted (Lines 19-24). This ensures that only border patterns are retained in the lattice. All patterns, either from the border set or the new candidate set, that are found to be frequent are added to $\mathcal{F}_s^\ell$. Such episodes are also added to F_{new}^i. Any remaining infrequent patterns belong to border set because otherwise they would have atleast one of infrequent subpatterns and would have been deleted at a previous level (Line 24). These patterns are added to $\mathcal{B}_s^\ell$ (Line 30). The candidate generation step is required to fill out the missing parts of the frequent lattice. We want to avoid a full blown candidate generation. Note that if a pattern is frequent in B_{s-1} and B_s then all its subpatterns are also frequent in both B_s and B_{s-1}. Any new pattern ($\notin \mathcal{F}_{s-1}^\ell \cup \mathcal{B}_{s-1}^\ell$) that turns frequent in B_s, therefore, must have atleast one subpattern that was not frequent in B_{s-1} but is frequent in B_s. All such patterns are listed in F_{new}^i. The candidate generation step (Line 31) for the next level generates only candidate patterns with atleast one subpattern $\in F_{new}^i$. This greatly restricts the number of candidates generated at each level without compromising the completeness of the results.

The space and time complexity of the candidate generation is now $O(|F_{new}^i|.|\mathcal{F}_s^i|)$ instead of $O(|\mathcal{F}_s^i|^2)$ and in most practical cases $|F_{new}^i| \ll |\mathcal{F}_s^i|$. This is crucial in a streaming application where processing rate must match the data arrival rate.

For a window W_s ending in the batch B_s, the set of output patterns can be obtained by picking the top-k most frequent patterns from the set $\mathcal{F}_s^\ell$. Each pattern also maintains a list that stores its batch-wise counts is last m batches. The window frequency is obtained by adding these entries together. The output patterns are listed in decreasing order of their window counts.

Example 5.3.2. *In this example we illustrate the procedure for incrementally updating the*

Algorithm 6 Mine top-k v-persistent patterns.

Input: A new batch of events B_s, the lattice of frequent and border patterns $(\mathcal{F}^*_{s-1}, \mathcal{B}^*_{s-1})$, and parameters k and θ

Output: The lattice of frequent and border patterns $(\mathcal{F}^*_s, \mathcal{B}^*_s)$

if $s = 1$ **then**
  f_{min} = high value
  while $|\mathcal{F}^\ell_s| < k$ **do**
    Mine patterns with frequency $\geq f_{min}$
    $f_{min} = f_{min} - \epsilon$
  f^s_k = frequency of the k^{th} most frequent pattern $\in \mathcal{F}^\ell_s$
  Mine ℓ-size patterns in B_s with frequency threshold $f_{min} = f^s_k - \theta$
  Store the frequent and border patterns (of size $= 1 \ldots \ell$) in $(\mathcal{F}^*_s, \mathcal{B}^*_s)$
else
  CountPatterns$(\mathcal{F}^\ell_{s-1}, B_s)$
  Set f^s_k = frequency k^{th} highest frequency (pattern $\in \mathcal{F}^\ell_{s-1}$)
  Set frequency threshold for B_s, $f_{min} = (f^s_k - \theta)$
  $\mathcal{C}^1 = \phi$ {New candidate patterns of size = 1}
  for $i = 1 \ldots \ell - 1$ **do**
    $\mathcal{F}^i_s = \phi$ {Frequent patterns of size i}
    $\mathcal{B}^i_s = \phi$ {Border patterns of size i}
    $F^i_{new} = \phi$ {List of newly frequent Patterns}
    CountPatterns$(\mathcal{F}^i_{s-1} \cup \mathcal{B}^i_{s-1} \cup \mathcal{C}^i, B_s)$
    for $\alpha \in \mathcal{F}^i_{s-1}$ **do**
      if $f^s(\alpha) \geq f_{min}$ **then**
        $\mathcal{F}^i_s = \mathcal{F}^i_s \cup \{\alpha\}$
      else
        $\mathcal{B}^i_s = \mathcal{B}^i_s \cup \{\alpha\}$
        Delete all its super-patterns from $(\mathcal{F}^*_{s-1}, \mathcal{B}^*_{s-1})$
    for $\alpha \in \mathcal{B}^i_{s-1} \cup \mathcal{C}^i$ **do**
      if $f^s(\alpha) \geq f_{min}$ **then**
        $\mathcal{F}^i_s = \mathcal{F}^i_s \cup \{\alpha\}$
        $F^i_{new} = F^i_{new} \cup \{\alpha\}$
      else
        $\mathcal{B}^i_s = \mathcal{B}^i_s \cup \{\alpha\}$
    C^{i+1} = **GenerateCandidate**$_{i+1}(F^i_{new}, \mathcal{F}^i_s)$
return $(\mathcal{F}^*_s, \mathcal{B}^*_s)$

frequent patterns lattice as a new batch B_s is processed (see Figure 5.7).

Figure 5.7(A) shows the lattice of frequent and border patterns found in the batch B_{s-1}. $ABCD$ is a 4-size frequent pattern in the lattice. In the new batch B_s, the pattern $ABCD$ is no longer frequent. The pattern $CDXY$ appears as a new frequent pattern. The pattern lattice in B_s is shown in Figure 5.7(B).

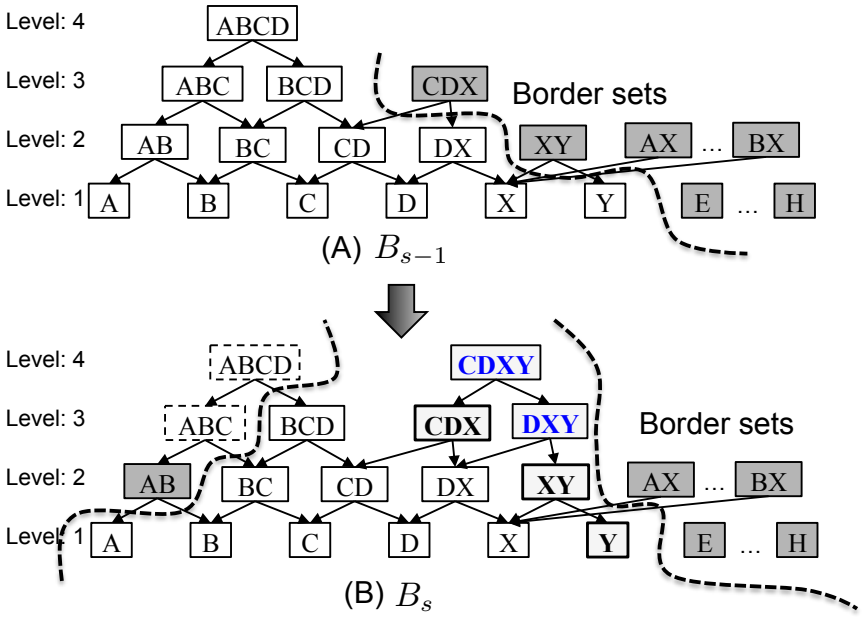


Figure 5.7: Incremental lattice update for the next batch B_s given the lattice of frequent and border patterns in B_{s-1}.

In the new batch B_s, AB falls out of the frequent set. AB now becomes the new border and all its super-patterns namely ABC, BCD and ABCD are deleted from the lattice.

At level 2, the border pattern XY turns frequent in B_s. This allows us to generate DXY as a new 3-size candidate. At level 3, DXY is also found to be frequent and is combined with CDX which is also frequent in B_s to generate CDXY as a 4-size candidate. Finally at level 4, CDXY is found to be frequent. This shows that border sets can be used to fill out the parts of the pattern lattice that become frequent in the new data.

5.3.6 Estimating δ dynamically

The parameter δ in the bounded rate change assumption is a critical parameter in the entire formulation. But unfortunately the choice of the correct value for δ is highly data-dependent. In the streaming setting, the characteristics of the data can change over time. Hence one predetermined value of δ cannot be provided in any intuitive way. Therefore we estimate δ from the frequencies of ℓ-size episodes in consecutive windows. We compute the differences in frequencies of episodes that are common in consecutive batches. Specifically, we consider the value at the 75th percentile as an estimate of δ. We avoid using the maximum change as it tends to be noisy. A few patterns exhibiting large changes in frequency can skew the estimate and adversely affect the mining procedure.

5.3.7 Counting serial episodes in windows and batches

When dealing with an infinite event sequence in batches or windows it is important to handle episode occurrences that cross over the window boundary in a meaningful way while defining the frequency measure. For frequency based on non-overlapped occurrences there can be atmost one occurrence of any episode that could straddle across the window boundary. We redefine the notion of frequency of an episode in a window or a batch of events as follows.

Definition 5.3.3 (Frequency of an episode)**.** *The frequency of an episode in a window W_s (or batch B_s) is defined as the maximum number of non-overlapped occurrences ending in the window W_s (or batch B_s).*

Therefore, an occurrence that begins before the time interval or window starts and completes within it will contribute towards the frequency of the episode. But any occurrence that begins in the window but completes outside it is only considered in the next window that contains the last event of that occurrence. This is illustrated in the following example.

Example 5.3.3 (Occurrences of episode $A \rightarrow B \rightarrow C \rightarrow D$)**.** *Figure 5.8 shows three non-overlapped occurrences of the serial episode $A \rightarrow B \rightarrow C \rightarrow D$ in an event sequence. According to Definition 5.3.3, only the first two occurrences are considered for the frequency of α in the window W. The last occurrence will be counted in the next window.*

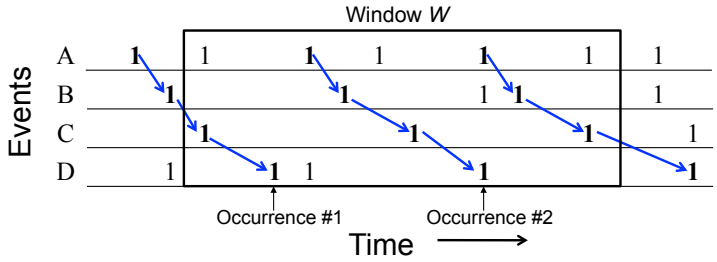


Figure 5.8: Illustration of occurrences of a serial episode $\alpha = A \rightarrow B \rightarrow C \rightarrow D$ in an example event sequence. The frequency of α in the time slice W is 2 (marked by the labels Occurrence #1 and Occurrence #2 above).

