Discovering Inter-Class Temporal Association Patterns

ABSTRACT

The entities involved in many chronologically ordered transactional data logs fall into two disjoint classes such as sellers and buyers, customers and items, etc. In many of these data logs the transactions among same pair of entities recur, e.g., a customer buying an item regularly. In these entity-to-entity temporal (EET) logs, the recurring relationship between the entities may expose interesting temporal patterns in the data. In this paper, we present a novel problem of discovering such temporal patterns which we call Temporal Association Patterns (TAPs) in EET logs. By analyzing TAPs, actionable insights in the data can be exposed. TAPs are discovered by partitioning EET logs into a sequence of log chunks, each chunk modelled as a bipartite graph. The sequence of bipartite graphs resulting from the EET logs is analysed to discover TAPs. Our technique to discover TAP is scalable; experiments, over real as well as synthetic data, show that our technique is efficient and TAPs enable us identify many interesting temporal patterns in the data.

1. INTRODUCTION

1.1 Temporal Association Patterns (TAPs)

Example 1: Consider the call detail records (CDRs) of a telecom company after it has rolled out a new tariff plan. There will be two categories of users in the CDRs: (a) who have subscribed to the plan and (b) who have not. Users may also change their status from (a) to (b) and vice versa but they cannot exist in both the categories simultaneously. Naturally, the temporal patterns in the calls between the users belonging to the two categories and its impact on revenue, is of extreme interest [19]. Suppose, we partition the CDR data into 12-hour chunks or *quanta,* 6AM-6PM (day period), 6PM-6AM (night period). A 12-hour period constitutes one *quantum* (τ). If there are multiple calls between a category (a) user and a category (b) user in a single quantum, we consolidate them into a single record. Each record has associated attributes such as call revenue, call duration, etc. The telecom company also has the user profiles in its database. The analysis of this data exposes interesting temporal patterns such as: i) a subset of category (b) users generate continuously declining revenue for calls to category (a) users; ii) for users within age group [25-35], average revenue through calls from category (a) to category (b) users during the 'night period' has increased but average call duration has increased more substantially, etc.

These patterns emerge as two subsets of users, belonging to two distinct classes respectively, exhibit a *distinct* temporal trend. The temporal pattern, associating these sets of users together is termed *Temporal Association Pattern* (TAP). TAPs expose a new class of insights that are distinct from traditional CDR mining [19].

The CDR data in this example has three main characteristics:

i) Entities, i.e., users, in the CDR data can be divided into two distinct classes c_1 and c_2 (category (a) is class c_1 and (b) is c_2).
ii) Each entity is uniquely identifiable.
iii) Between them, entities have multiple interactions, ordered chronologically in the log.

Besides CDR data logs, many real-world transactional and interactional data logs possess such characteristics. For instance, transaction data about customers (c_1) buying items (c_2), sellers (c_1) selling to buyers (c_2), and members of a social media group (c_1) interacting with non-members (c_2) are some of the example of such data logs. The entities are distinguishable through their unique *ids* and typically entities also have one or more distinguishable features (age, gender, interests, size, color, category, etc.). These logs are ordered chronologically and each interaction/transaction is associated with at least one measurable quantity, e.g., number of items bought, the transaction amount, call revenue/call duration, length of the message, etc. Data logs possessing such characteristics are termed *entity-to-entity temporal* (EET) data logs.

In this paper, we motivate the need to solve a novel problem of discovering a new class of patterns, Temporal Association Patterns (TAP), over EET data logs. We show that TAPs expose hidden micro or macro temporal patterns in the data. Such temporal patterns, if discovered, can be exploited for many purposes such as targeted marketing, identifying potential users to acquire, root cause analysis, predicting user churn, etc.

Example 2: We studied real data pertaining to sales of a leading pharmaceutical company, comprising more than 273,000 transactions, spread over a period of several months. The entities involved in each transaction belong to two distinct classes, namely, sales agents and accounts (doctors and hospitals). The data comprised transactions involving over 1800 unique sales agents (SAs) and more than 22000 unique accounts. Transaction records were ordered chronologically and contained the amount of sales (a quantifiable attribute), drugs sold, etc. Therefore, the sales data conforms to the EET data log characteristics.