5.4 Results

The aim of the experimental section is to bring out the key aspects of the streaming pattern mining problem that we have formulated so far. In the related works sections we noted that there are no algorithms designed specifically to handle episodes. We pointed out the reasons why some of the other streaming pattern mining algorithms for itemsets and sequential patterns cannot be easily applied in the context of frequent episodes. Hence we do not

compare the proposed algorithms with existing algorithms or their adaptations. We also observe that the top-k frequent patterns in a batch can be much different from those of the top-k in the window containing the batch. In order to bring out the advantages of our proposed methods we fashion trivial algorithms that mine top-k episodes in batches. We compare the performance of all the algorithms with ground truth obtained by mining top-k using an entire window of data at each instant.

In this section we present results both on synthetic data and data from real neuroscience experiments. We compare the performance of the proposed streaming episode mining algorithm on synthetic data to quantify the effect of different parameter choices and data characteristics on the quality of the top-k episodes reported by each method. Finally we show the quality of results obtained on neuroscience data.

For the purpose of comparing the quality of results we setup the following six variants of the mining frequent episodes:

Alg 0: This is the naive brute force top-k mining algorithm that loads an entire window of events at a time and mines the top-k episode by repeatedly lowering the frequency threshold for mining. When a new batch arrives, events from the oldest batch are retired and mining process is repeated from scratch. This method acts as the baseline for comparing all other algorithms in terms of precision and recall.

Alg 1: The top-k mining is done batch-wise. The top-k episodes over a window are reported from within the set of episodes that belong to the batch-wise top-k of atleast one batch in the window.

Alg 2: Here the algorithm is same as Alg 1, but once an episodes enters the top-k in any of the batches in a window, it is tracked over several subsequent batches. An episode is removed from the list of episodes being tracked if it does not occur in the top-k of last m consecutive batches. This strategy helps in obtaining a larger candidate set and also in getting more accurate counts of candidate patterns over the window.

Alg 3: This algorithm uses a batch-wise frequency threshold $f_k^s - 2\delta$ which ensures that the top-k episodes in the next batch B_{s+1} are contained in the frequent lattice of B_s. This avoids multiple passes of the data while trying to obtain k most frequent episodes lowering the support threshold iteratively. The patterns with frequency between f_k^s and $f_k^s - 2\delta$ also improve the overall precision and recall with respect to the window.

Alg 4: In this case the batch-wise frequency threshold is $f_k^s - 2(m-v)\delta$ which guarantees finding all (v,k)-persistent episodes in the data. We report results for $v = 3m/4$ and $v = m/2$.

Alg 5: Finally, this last algorithm uses the heuristic proposed in Section 5.3.4. Again we report results for $v = 3m/4$ and $v = m/2$.

Table 5.4: Details of synthetic datasets used in evaluating streaming algorithms.

Dataset Name	Alphabet Size	Rest Firing Rate	Number of Patterns
A1	500	10.0	50
A2	1000	10.0	50
A3	5000	10.0	50
B1	1000	2.0	50
B2	1000	10.0	50
B3	1000	25.0	50
C1	1000	10.0	10
C1	1000	10.0	25
C1	1000	10.0	50

5.4.1 Synthetic datasets

Data generation model: The data generation model for synthetic data is based on the same inhomogeneous Poisson process model as used in Chapter 3 for evaluating the algorithm for learning excitatory dynamic networks. We introduce two changes to this model. First, in order to mimic real data more closely in the events that constitute the background noise, the event-type distribution is modified to follow a power law distribution. This gives the long tail characteristics to the simulated data.

The second modification was to allow the rate of arrival of episodes to change over time. As time progresses, the frequency of episodes in the recent window or batch slowly changes. We use a randomized scheme to update the connection strengths in the neuronal simulation model. The updates happen at the same timescale as the batch sizes used for evaluation.

The datasets we used for experimental evaluation are listed in Table 5.4. The name of the data set is listed in Column 1, the size of the alphabet (or total number of event types) in Column 2, the average rest firing rate in Column 3 and the number of patterns embedded in Column 4. In these datasets the data length is set at 50×10^5 sec which is roughly 50 million events, the alphabet size is varied from 1000 to 5000, the resting firing rate from 10.0 to 25.0, and the number of patterns embedded in the data from 25 to 50.

5.4.2 Comparison of algorithms

In Fig. 5.9, we compare the five algorithms—Alg 1 through Alg 5—that report the frequent episodes over the window looking at one batch at a time with the baseline algorithm Alg 0 that stores and processes the entire window at each window slide. The results are averaged over all 9 data sets shown in Table 3.2. We expect this to marginalize the data characteristics and give a more general picture of each algorithm's performance. The parameter settings

for the experiments are shown in Table 5.5. Fig. 5.9 (a) plots the precision of the output of each algorithm compared to that of Alg 0 (treated as ground truth). Similarly, Fig 5.9 (b) shows the recall. Since the size of output of each algorithm is roughly k, the corresponding precision and recall numbers are almost the same. Average runtimes are shown in Fig. 5.9 (c) and average memory requirement in MB is shown in Fig. 5.9 (d).

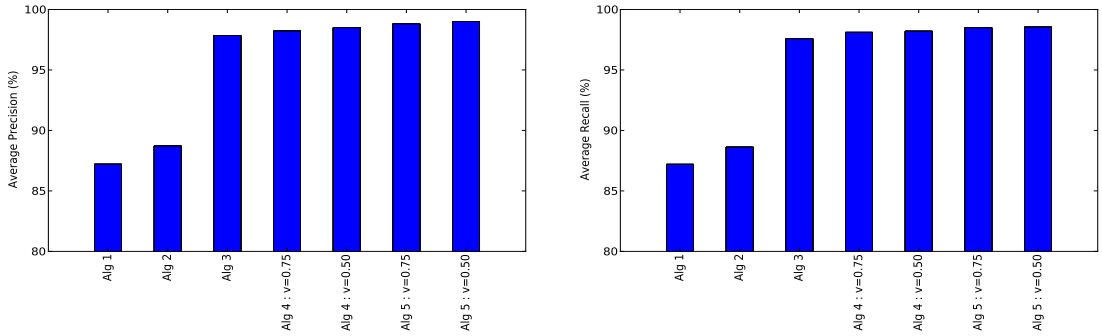
Table 5.5: Parameter settings of streaming algorithms.

Parameter	Value(s)
Batch size T_b	10^5 sec ($\approx$ 1 million events per batch)
Number of batches in a window m	10 (5,15)
v, in (v, k)-persistence	0.5m, 0.75m
k in (v, k)-persistence and in top-k	25, 50
ℓ - size of episode	4

We consistently observe that Alg 1 and 2 give lower precision and recall values compared with any other algorithm. This reinforces our observation that the top-k patterns in a window can be much different from the top-k patterns in the constituent batches. Alg 2 provides only a slight improvement over Alg 1 by tracking an episode once it enters the top-k over subsequent batches. This improvement can be attributed to the fact that window frequencies of patterns that were once in the top-k is better estimated. Alg 3 gives higher precision and recall compared to Alg 1 and 2. The frequency threshold used in Alg 3 is given by Corollary 5.3.2. Using this result we are able to estimate the value $f_k^s - 2\delta$ by simply counting the episodes that are frequent in the previous batch. This avoids multiple iterations required in general for finding the top-k patterns. Fortunately this threshold also results in significantly higher support and precision.

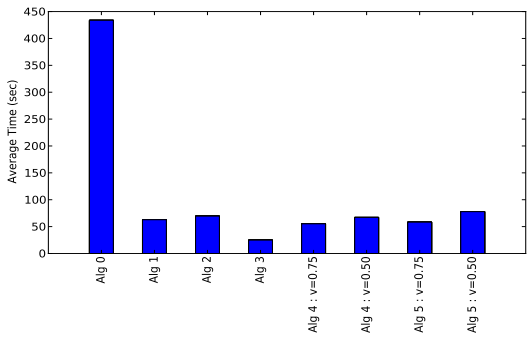
We ran Alg 4 and 5 for two different values of v, viz. $v = m/2$ and $v = 3m/4$. Both these algorithms guarantee finding all (v, k)-persistent patterns for respective values of v. For the sake of comparison we also include episodes that exceeded the frequency threshold prescribed by (v, k)-persistence, but do not appear in top-k of atleast v batches. We observe that the precision and recall improves a little over Alg 3 with reasonable increase in memory and runtime requirements. For a higher value of v, this algorithm insists that the frequent patterns must persist over more batches. This raises the support threshold and as a result there is improvement in terms of memory and runtime, but a small loss in precision and recall. Note that the patterns missed by the algorithm are either not (v, k)-persistent or our estimation of δ has errors in it. In addition, Alg 5 gives a slight improvement over Alg 4. This shows that our heuristic presented in Section 5.3.3 is effective.

Overall the proposed methods (Alg 3, 4 and 5) give atleast one order of magnitude improvement over the baseline algorithm (Alg 0) in terms of both time and space complexity.

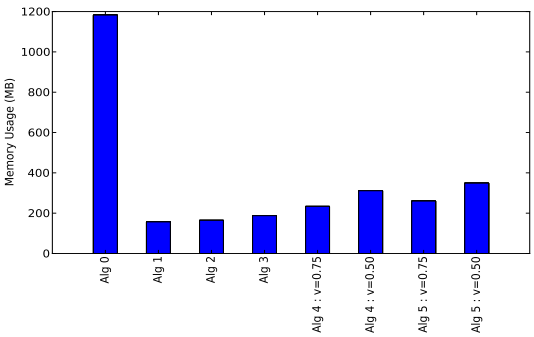


(a) Precision

(b) Recall



(c) Runtime



(d) Memory

Figure 5.9: Comparison of average performance of different streaming episode mining algorithm. Alg 1 and 2 give lower precision and recall values compared with any other algorithms. Overall the proposed methods give atleast one order of magnitude improvement over the baseline algorithm (Alg 0) in terms of both time and space complexity.

Performance over time

Next we consider one dataset A2 with number of event types = 1000, the average resting firing rate as 10 Hz, and the number of embedded patterns = 50. On this data we show how the performances of the five algorithms change over time. The window size is set to be $m = 10$ and the batch size = 10^5 sec. Fig. 5.10 shows the comparison over 50 contiguous batches. Fig. 5.10 (a) and (b) show the way precision and recall evolve over time. Fig. 5.10 (c) and (d) show the corresponding memory usage and runtimes.

The data generation model allows the episode frequencies to change slowly over time. In the dataset used in the comparison we change frequencies of embedded episodes at two time intervals: batch 15 to 20 and batch 35 to 42. In Fig. 5.10 (a) and (b), we have a special plot shown by the dashed line. This is listed as Alg 0 in the legend. What this line shows is the comparison of top-k episodes between consecutive window slides. In other words the top-k episodes in window W_{s-1} are considered as the predicted output in order to obtain the precision and recall for W_s. The purpose of this curve is to show how the true top-k set changes with time and show how well the proposed algorithms track this change.

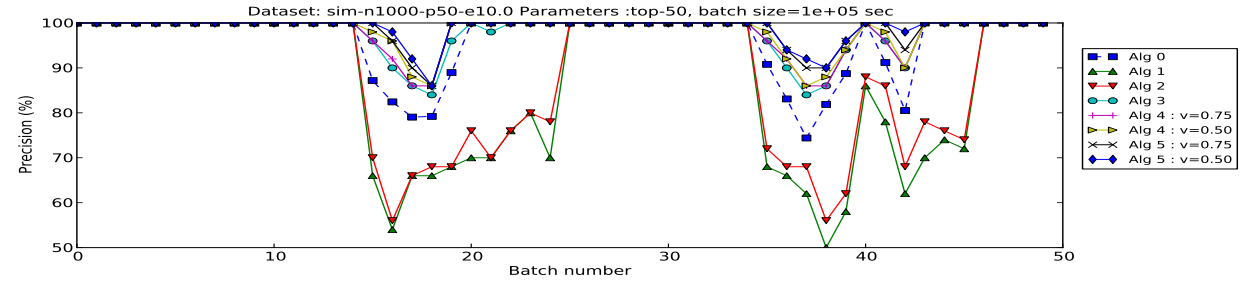
Alg 1 and 2 perform poorly. On an average in the transient regions (batch 15 to 20 and batch 35 to 42) they perform 15 to 20% worse than any other method. Alg 3, 4 and 5 (for v=0.75m and v=0.5m) perform consistently above the reference curve of Alg 0. It expected of any reasonable algorithm to do better than the algorithm which uses the top-k of W_{s-1} to predict the top-k of the window W_s. The precision and recall performance are in the order Alg 3 < Alg 4 v=0.75m < Alg 4 v=0.5m < Alg 5 v=0.75m < Alg 4 v=0.5m. This is in the same order as the frequency thresholds used by each method, and as expected.

In terms of runtime and memory usage, the changing top-k does not affect these numbers. The lowest runtimes are those of Alg 3. The initial slope in the runtimes and memory usage seen in Algo 0, is due to the fact that the algorithm loads the entire window, one batch at a time into memory. In this experiment the window consists of $m = 10$ batches. Therefore only after the first 10 batches one complete window span is available in memory.

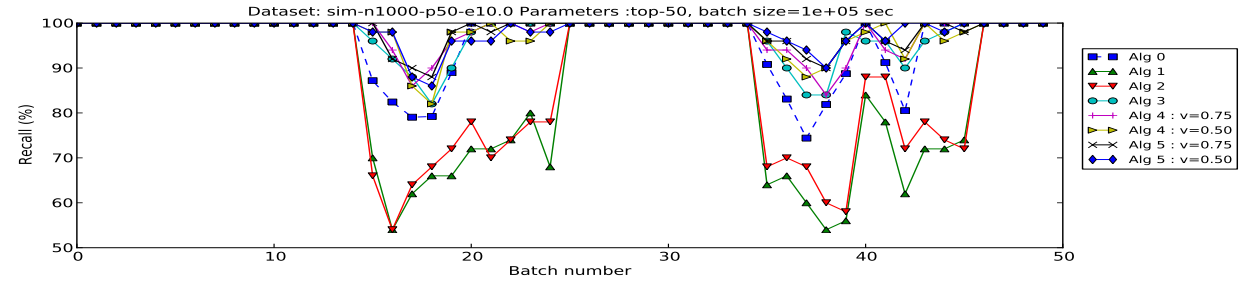
Effect of data characteristics

In this section we present results on synthetic data with different characteristics, namely, number of event types (or alphabet size), noise levels and number of patterns embedded in the data.

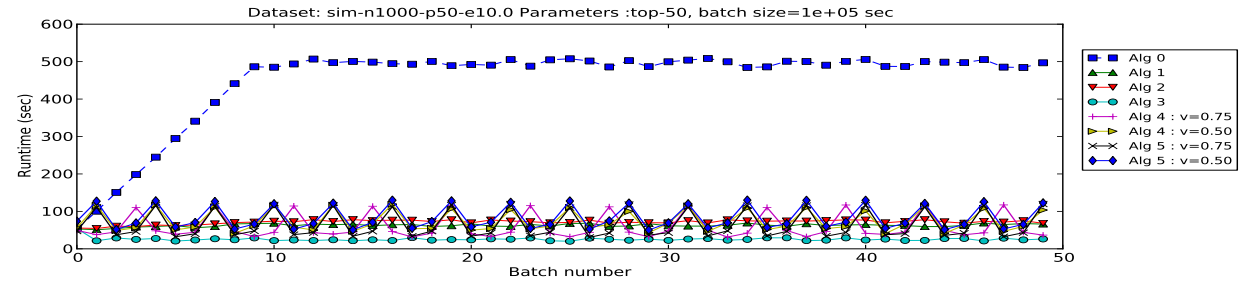
In Fig. 5.11 we report the effect of alphabet size on the quality of result of the different algorithms. In datasets A1, A2 and A3 the alphabet size, i.e. the number of distinct event types, is varied from 500 to 5000. We observe that for smaller alphabet sizes the performance is better. Alg 1 and 2 perform consistently worse than the other algorithms for different alphabet sizes.



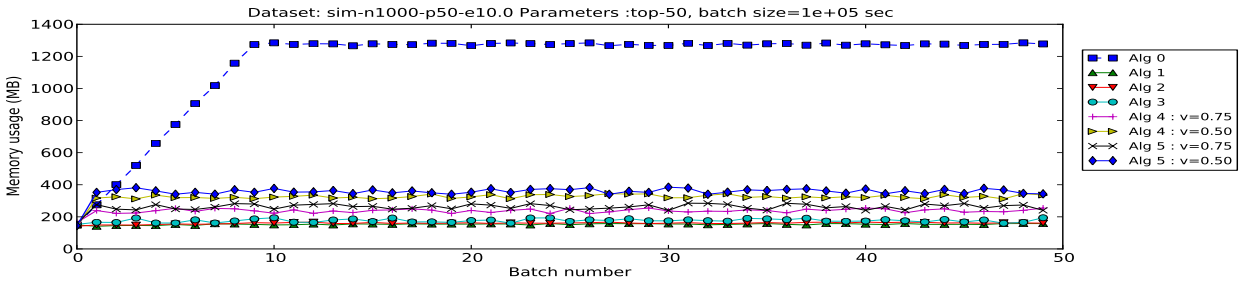
(a) Precision



(b) Recall

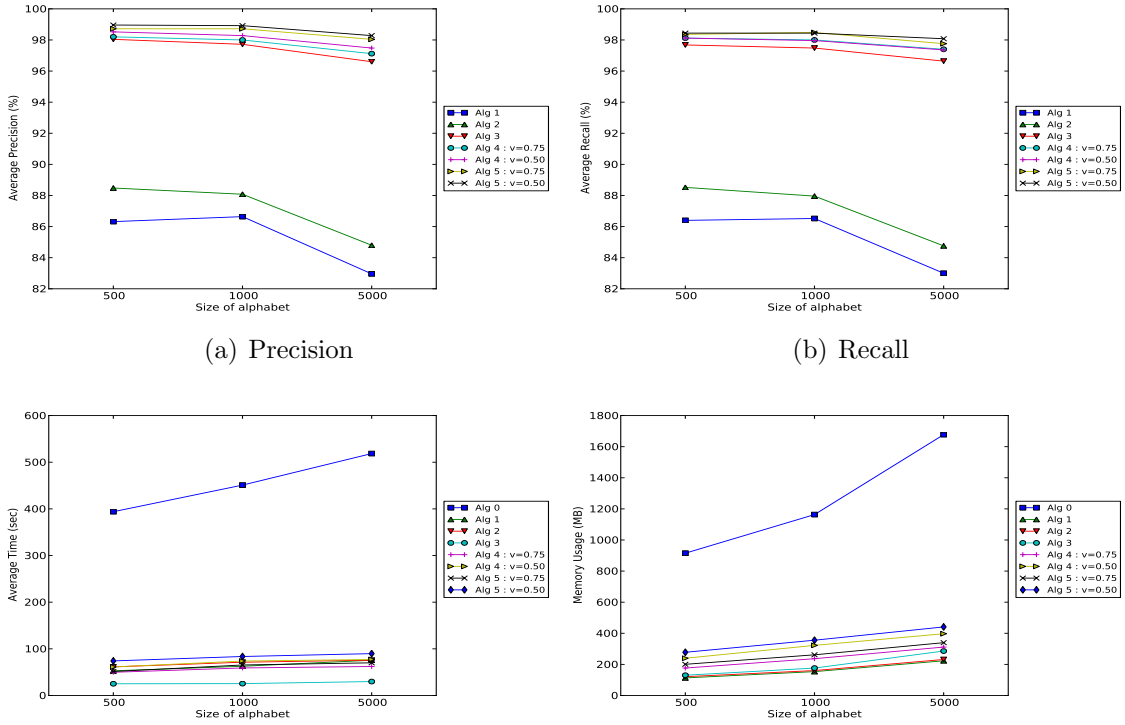


(c) Runtime



(d) Memory

Figure 5.10: Comparison of the performance of different streaming episode mining algorithms over a sequence of 50 batches (where each batch is 10^5 sec wide and each window consists of 10 batches).