The sales agents worked in 364 small geographical units spread over a contiguous region. The quantum size τ was fixed at 8000 transactions per quantum. The temporal patterns discovered over this data included one in which a subset of seven sales agents observed a continuous decline in their sales volume. When we analyzed the profile of these agents, it was found that six of them were dealing with the same subset of nine drug brands (out of more than 400 brands) and further their operations were confined to two closely located geographical units (out of 364). Perhaps some local factors in that geography along with the fact that a set of agents were focusing on the same drugs within small geography were impacting sales of particular types of drugs. With this insight in hand, the pharmaceutical company could address the problem more effectively.

Why is the discovery of TAPs important? For the TAP in Example 2, a small subset of sales agents had shown a decline in their sales. There is nothing unusual about it in a group of 1800 sales agents. However, the pattern became interesting *because* of the fact that these agents belonged to same geographical region and were selling similar drugs. Discovery of TAPs is important because TAPs help us find such insights. The insight was discovered in the first place because the agents could be grouped together. Associating a subset of entities (seven agents among 1800 agents), showing similar

temporal trends in their sales data*, in absence of any specific user query*, is non-trivial. It is an example of how TAPs enable discovery of useful insights in the data, hitherto not studied.

In this paper, we show how to discover this new class of patterns. To the best of our knowledge, ours is the first work to recognize and discover inter-class temporal association patterns.

1.2 Research Contributions

- We present a novel problem: discovering inter-class *temporal association patterns* (TAPs) among entities. TAPs expose insights in the context of temporal data logs. We argue about the novelty of our work in the context of prior art in Section 2.
- We model the problem of discovering TAPs as one of identifying patterns in a sequence of bipartite graphs. We discuss this translation in Section 3. Formal problem statement and how to discover insights from TAPs is explained in Section 4. Our approach is generic and is applicable to a wide class of data.
- Our technique to discover TAPs is scalable with bounded computational and memory overheads. A highly efficient algorithm to discover TAP is presented in Section 5. In Section 6, we present a method to discover similar and recurring TAPs.
- Our empirical results, presented in Section 7, on real and synthetic data show that we are able to identify useful insights by discovering TAPs.

2. RELATED WORK

Association Rule Mining (ARM): ARM has been an active area of research for long. In ARM, the objective is to analyze transactional data to identify weighted set called *association rules* among entities in an efficient manner [13][14][15]. However, there are significant differences between ARM and the problem of discovering TAPs such as: a) In ARM, the association is among the entities belonging to the same class (among a subset of items) and not between entities belonging to different classes (say among customers and items); b) Association rules are true for entire dataset. ARM does not identify 'local' patterns confined to a time window, whereas, TAPs could be 'local'. For instance, a TAP could be true only from quantum i to quantum $i+n$; c) Association rules do not expose temporal patterns in the data. An association rule is a weighted set of items, where as a TAP is a pattern.

A straightforward approach using ARM techniques will not work for discovering TAPs. Let us say, to apply ARM based techniques, we assign a unique id to each pair of entities. Thereafter we apply ARM based techniques but; 1) we still do not know which pair of entities will constitute a transaction that leads to discovery of TAPs and 2) more importantly, an ARM based technique may discover a set of items but is not designed to identify temporal trends.

Temporal Rule Mining: The term "temporal association rules" occurs in [10][11]. However, the meaning of *temporal* in this class of work is completely different. A technique is presented in [10] to identify patterns in a sequence log where each item in the log sequence has a certain *exhibition period.* The paper deals with the issue that items published earlier, have an unfair advantage over the items that are exhibited later. In [11], authors focus on mining event sets. Each variable in the set representing an event is bound with a time constraint such as; two variables cannot be more than a given distance apart in time. The objective is to search for patterns that maintain a partial order between its entities. There are clear differences between this work and ours.

Existing work on *temporal association rule* mining cannot handle the problem of discovering inter-class temporal association patterns seamlessly. Existing work can be broadly divided into three categories [1]:

A) Sequential patterns mining: In this class of work, for a collection of transactions, the objective is to discover ordered sequence of items that occur in sufficiently many of those transactions [2][3][4].

B) *Frequent Episodes*: In the frequent episode mining framework, the data is given a single long sequence and the objective is to unearth temporal patterns (called episodes) that occur sufficiently often in the long sequence[5][6].