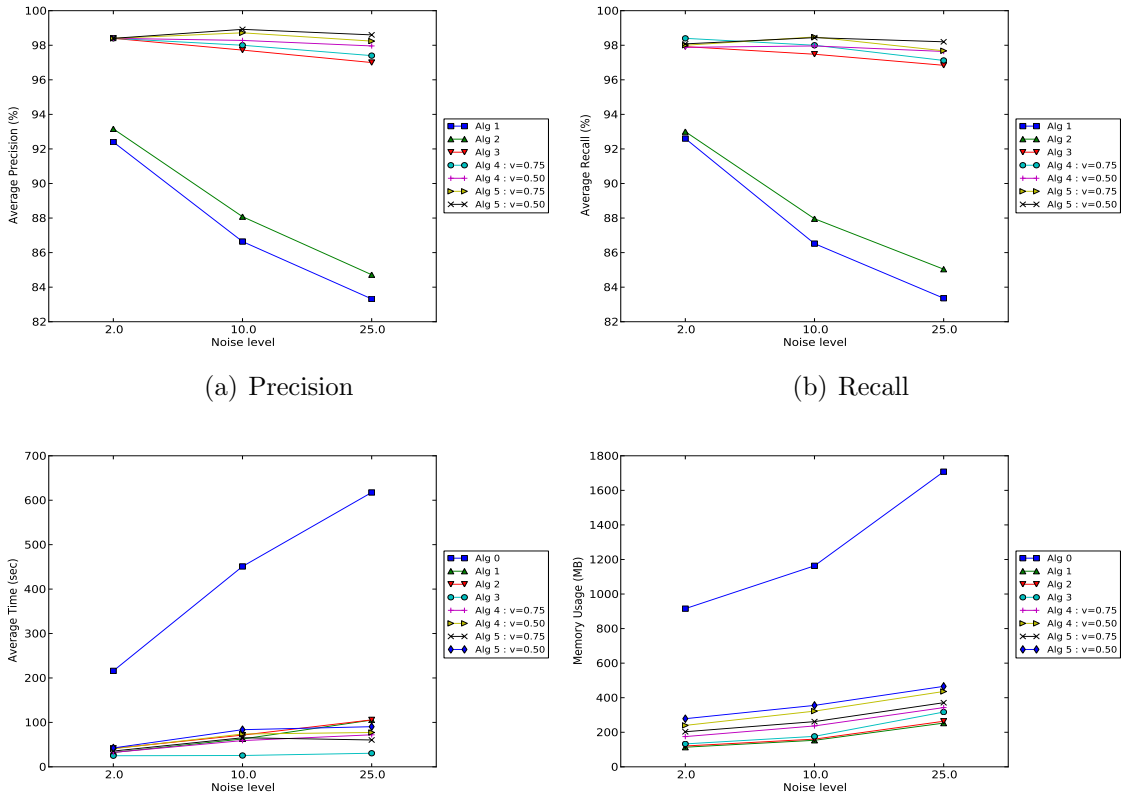
(a) Precision (b) Recall

(c) Runtime (d) Memory

Figure 5.11: Effect of alphabet size on streaming algorithms. The proposed algorithms are robust to large alphabet sizes. Precision and recall drop by only 2-4% going from alphabet size of 500 to 5000.

In this experiment we find that the quality of results for the proposed algorithms is not very sensitive to alphabet size. The precision and recall numbers drop by only 2-4%. This is quite different from the pattern mining setting where the user provides a frequency threshold. In our experience alphabet size is critical in the fixed frequency threshold based formulation. For low thresholds, large alphabet sizes can quickly lead to uncontrolled growth in the number of candidates. In our formulation the support threshold is dynamically readjusted and as a result the effect of large alphabet size is attenuated.

Next, in Fig. 5.12, we show the effect of noise. The average rate of firing of the noise event-types (event types that do not participate in pattern occurrences) is varied from 2.0 Hz to 25 Hz. The precision and recall of Alg 1 and 2 degrade quickly with increase in noise. A small decrease in precision and recall of Alg 3 and Alg 4 is seen. But the performance of Alg 5 (for both v=0.75m and v=0.5m) stays almost at the same level. It seems that the frequency threshold generated by Alg 5 is sufficiently low to finds the correct patterns even at higher noise level but not so low as to require significantly more memory ($\approx$ 400 MB) or runtime ($\approx$ 70 sec per batch at noise = 25.0 Hz for v=0.5m) as compared to other algorithms.



(a) Precision (b) Recall

(c) Runtime (d) Memory

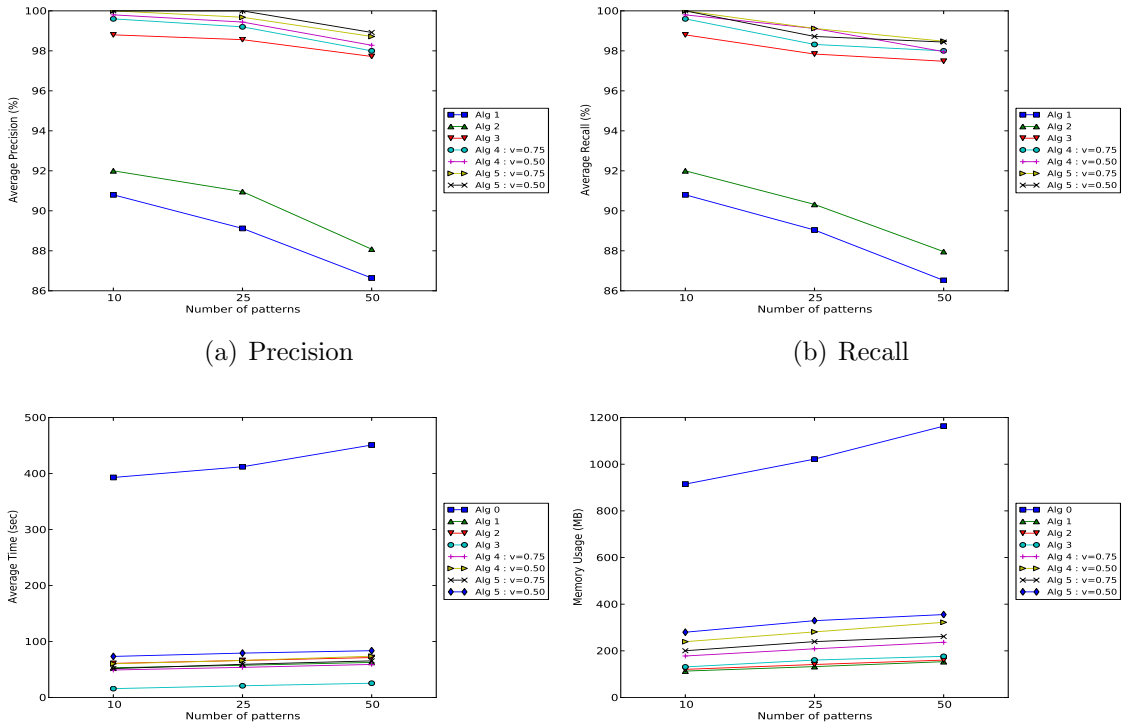
Figure 5.12: Effect of noise on streaming algorithms. In terms of precision and recall Alg 5 is most robust to noise in the data.

In Fig. 5.13, we change the number of patterns embedded in the data and study the effect. The number of embedded patterns vary from 10 to 50. Once again the performance of our proposed methods is fairly flat in all the metrics. Alg 3, 4 and 5 are seen to be less sensitive to the number of patterns embedded in the data than Alg 1 and 2.

Effect of parameters

So far the discussion has been about the effect of data characteristics of the synthetic data. The parameters of the mining algorithms were kept fixed. In this section we look at two important parameters of the algorithms, namely, the batch size T_b and the number of batches that make up a window, m.

In Fig. 5.14, the quality and performance metrics are plotted for three different batch sizes: 10^3, 10^4 and 10^5 (in sec). Batch size appears to have a significant effect on precision and recall. There is a 10% decrease in both precision and recall when batch size is reduced to



(a) Precision (b) Recall

(c) Runtime (d) Memory

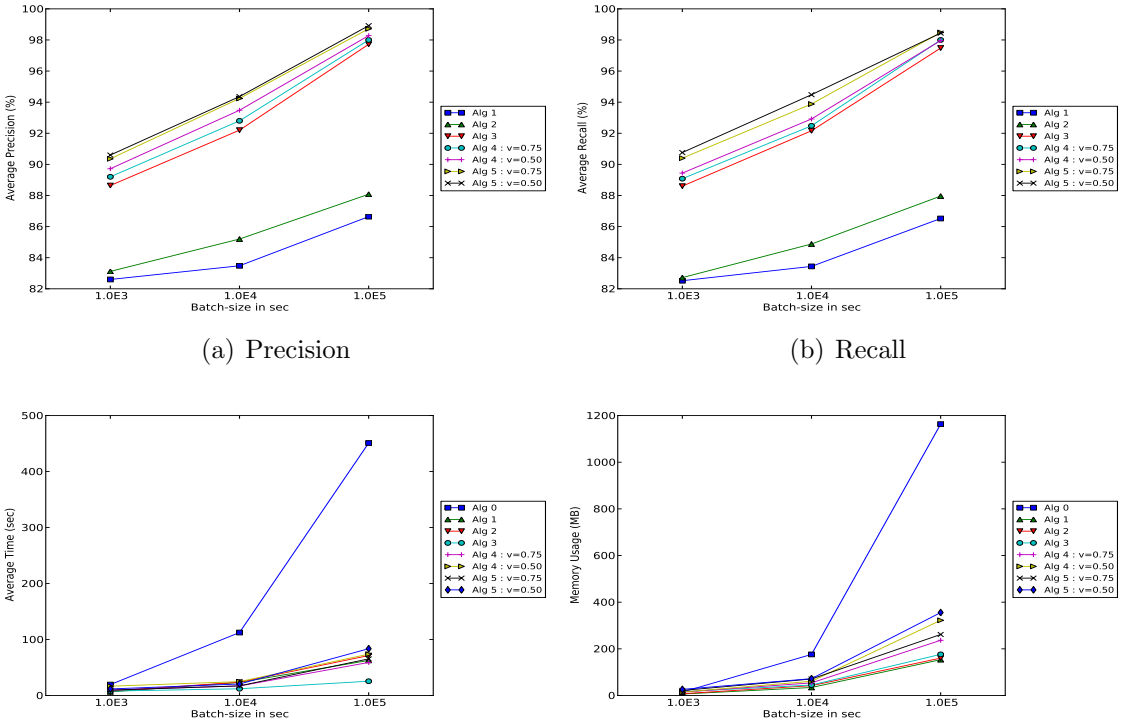
Figure 5.13: Effect of number of embedded patterns on streaming algorithms. Alg 3, 4 and 5 are seen to be less sensitive to the number of patterns embedded in the data than Alg 1 and 2.

10^3 sec from 10^5. But note that a 100 fold decrease in batch size only changes the quality of the result by 10%.

It is not hard to imagine that for smaller batch sizes the episode statistics can have higher variability in different batches resulting in a lower precision and recall over the window. As the size of the batch grows the top-k in the batch starts to resemble the top-k of the window. Transient patterns will not be able to gather sufficient support in a large batch size.

As expected, the runtimes and memory usage are directly proportional to the batch size in all cases. The extra space and time is required only to handle more data. Batch size does not play a significant role in growth of number of candidates in the mining process.

Next in Fig. 5.15, we show how the number of windows in a batch affect the performance. Precision and recall are observed to decrease linearly with the number of batches in a window in Fig. 5.15(a) and (b), whereas the memory and runtime requirements grow linearly with the number of batches. The choice of number of batches provides the trade-off between the window size over which the user desires the frequent persistent patterns and the accuracy of the results. For larger window sizes the quality of the results will be poorer. Note that



(a) Precision (b) Recall

(c) Runtime (d) Memory

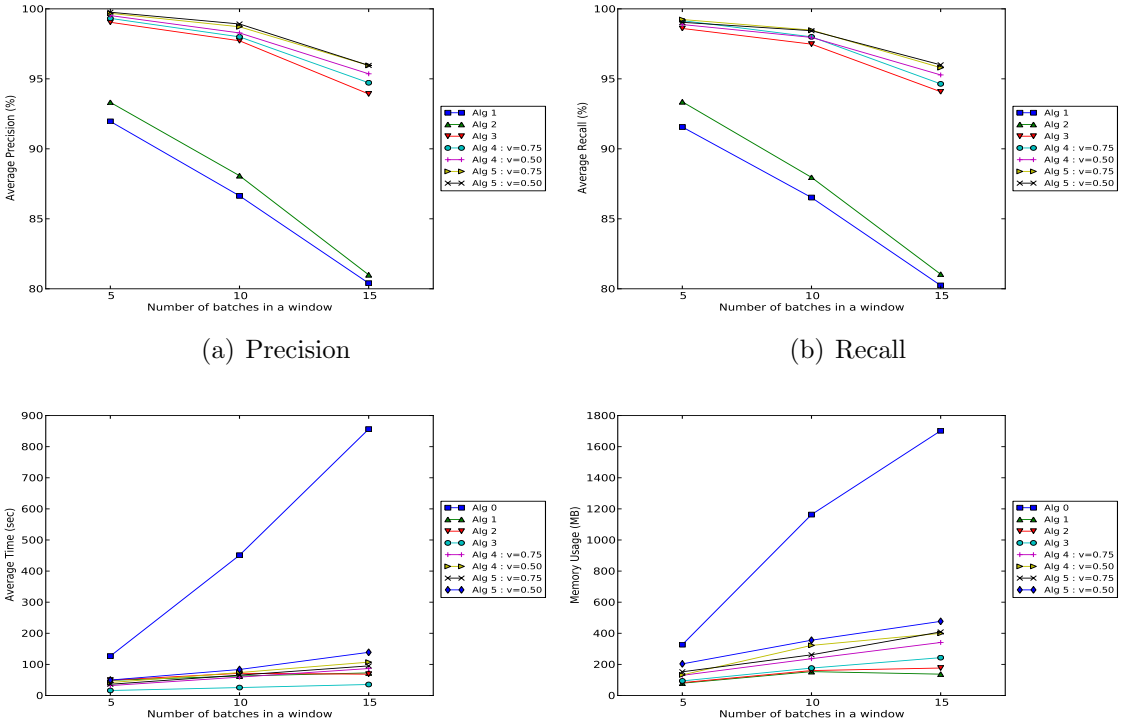
Figure 5.14: Effect of batchsize on streaming algorithms. Larger batch sizes have higher precision and recall. Precision and recall increase logarithmically with batch size. (Note that x-axis is in log scale)

the memory usage and runtime does not increase much for algorithms other than Alg 0 (see Fig. 5.15 (c) and (d)). Because these algorithms only process one batch of data irrespective of the number of batches in window. Although for larger windows the batch-wise support threshold decreases. In the synthetic datasets we see that this does not lead to unprecedented increase in the number of candidates.

5.4.3 Multi-neuronal data

Multi-electrode arrays provide high throughput recordings of the spiking activity in neuronal tissue and are hence rich sources of event data where events correspond to specific neurons being activated. We used the data from dissociated cortical cultures gathered by Steve Potter's laboratory at Georgia Tech [1] over several days. This is a rich collection of recordings from a 64-electrode MEA setup.

We show the result of mining frequent episodes in the data collected over several days from Culture 6 [1]. We use a batch size of 150 sec and all other parameters for mining are the



(a) Precision (b) Recall

(c) Runtime (d) Memory

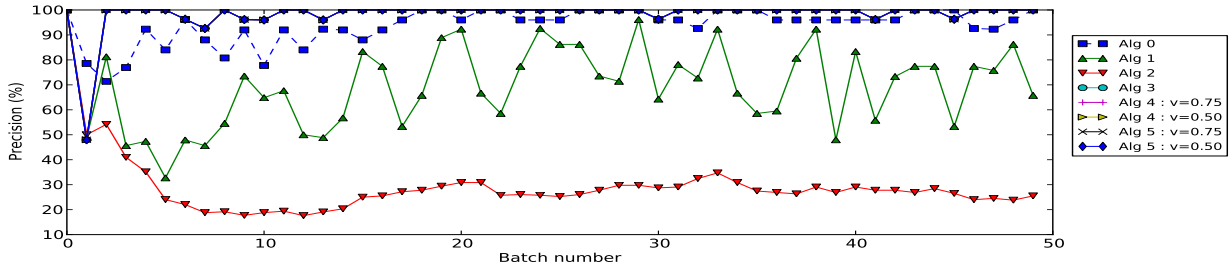
Figure 5.15: Effect of number of batches in a window on streaming algorithms. The memory usage and runtime does not increase much for algorithms other than Alg 0

same as that used for the synthetic data. The plots in Fig. 5.16 show the performance of the different algorithms as time progresses. Alg 1 and 2 give very low precision values which implies that the top-k in a batch is much different from the top-k in the window. This reaffirms the need for the kind of modeling we propose in this chapter. Alg 3, 4 and 5 perform equally well over the MEA data with Alg 3 giving the best runtime performance.

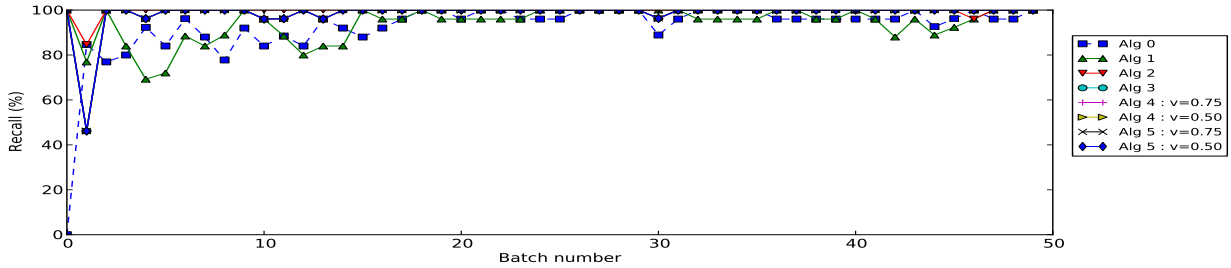
At times Alg 3 requires slightly higher memory than the other algorithm (Alg 4 and 5). This may seem counter intuitive as Alg 4 and 5 use lower frequency threshold. But since δ is dynamically estimated from all episodes being tracked by the algorithm it can easily be the case that the δ estimates made by Alg 3 are looser and hence result in higher memory usage.