C) Patterns with explicit time constraints: The difference between discovering *frequent episodes* and this work [7][8][9] is that there is an additional time constraint over items involved in a pattern, i.e., no two items in a pattern are more than t units apart in the pattern where t is the time difference between two items in a data sequence.

The objective of each of these works is unrelated to the discovery of TAPs. For example, in sequential pattern mining, the sequence is defined between the entities within a single transaction. Further, with these techniques, all the differences as w.r.t. ARM apply as well. By discovering TAPs, we identify many micro and macro trends in data, discovering interesting insights, hitherto not studied.

Temporal Graph Mining: There is a large body of work on temporal graph mining. In [20], the goal of the authors is to find most central nodes in a time evolving bipartite graph. The goal of dynamic tensor analysis method in [21], involving higher order data, i.e., data with multiple classes and not just two, is to provide compact summery over data. The goal of these techniques is completely at variance with our goal of discovering TAPs.

3. PREPARING EET LOGS

Each log record r in the EET log D is a tuple <$\boldsymbol{l}$**, u, v, attr**> where $r.\boldsymbol{l}$ is the timestamp of record r, $r.\mathbf{u}$, $r.\mathbf{v}$ are set of entities and $r.$**attr** is the set of attributes associated with r respectively. $\mathbf{U} = \bigcup_{i=1}^{|D|} r_i.\mathbf{u}$, $\mathbf{V} = \bigcup_{i=1}^{|D|} r_i.\mathbf{v}$. $\mathbf{U}$, $\mathbf{V}$ are the set of entities belonging to two distinct classes. Call duration, message length, transaction amount, discount rate, etc., are examples of attributes in set **attr** for different types of EET logs. For a given EET log data D all the log records contain the same set of attributes (although some attributes may be null for a log record). The relation between the entities, belonging to two classes, in a log record r of an EET log D could be one-to-one, e.g., each log record contains one seller and one buyer; or one-to-many, e.g., each log record contains a customer and many items; or many-to-many, e.g., each log record contains the details of an interaction between the members of an online group with people outside that group.

3.1 Project, Quantize and Map

To discover TAPs, the EET log data goes through a simple three step process; project, quantize and map. Users specify three parameters:

1) a **projection** function $\boldsymbol{\pi}$ **(**σ, T**,** D_{raw}**)**, to project a subset of log records in a given EET data log D_{raw}. The raw EET log data, D_{raw} is projected as $D = \boldsymbol{\pi}$ (σ, T**,** D_{raw}). For instance, in Example 1, TAP (ii) is discovered over a subset of log records pertaining to only the 'night period'. Specifying the projection function is optional. σ is a trigger condition that takes either time and/or attribute values of a log record as input. If σ becomes true at time t, the log records in time window t to (t+T) are projected. For TAP (ii) in Example 1, T=12-hours. Trigger condition σ becomes true at 6PM on each day.

2) a **quantization** parameter **τ,** specifying the time granularity at which the temporal association between the entities is tracked (in Example 1, τ is 12-hours). The EET log is divided into a sequence of quanta of duration **τ.** Each quantum contains log records from a non-overlapping time period of τ time units (the last quantum could be smaller, as shown in Figure 3). The time granularity τ of a *quantum* for a given data log is based on the data characteristics and user requirements. One can discover different TAPs by varying τ. For a given τ, let the EET log data D result in m non-overlapping partitions D_is of τ time units; $\bigcup D_i = D; (1 \le i \le m)$ and $\forall D_i, D_j; D_i \cap D_j = \phi;\ 1 \le i, j \le m$. For log records r, q; $r \in D_i$ and $q \in D_j$, if r has occurred before q in D, then $i \le j$.

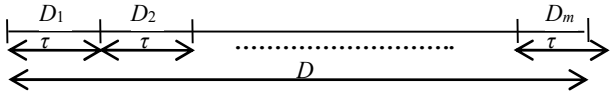


Figure 3. EET log D is divided at time granularity τ

Once data is partitioned in quanta based on τ, if there are multiple log records *within a quantum*, involving a given pair of entities u and v, they are aggregated and are treated as one single record. Hence, this step is called the *quantization* and *aggregation* step.

3) a state **mapping** function **ψ**. The attribute(s) of an aggregated record r for a given pair of entities (u, v) *within a quantum* are mapped to a state S_i ($S_i \in [S_1..S_s]$) with the aid of the user-specified function **ψ**; S_i represents the state of that pair of entities (u, v) in that quantum. With discrete state levels, 'almost similar' records are mapped to the same state. This step is essential for finding temporal patterns. Function **ψ** is similar to a user query. TAPs are discovered for the specified **ψ**. If a new **ψ** is specified, a different set of TAPs is discovered over the same EET data log.

Example 3: In Example 1, suppose there are two attributes for a log record in the EET log, call duration c_d and call revenue c_r. Suppose a user is interested in studying the temporal patterns with respect to only c_d. A user specified mapping function **ψ** (.) from the call record to a state is as follows: SHORT (S_1) if $c_d \le \theta_1$, MEDIUM (S_2) if $\theta_1 < c_d \le \theta_2$ and LONG (S_3) if $c_d > \theta_2$. **ψ** (.) ignores c_r attribute and maps the attribute(s) of a log record to one of the three states based on the user specified function in her query. A different **ψ** (.) may use both c_d and c_r, say to study the temporal patterns in users' 'call behavior' and its impact on revenue.

3.2 Modeling Data Logs as Bipartite Graphs

For a bipartite graph G (V, E), $V = V_1 \bigcup V_2$, $V_1 \bigcap V_2 = \phi$ and $\forall (u,v) \in E; u \in V_1 \wedge v \in V_2$ where (u, v) is an edge between nodes u and v in the graph. Since entities in a log record belong to two distinct classes, each partition of EET data log D_k ($1 \le k \le m$) is represented as a bipartite graph and each log record r in D_k is represented by one or more edges in this graph, as described below. Therefore the complete EET data log is represented as a sequence of bipartite graphs (Figure 2).

For partition D_k, we construct a bipartite graph G_k (V_k, E_k) as follows: An entity u (v) is represented as a node in G_k. For a log record r, next in the sequence in D_k, for each entity $u \in r.\mathbf{u}$ and $v \in r.\mathbf{v}$, if corresponding entity nodes u and v exist in the graph G_k, edge $e^k_{(u,v)}$ is added between them otherwise; we first add the node(s) u and/or v in the graph before adding the edge. $e^k_{(u,v)}$ denotes the edge between entities (u, v), in k^{th} bipartite graph G_k in the graph sequence. If the edge already exists in G_k, the values of the attributes associated with the edge are incremented by the corresponding attribute values in log record r. Thus, we aggregate the transactions between a given pair of entities (u, v) within a quantum. Since for each $u \in r.\mathbf{u}$, $v \in r.\mathbf{v}$, an edge (u, v) is added/updated in the graph with the attribute values in $r.\mathbf{attr}$, a total of $|r.\mathbf{u}|.|r.\mathbf{v}|$ edges are added or updated in graph G_k. Bipartite graphs G_ks ($1 \le k \le m$) are undirected. Hence, the TAPs discovered over the bipartite graph sequence will be the same even if entities in set **U** and set **V** swap their positions in the graph.

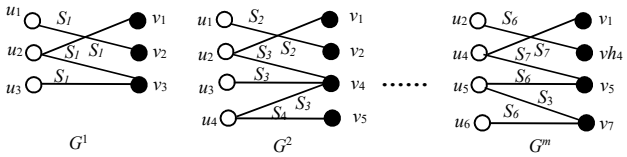


Figure 2. A Sequence of Bipartite Graphs

Each edge $e \in E_k$ in the graph G_k (V_k, E_k) is mapped to a discreet state $S \in [S_1..S_s]$. Edge state is a function of its attributes; state is assigned to an edge based on a user specified mapping function.

$$e.S = \boldsymbol{\psi}(e.\mathbf{attr}) \mid e.\mathbf{attr} \rightarrow S \mid S \in [S_1..S_s] \qquad (1)$$

$e.\mathbf{attr}$ is the list of attribute values associated with edge e. **ψ** is a user specified function. **ψ** is dependent on the data. Our technique to discover TAPs is applicable on a wide variety of data logs. Therefore, it is imperative that users define the function **ψ (.)** according to the data characteristics. For different **ψ** functions, different TAPs can be discovered over same data.