5.5 Conclusion

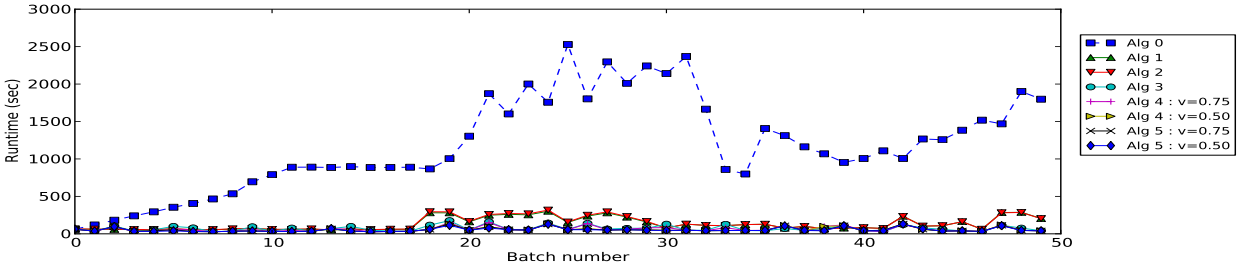
In this chapter, we have studied the important problem of mining frequent episodes in infinite data streams. In particular our contribution in this work is three fold. We unearth an interesting aspect of temporal data mining where the data owner may desire results over



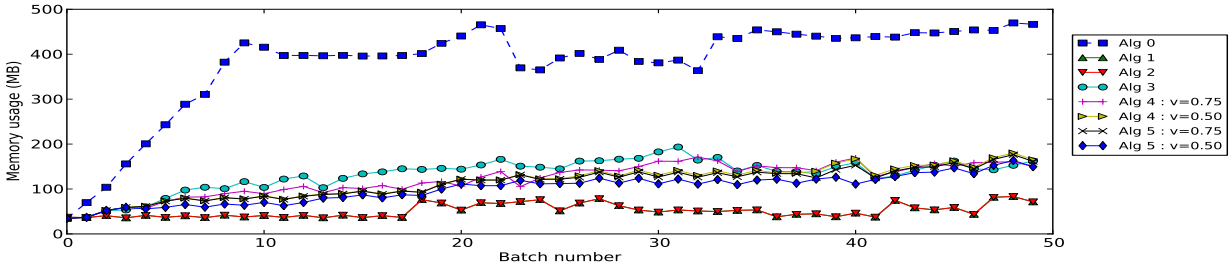
(a) Precision



(b) Recall



(c) Runtime



(d) Memory

Figure 5.16: Comparison of performance of different algorithms on real multi-neuronal data. Alg 3, 4 and 5 perform equally well over the MEA data with Alg 3 giving the best runtime performance.

a span of time in the data that cannot fit in the memory or be processed at a rate faster than the data generation rate. We have proposed a new sliding window model which slides forward in hops of batches. At any point only one batch of data is available for processing. We have studied this problem and identified the theoretical guarantees one can give and the necessary assumptions for supporting them.

In many real applications we find the need for characterizing pattern not just based on their frequency but also their tendency to persist over time. In particular, in neuroscience, the network structure underlying an ensemble of neurons changes much slowly in comparison to the culture wide periods bursting phenomenon. Thus separating the persistent patterns from the bursty ones can give us more insight into the underlying connectivity map of the network. We have proposed the notion of (v, k) persistent patterns to address this problem and outlined methods to mine all (v, k)-persistent patterns in the data. Finally we have provided detailed experimental results on both synthetic and real data to show the advantages of the proposed methods.

Finally, we reiterate that although we have focused on episodes, the ideas presented in this chapter could be applied to other pattern classes with similar considerations.

Chapter 6

Conclusion

Our central theme in this dissertation has been to short-circuit the search for formal temporal models by creating bridges to frequent episode mining. Through this work we have added to the small but growing literature in KDD on creating bridges between diverse threads. We can imagine that as more awareness of these bridges happens, software based on frequent episode mining can become more prominent and will be used as the first "line of attack" for important data mining problems .

In this dissertation we have dealt with four challenging temporal data mining problems and connected them together with the common theme of frequent episode discovery. The pattern class of frequent episodes brings with it a suite of very efficient pattern discovery algorithms:

1. We have demonstrated the use of frequent episodes for motif mining in multivariate time series data. Motif mining in time series has generally not been seen as a discrete pattern mining problem (with some notable exceptions [74, 45, 44]). Our application of episode mining has helped address many issues inherent in the domain like sensitivity to noise and false positives.

2. Our newly developed connections between a graphical model like dynamic Bayesian networks and frequent episodes has two benefits. There is a significant improvement in efficiency of structure learning algorithms and, in return, frequent episode mining can be connected to a stronger theoretical basis. We have established this connection by constraining the conditional probability structure to excitation. There is a potential to explore more ways of defining special classes of models that can leverage frequent patterns in multiple ways.

3. One of the difficulties in most pattern mining algorithms is the numerousness of the output. This is an even bigger problem in temporal patterns like episodes where permutations also come into play. In order to consume these numerous frequent patterns they must be summarized into more meaningful structures. We have shown how episodes

can be used as a basis to learn partial orders. Directly learning partial orders from data is known to be a difficult problem [98]. Using an intermediate representation (such as episodes) to solve a hard problem can be an important strategy in many domains.

4. The final topic in this dissertation has a cross-cutting appeal. Having established the role of frequent episode in several problem domains, it becomes important to study the problem of mining such patterns in infinite data streams. Constraining the class of episodes motivated by an application yields efficient and tractable algorithms. This is an important take-away lesson we learnt in this dissertation, i.e., specializing a seemingly hard problem to incorporate the special characteristics of the application domain can be a secret sauce in the recipe to solving them.

This dissertation has opened up many vistas for future exploration. The area of probabilistic modeling of different pattern classes has great research potential. We can explore more formal classes of models, and more general temporal pattern mining algorithms. We can even try to unify these approaches with the generalized automata-based frequent mining formulation of Laxman *et al* [17]. A typically hard problem is to derive a generative model for frequent partial order patterns. In this class of patterns, data mining approaches frequently struggle with the question of specificity of a pattern. A sequence in the data can satisfy a hierarchy of partial orders. A significance test is needed that trades off the right amount of generalization with the interestingness of a pattern.

The streaming work has led us to apply our algorithms on ever increasing datasets. We are in the process of exploring scalable and distributed paradigms like map-reduce [129] for broader applicability. This will open up a new set of research issues including, but not restricted to, data distribution, preserving the temporal aspect of the data, and breaking the problem into parallelizable pieces. We have taken initial steps using specialized parallel processing hardware in the form of graphical processing units (GPUs) to port frequent episode mining algorithms [130, 131]. Designing parallel algorithms that can leverage the architecture of specialized hardware brings up several challenges especially if the data is time-stamped or ordered. In episode mining the challenge of maintaining data continuity across distributed processors and collecting the result back in correct order are the highlights of this problem.

Finally even non-sequential problems like multiple sequence alignment (MSA) can be approached using frequent episodes. Frequent episodes can be a useful summary structure for reasoning about broader regularities in data. One interesting problem addressed in this context is that of finding couplings across multiple residue positions in proteins of different organisms [132, 133]. Couplings can be thought of as structural constraints that are required in order for a protein molecule to allow it to function. These constraints are respected in organisms across the living world. Frequent episodes can capture regularities in the sequences of amino acid residues which are the building blocks of proteins. These patterns can then be used to learn probabilistic models like Markov random fields [134] to understand these constraints imposed by nature.

Bibliography

[1] D. A. Wagenaar, J. Pine, and S. M. Potter, "An extremely rich repertoire of bursting patterns during the development of cortical cultures," *BMC Neuroscience*, 2006.

[2] N. Takahashi *et al.*, "Watching neuronal circuit dynamics through functional multi-neuron calcium imaging (fmci)," *Neuroscience Research*, vol. 58, no. 3, pp. 219 – 225, 2007.

[3] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds., ch. From data mining to knowledge discovery: an overview, pp. 1–34. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996.

[4] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, pp. 487–499, Sep 1994.

[5] S.-J. Lin, R. Atlas, and K.-S. Yeh, "Global weather prediction and high-end computing at nasa," *Computing in Science and Engg.*, vol. 6, pp. 29–34, January 2004.

[6] S. Wuchty, Z. Oltvai, and A. Barabasi, "Evolutionary conservation of motif constituents in the yeast protein interaction network.," *Nature Genetics*, vol. 35, no. 2, pp. 176–9, 2003.

[7] M. Kato, N. Hata, N. Banerjee, B. Futcher, and M. Zhang, "Identifying combinatorial regulation of transcription factors and binding motifs.," *Genome Biol*, vol. 5, no. 8, p. R56, 2004.

[8] H. Mannila, H. Toivonen, and A. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 259–289, 1997.

[9] R. Jin and G. Agrawal, "An algorithm for in-core frequent itemset mining on streaming data," in *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 210–217, IEEE Computer Society, 2005.

[10] M. J. Zaki and C.-J. Hsiao, "Charm: An efficient algorithm for closed itemset mining," in *SIAM International Conference on Data Mining*, pp. 457–473, 2002.

[11] C.-J. Hsiao and M. Zaki, "Efficient Algorithms for Mining Closed Itemsets and their Lattice Structure," *IEEE Transactions on Knowledge and Data Engineering*, vol. Vol. 17, pp. pages 462–478, Apr 2005.

[12] R. Wong and A. Fu, "Mining top-k frequent itemsets from data streams," *Data Mining and Knowledge Discovery*, vol. 13, pp. 193–217, 2006. 10.1007/s10618-006-0042-x.

[13] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, pp. 1–12, May 2000.

[14] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Eleventh International Conference on Data Engineering* (P. S. Yu and A. S. P. Chen, eds.), (Taipei, Taiwan), pp. 3–14, IEEE Computer Society Press, 1995.

[15] J. Pei, J. Han, B. Mortazavi-Asl, and H. Pinto, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proceedings of the 17th International Conference on Data Engineering*, (Washington, DC, USA), pp. 215–, IEEE Computer Society, 2001.

[16] L. Mendes, B. Ding, and J. Han, "Stream sequential pattern mining with precise error bounds," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, pp. 941 –946, 2008.

[17] S. Laxman, P. Sastry, and K. Unnikrishnan, "Discovering frequent episodes and learning hidden markov models: A formal connection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 11, 2005.

[18] T. Mielikainen, "Discovery of serial episodes from streams of events," in *SSDBM '04: Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, (Washington, DC, USA), p. 447, IEEE Computer Society, 2004.

[19] D. Patnaik, P. Sastry, and K. Unnikrishnan, "Inferring neuronal network connectivity from spike data: A temporal data mining approach," *Scientific Programming*, vol. 16(1), pp. 49–77, 2008.

[20] A. Tanay, R. Sharan, and R. Shamir, "Biclustering Algorithms: A Survey," in *Handbook of Computational Molecular Biology* (S. Aluru, ed.), Chapman & Hall/CRC Computer and Information Science Series, 2005.

[21] S. Madeira and A. Oliveira, "Biclustering Algorithms for Biological Data Analysis: A Survey," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. Vol. 1, pp. 24–45, Jan 2004.

[22] S. Venkataraghavan†, "Biclustering algorithms for detecting combinatorial control of gene expression," Master's thesis, Virginia Polytechnic and State University, 2005. Available at http://scholar.lib.vt.edu/theses/available/.

[23] M. Zaki and N. Ramakrishnan, "Reasoning about Sets using Redescription Mining," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*, pp. 364–373, Aug 2005.

[24] L. Parida and N. Ramakrishnan, "Redescription Mining: Structure Theory and Algorithms," in *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05)*, pp. 837–844, July 2005.

[25] D. Kumar, N. Ramakrishnan, M. Potts, and R. F. Helm, "Turning cartwheels: An alternating algorithm for mining redescriptions," Tech. Rep. cs.CE/0311048, Computing Research Repository, Nov 2003.

[26] D. J. Hand, P. Smyth, and H. Mannila, *Principles of data mining*. Cambridge, MA, USA: MIT Press, 2001.

[27] M. I. Jordan, ed., *Learning in Graphical Models*. MIT Press, 1998.

[28] D. Koller and N. Friedman, *Probabilistic Graphical Models Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[29] F. V. Jensen, *Introduction to Bayesian Networks*. Springer, 1997.

[30] R. Kindermann and J. L. Snell, *Markov Random Fields and Their Applications*. AMS, 1980.

[31] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Wiley, 2008.

[32] S. Laxman and P.S.Sastry, "A survey of temporal data mining," *SADHANA, Academy Proceedings in Engineering Sciences*, 2005.

[33] S. Laxman, P. Sastry, and K. Unnikrishnan, "A fast algorithm for finding frequent episodes in event streams," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pp. 410–419, 2007.

[34] S. Laxman, *Discovering frequent episodes: Fast algorithms, Connections with HMMs and generalizations*. PhD thesis, IISc, Bangalore, India, September 2006.

[35] N. Friedman, K. Murphy, and S. Russell, "Learning the structure of dynamic probabilistic networks," in *Proc. UAI'98*, pp. 139–147, Morgan Kaufmann, 1998.

[36] L. R. Rabiner, ch. A tutorial on hidden Markov models and selected applications in speech recognition, pp. 267–296. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.

[37] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[38] S. Eldawlatly, Y. Zhou, R. Jin, and K. G. Oweiss, "On the use of dynamic bayesian networks in reconstructing functional neuronal networks from spike train ensembles," *Neural Computation*, vol. 22, pp. 158–189, Jan 2010.

[39] F. Bromberg, D. Margaritis, and V. Honavar, "Efficient markov network structure discovery using independence tests," *Journal of Artificial Intelligence Research*, vol. 35, no. 1, pp. 449–484, 2009.

[40] D. Pavlov, H. Mannila, and P. Smyth, "Beyond independence: Probabilistic models for query approximation on binary transaction data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1409–1421, 2003.

[41] C. Wang and S. Parthasarathy, "Summarizing itemset patterns using probabilistic models," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (New York, NY, USA), pp. 730–735, ACM, 2006.

[42] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, "Detecting Time Series Motifs under Uniform Scaling," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (New York, NY, USA), pp. 844–853, ACM, 2007.

[43] P. Patel, E. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," in *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, (Washington, DC, USA), p. 370, IEEE Computer Society, 2002.

[44] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (New York, NY, USA), pp. 493–498, ACM, 2003.

[45] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," in *Proceedings of the Second Workshop on Temporal Data Mining*, pp. 53–68, 2002.

[46] J. Koomey, "Computing's environmental cost." The Economist, Print Edition, May 22 2008.

[47] J. Koomey, "Power conversion in servers and data centers: A review of recent data and developments," in *Applied Power Electronics Conference*, Feb 25 2008.

[48] J. M. Kaplan, W. Forrest, and N. Kindler, "Revolutionizing data center efficiency," tech. rep., McKinsey Report, 2008.

[49] J. Leake and R. Woods, "Revealed: The Environmental Impact of Google Searches." Times Online, January 11 2009.

[50] T. Vanderbilt, "Data center overload." New York Times, June 8 2009.

[51] B. Watson, A. Shah, M. Marwah, C. Bash, R. Sharma, C. Hoover, T. Christian, and C. Patel, "Integrated design and management of a sustainable data center," in *InterPACK '09: Proceedings of ASME InterPACK*, (New York, NY, USA), ASME, July 2009.

[52] D. Patnaik *et al.*, "Sustainable operation and management of data center chillers using temporal data mining," in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 1305–1314, ACM, 2009.

[53] R. Sharma *et al.*, "On building next generation data centers: Energy flow in the information technology stack," in *Compute 2008*, (Bangalore, India), Jan 2008.

[54] C. Belady, "In the data center, power and cooling costs more than the IT equipment it supports," *Electronics Cooling*, vol. 13, Feb 2007.

[55] T. Boucher, D. Auslander, C. Bash, C. Federspiel, T. Patel, C.D. Boucher, D. Auslander, C. Bash, C. Federspiel, T. Patel, C.D. Boucher, D. Auslander, C. Bash, C. Federspiel, and C. Patel, "Viability of dynamic cooling control in a data center environment," *Electronic Packaging*, June 2006.

[56] X. Xuan and K. Murphy, "Modeling Changing Dependency Structure in Multivariate Time Series," in *ICML '07: Proceedings of the 24th International Conference on Machine Learning*, (New York, NY, USA), pp. 1055–1062, ACM, 2007.

[57] A. J. Shah, V. P. Carey, C. E. Bash, and C. D. Patel, "Exergy analysis of data center thermal management systems," *Journal of Heat Transfer*, vol. 130, no. 2, pp. 021401.1–021401.10, 2008.

[58] N. Friedman, I. Nachman, and D. Pe'er, "Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm," in *5th Conf. on Uncertainty in Artificial Intelligence UAI (1999)*, pp. 206–215., 1999.

[59] D. Patnaik *et al.*, "Discovering excitatory networks from discrete event streams with applications to neuronal spike train analysis," in *ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 407–416, IEEE Computer Society, 2009.

[60] L. Bautista and R. Sharma, "Analysis of environmental data in datacenters," tech. rep., HP Labs, 2007.

[61] R. Sharma, R. Shih, C. Patel, and J. Sontag, "Application of Exploratory Data Analysis (EDA) Techniques to Temperature Data in a Conventional Data Center," in *Proceedings of ASME IPACK'07*, 2007.

[62] M. Marwah, R. Sharma, W. Lugo, and L. Bautista, "Anomalous thermal behavior detection in data centers using hierarchical pca," in *SensorKDD '09: Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*, ACM, 2010.

[63] E. Hoke, J. Sun, and C. Faloutsos, "Intemon: Intelligent system monitoring on large clusters," in *VLDB '06: Proceedings of the 32nd International Conference on Very Large Databases*, pp. 1239–1242, VLDB Endowment, 2006.

[64] E. Hoke, J. Sun, J. D. Strunk, G. R. Ganger, and C. Faloutsos, "Intemon: Continuous mining of sensor data in large-scale self-infrastructures," *SIGOPS Operating Systems Review*, vol. 40, no. 3, pp. 38–44, 2006.

[65] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming Pattern Discovery in Multiple Time-series," in *VLDB '05: Proceedings of the 31st International Conference on Very large data bases*, pp. 697–708, VLDB Endowment, 2005.

[66] P. McLachlan, T. Munzner, E. Koutsofios, and S. North, "Liverac: Interactive visual exploration of system management time-series data," in *CHI '08: Proceeding of the Twenty-sixth Annual SIGCHI Conference on Human Factors in Computing Systems*, (New York, NY, USA), pp. 1483–1492, ACM, 2008.

[67] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 219–230, ACM, 2004.

[68] P. Bodik, M. Goldszmidt, and A. Fox, "Hilighter: Automatically building robust signatures of performance behavior for small- and large-scale systems.," in *SysML*, 2008.

[69] X. Gu, S. Papadimitriou, P. S. Yu, and S.-P. Chang, "Toward predictive failure management for distributed stream processing systems," in *ICDCS '08: Proceedings of The 28th International Conference on Distributed Computing Systems*, (Washington, DC, USA), pp. 825–832, IEEE Computer Society, 2008.

[70] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure Prediction in IBM BlueGene/L Event Logs," in *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pp. 583–588, 2007.

[71] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *22nd ACM Symposium on Operating Systems Principles*, 2009.

[72] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *VLDB Journal*, vol. 1, no. 2, pp. 1542–1552, 2008.

[73] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in Data Mining and Knowledge Discovery*, pp. 2–11, ACM, 2003.

[74] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: A Novel Symbolic Representation of Time Series," *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 107–144, 2007.

[75] S. Laxman, V. Tankasali, and R. White, "Stream prediction using a generative model based on frequent episodes in event sequences," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*, (Las Vegas, USA), August 2008.

[76] K. Wang, J. Zhang, F. Shen, and L. Shi, "Adaptive learning of dynamic bayesian networks with changing structures by detecting geometric structures of time series," *Knowledge and Information Systems*, vol. 17, no. 1, pp. 121–133, 2008.

[77] T. D. Nielsen and F. V. Jensen, "Alert systems for production plants: A methodology based on conflict analysis," *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pp. 76–87, 2005.

[78] Y. Shi, M. Klustein, I. Simon, T. Mitchell, and Z. Bar-Joseph, "Continuous hidden process model for time series expression experiments," *Bioinformatics*, vol. 23, pp. 459–467, Jul 2007.