For a pair of entities (u, v), if their records are mapped to a sequence of states, spread over consecutive graphs $G_k G_{k+1}..G_{k+n-1}$, that sequence represents a temporal pattern of length n.

Example 4: For the user specified function in Example 3, let sets of entities **u** and **v** follow a *same* sequence of states "$S_1S_1S_2S_2S_3S_3$". The sequence shows that the entities associated with the sequence display a temporal pattern of increasing call duration. Let this sequence of states be represented by P; $|P|=6$. $P.\mathbf{u}$ and $P.\mathbf{v}$ represent the set of entities associated with P.

4. DEFINITION of TAPs

Let an EET log be represented as a sequence of m bipartite graphs for a time granularity τ. Further, each edge in a graph is assigned a state $S \in [S_1..S_s]$ for a given **ψ**. Let $u \in \mathbf{U}$ and $v \in \mathbf{V}$ be a pair of nodes (entities) in the EET graph sequence. If there are recurring interactions between u and v spread over a sequence of quanta, length of the sequence be n, there exist an edge $e^{k+l}_{(u,v)}$ in each graph G_{k+l}; $0 \le l \le n\text{-}1$. $e^{k+l}_{(u,v)}.S$ represents the state associated with that edge in graph G_{k+l}. Therefore, the temporal pattern of interactions between entities u and v is captured by the sequence of these states. This sequence represents a TAP. Let the set of entities, $\mathbf{u} \subset \mathbf{U}$ and $\mathbf{v} \subset \mathbf{V}$, follow a *same* temporal pattern.

Def. 4.1: For any pair of nodes $\{u, q\} \subset \mathbf{u}$ $\exists$ $\{v, r\} \subset \mathbf{v}$, such that $\{e^{k+l}_{(u,v)}, e^{k+l}_{(q,r)}\} \subset E_{k+l}$ and $e^{k+l}_{(u,v)}.S = e^{k+l}_{(q,r)}.S$; $0 \le l \le n\text{-}1$.

Therefore, any pair of entities $\{u, v\}$, $u \in \mathbf{U}$ and $v \in \mathbf{V}$, is associated with any other pair of entities $\{q, r\}$, $q \in \mathbf{U}$ and $r \in \mathbf{V}$, if both pairs of entities follow the same sequence of states. This is termed *spatial* correlation between the entities. Further, for both the pairs of entities, the sequence starts in the same quantum k. It is called *temporal* correlation. All such pairs of nodes, belonging to two classes, which are temporally and spatially correlated are associated together by virtue of following the same sequence of states, which started in the same quantum. This sequence of

discrete states represents a TAP. The sequence is an arbitrary permutation of states $S_i \in [S_1..S_s]$.

Def. 4.2: A TAP P of length n, starting in *quantum* k, is an ordered sequence of states, $P=\prod_{i=1}^{n} S_i : k \mid S_i \in \{S_1..S_s\}, \beta \le n \le m$; *min* $(|P.\mathbf{u}|, |P.\mathbf{v}|) \ge \alpha$.

Entities $P.\mathbf{u} \subseteq \mathbf{U}$ and $P.\mathbf{v} \subseteq \mathbf{V}$ follow the sequence of states represented by P. $P.\mathbf{u}$ and $P.\mathbf{v}$ are the support sets of P. TAPs are discovered in the context of user specified $\boldsymbol{\pi}$, $\boldsymbol{\psi}$ **(.)** and $\boldsymbol{\tau}$. By varying them, users can obtain different *temporal insights* on the same data.

A TAP is spread over a consecutive sequence of quanta $D_kD_{k+1}..D_{k+n-1}$ $(1 \le k \le m-n+1)$ such that there exist at least α entity pairs that follow the same sequence of states represented by P from quantum k to $k+n-1$. The pattern also includes the *quantum* number k, in which it starts. Each state S_i in the TAP is one of the s discrete states. In Example 4, a TAP $P=S_1S_1S_2S_2S_3S_3|k$, which started in k^{th} quantum. For the $\boldsymbol{\psi}$ in Example 3, this pattern reveals that the call duration between the associated entities is increasing over time.