[79] P. Papapetrou *et al.*, "Mining frequent arrangements of temporal intervals," *Knowledge and Information Systems*, vol. 21, no. 2, pp. 133–171, 2009.

[80] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, pp. 462–467, May 1968.

[81] J. Williamson, "Approximating discrete probability distributions with bayesian networks," in *Proc. Intl. Conf. on AI in Science & Technology, Tasmania*, pp. 16–20, 2000.

[82] M. Meila, "An accelerated chow and liu algorithm: Fitting tree distributions to high-dimensional sparse data," in *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 249–257, 1999.

[83] T. Wang and J. Yang, "A heuristic method for learning bayesian networks using discrete particle swarm optimization," *Knowledge and Information Systems*, 2009.

[84] K. Murphy, *Dynamic Bayesian Networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, CA, USA, 2002.

[85] F. Rieke, D. Warland, R. Steveninck, and W. Bialek, *Spikes: Exploring the Neural Code*. The MIT Press, 1999.

[86] J. K. Seppanen, *Using and extending itemsets in data mining: Query approximation, dense itemsets and tiles*. PhD thesis, Helsinki University of Technology, 2006.

[87] H. Sprekeler, C. Michaelis, and L. Wiskott, "Slowness: An objective for spike-timing-dependent plasticity?," *PLoS Computational Biology*, vol. 3, no. 6, 2007.

[88] G. Czanner, U. T. Eden, S. Wirth, M. Yanike, W. A. Suzuki, and E. N. Brown, "Analysis of between-trial and within-trial neural spiking dynamics," *Journal of Neurophysiology*, no. 99, pp. 2672–2693, 2008.

[89] P. S. Sastry and K. P. Unnikrishnan, "Conditional probability based significance tests for sequential patterns in multi-neuronal spike trains," *Neural Computation*, vol. 22, no. 2, pp. 1025–1059, 2010.

[90] V. Raajay, "Frequent episode mining and multi-neuronal spike train data analysis," Master's thesis, IISc, Bangalore, 2009.

[91] E. M. Izhikevich, "Polychronization: Computation with spikes," *Neural Comput.*, vol. 18, no. 2, pp. 245–282, 2006.

[92] D. M. Chickering, "Optimal structure identification with greedy search," *J. Mach. Learn. Res.*, vol. 3, pp. 507–554, 2003.

[93] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, 1992.

[94] C. De Coster *et al.*, "Identifying priorities in methodological research using ICD-9-CM and ICD-10 administrative data: report from an international consortium," *BMC Health Serv Res.*, vol. 6, p. 77, Jun 2006.

[95] E. T. Jaynes, "Information theory and statistical mechanics," *Phys. Rev.*, vol. 106, pp. 620–630, May 1957.

[96] K. Booth and G. Lueker, "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity using PQ-tree Algorithms," *JCSS*, vol. 13, no. 3, pp. 335–379, 1976.

[97] N. Tatti, "Maximum entropy based significance of itemsets," in *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pp. 312–321, 2007.

[98] H. Mannila and C. Meek, "Global Partial Orders from Sequential Data," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pp. 161–168, 2000.

[99] G. Landau, L. Parida, and O. Weimann, "Using PQ Trees for Comparative Genomics," in *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching*, pp. 128–143, 2005.

[100] S. Heber and J. Stoye, "Finding all common intervals of k permutations," in *Proc. of the 12th Combinatorial Pattern Matching CPM'01*, (London, UK, UK), pp. 207–218, Springer-Verlag, 2001.

[101] W. L. Healy and R. Iorio, "Total hip arthroplasty: optimal treatment for displaced femoral neck fractures in elderly patients," *Clin Orthop Relat Res*, pp. 43–48, Dec 2004.

[102] E. Sendtner, T. Renkawitz, P. Kramny, M. Wenzl, and J. Grifka, "Fractured neck of femur–internal fixation versus arthroplasty," *Dtsch Arztebl Int*, vol. 107, pp. 401–407, Jun 2010.

[103] J. A. Lowe, B. D. Crist, M. Bhandari, and T. A. Ferguson, "Optimal treatment of femoral neck fractures according to patient's physiologic age: an evidence-based review," *Orthop Clin North Am*, vol. 41, pp. 157–166, Apr 2010.

[104] W. Kormos, *Primary Care Medicine*, ch. Approach to the Patient With Sinusitis, pp. 1402–1407. Lippincot WIlliams and WIlkins, 2009.

[105] T. M and L. PL, *Nasal Polyps: Origin, Etiology, Pathenogensis, and Structure*, ch. Diseases of the sinuses: diagnosis and management, pp. 57–68. 2001.

[106] R. T. Loder, "The demographics of playground equipment injuries in children," *J Pediatr Surg*, vol. 43, pp. 691–699, Apr 2008.

[107] A. W. Howard, C. Macarthur, L. Rothman, A. Willan, and A. K. Macpherson, "School playground surfacing and arm fractures in children: a cluster randomized trial comparing sand to wood chip surfaces," *PLoS Med*, vol. 6, Dec 2009.

[108] K. C. Chen, S. W. Hung, V. K. Seow, C. F. Chong, T. L. Wang, Y. C. Li, and H. Chang, "The role of emergency ultrasound for evaluating acute pyelonephritis in the ed," *Am J Emerg Med*, Apr 2010.

[109] G. Savova *et al.*, "Towards temporal relation discovery from the clinical narrative," in *AMIA Annu Symp Proc.*, pp. 569–572, 2009.

[110] J. Denny *et al.*, "Extracting timing and status descriptors for colonoscopy testing from electronic medical records," *J Am Med Inform Assoc.*, vol. 17, pp. 383–8, Jul-Aug 2010.

[111] S. Concaro *et al.*, "Mining health care administrative data with temporal association rules on hybrid events," *Methods Inf Med*, vol. 50, Dec 2010.

[112] R. Bellazzi *et al.*, "Methods and tools for mining multivariate temporal data in clinical and biomedical applications.," in *Conf Proc IEEE Eng Med Biol Soc.*, pp. 5629–32, 2009.

[113] B. Reis *et al.*, "Longitudinal histories as predictors of future diagnoses of domestic abuse: modelling study," *BMJ*, 2009.

[114] S. Concaro *et al.*, "Temporal data mining for the assessment of the costs related to diabetes mellitus pharmacological treatment," in *AMIA Annu Symp Proc.*, pp. 119–123, 2009.

[115] H. Svanstrom *et al.*, "Temporal data mining for adverse events following immunization in nationwide danish healthcare databases.," *Drug Saf.*, vol. 33, pp. 1015–25, Nov 2010.

[116] C. Plaisant *et al.*, "Searching electronic health records for temporal patterns in patient histories: a case study with microsoft amalgaa," in *AMIA Annu Symp Proc. 2008*, pp. 601–605, 2008.

[117] R. Moskovitch and Y. Shahar, "Medical temporal-knowledge discovery via temporal abstraction," in *AMIA Annu Symp Proc.*, pp. 452–456, 2009.

[118] R. Kukafka *et al.*, "Redesigning electronic health record systems to support public health," *J. of Biomedical Informatics*, vol. 40, pp. 398–409, August 2007.

[119] L. Martino and S. Ahuja, "Privacy policies of personal health records: an evaluation of their effectiveness in protecting patient information," in *Proc. of 1st ACM Intl. Health Informatics Symp.*, (New York, NY, USA), pp. 191–200, ACM, 2010.

[120] Y. Yan, G. Fung, J. G. Dy, and R. Rosales, "Medical coding classification by leveraging inter-code relationships," in *Proc. 16th KDD'10*, (New York, NY, USA), pp. 193–202, ACM, 2010.

[121] A. Perotte, "Using the Entropy of ICD9 Documentation Across Patients to Characterize Using the Entropy of ICD9 Documentation Across Patients to Characterize Disease Chronicity." AMIA Annual Symp., Nov 2010.

[122] E. Jamrozik, N. de Klerk, and A. Musk, "Clinical review: Asbestos-related disease," *Intern Med J*, Feb 2011.

[123] S. Muthukrishnan, "Data streams: Algorithms and applications," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 2, 2005.

[124] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, pp. 51–55, March 2003.

[125] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pp. 346–357, VLDB Endowment, 2002.

[126] T. Calders, N. Dexters, and B. Goethals, "Mining frequent itemsets in a stream," in *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 83–92, IEEE Computer Society, 2007.

[127] A. Mueen and E. Keogh, "Online discovery and maintenance of time series motif," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010.

[128] J. Kleinberg, "Bursty and hierarchical structure in streams," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, (New York, NY, USA), pp. 91–101, ACM, 2002.

[129] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2004.

[130] D. Patnaik, S. P. Ponce, Y. Cao, and N. Ramakrishnan, "Accelerator-oriented algorithm transformation for temporal data mining," in *NPC '09: Proceedings of the 2009 Sixth IFIP International Conference on Network and Parallel Computing*, (Washington, DC, USA), pp. 93–100, IEEE Computer Society, 2009.

[131] Y. Cao, D. Patnaik, S. Ponce, J. Archuleta, P. Butler, W.-c. Feng, and N. Ramakrishnan, "Towards chip-on-chip neuroscience: fast mining of neuronal spike streams using graphics hardware," in *CF '10: Proceedings of the 7th ACM international conference on Computing frontiers*, (New York, NY, USA), pp. 1–10, ACM, 2010.

[132] M. Socolich, S. W. Lockless, W. P. Russ, H. Lee, K. H. Gardner, and R. Ranganathan, "Evolutionary information for specifying a protein fold," *Nature*, vol. 437, pp. 512–518, Sep 2005.

[133] W. P. Russ, D. M. Lowery, P. Mishra, M. B. Yaffe, and R. Ranganathan, "Natural-like function in artificial ww domains," *Nature*, vol. 437, pp. 579–583, Sep 2005.

[134] J. Thomas, N. Ramakrishnan, and C. Bailey-Kellogg, "Graphical models of residue coupling in protein families," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. 5, pp. 183–197, April-June 2008.