4.1 Problem Statement

P is called temporal association pattern (TAP), $|P|$ is the length of this pattern. If *min* $(|\mathbf{u}|, |\mathbf{v}|) \ge \boldsymbol{\alpha}$ and $|P| \ge \boldsymbol{\beta}$, such a pattern is of interest to us. $|\mathbf{u}|$ $(|\mathbf{v}|)$ is the cardinality of set $\mathbf{u}$ $(\mathbf{v})$. $\boldsymbol{\alpha}$ is the support and $P.\mathbf{u}$ and $P.\mathbf{v}$ are the support set for P. Since a TAP represents a temporal pattern, it is imperative to specify their minimum length $(\boldsymbol{\beta})$ to identify meaningful trends.

Thus, our objective is to identify all TAPs of length $\ge \boldsymbol{\beta}$ with support of at least $\boldsymbol{\alpha}$, i.e., *min* $(|\mathbf{u}|, |\mathbf{v}|) \ge \boldsymbol{\alpha}$. Specifically:

Problem Definition**:** *For a given EET data log comprising log records connecting two disjoint sets of entities* ***U*** *and* ***V****, a user specified function* $\boldsymbol{\psi}$ **(.)** *and parameter* $\boldsymbol{\tau}$*, how do we identify the set of temporal association patterns* (*TAPs*) ***P*** *between entities such that* $\forall P \in \mathbf{P}$*, min* $(|P.\mathbf{u}|, |P.\mathbf{v}|) \ge \alpha$ *and* $|P| \ge \beta$.

Without loss of generality, $\forall u \in P.\mathbf{u}, \exists v \in P.\mathbf{v}$, such that u and v follow the sequence of states represented by pattern P.

$\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ are set to their default values but users can also specify them. Hence, besides the optional parameters ($\boldsymbol{\pi}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$), quantum size $\boldsymbol{\tau}$ and mapping function $\boldsymbol{\psi}$ (.) are the only inputs needed by our algorithm to discover TAPs

Other considerations involved in identification of TAPs include:

1. Firstly, only the longest sequence followed by a given set of entities is a TAP. For example, for the P shown above, another pattern $P' = S_1S_2S_2S_3S_3$ is not a TAP *if* (1) both the patterns, P and P' are concurrent (cf. Def. 5.2.1) and (2) the set of entities ($P.\mathbf{u}$ and $P.\mathbf{v}$) in P and set of entities ($P'.\mathbf{u}$ and $P'.\mathbf{v}$) in P' are exactly the same, i.e., $P.\mathbf{u} = P'.\mathbf{u}$ and $P.\mathbf{v} = P'.\mathbf{v}$. However, if $P.\mathbf{u} \neq P'.\mathbf{u}$ and/or $P.\mathbf{v} \neq P'.\mathbf{v}$ then P' is a valid TAP. In Section 5, we present a highly efficient methodology to identify all the longest TAPs with unique support sets.
2. Secondly, for many applications, users may want to associate the entities with the same TAP if only a minor deviation is observed in their temporal behavior. For a given set of TAPs $\mathbf{P}$ and a given deviation threshold λ, identifying a minimal set of TAPs $\mathbf{P_s}$, such that each TAP in $\mathbf{P}$ is within λ deviation threshold of a TAP in $\mathbf{P_s}$, is shown to be NP-Complete. In Section 6.1, we present an algorithm to discover similar TAPs with the best approximation bounds.
3. Thirdly, a TAP could be recurring, i.e., the same pattern occurs for different sets of entities in a temporally shifted manner in the data. In some applications, users may be interested only in recurring TAPs to expose insights developed over long term observations. In Section 6.2, we present our technique to identify recurring TAPs.

Table 1. Notation

Symbol	***Definition***
τ	Time granularity at which the EET log D is divided.
m	Number of quanta into which a EET log D gets divided for time granularity τ.
D_k	k^{th} partition for EET data log D.
$G_k(V_k, E_k)$	Bipartite graph, representing the D_k^{th} partition.
$\mathbf{U}$, $\mathbf{V}$	Two disjoint sets of entities in an EET data log.
u (v)	$u \in \mathbf{U}$ ($v \in \mathbf{V}$) is an entity belonging to set $\mathbf{U}$ ($\mathbf{V}$)
r	A log record in the EET log, represented as a tuple <l, **u, v, attr**> where $r.\mathbf{u} \subset \mathbf{U}$ and $r.\mathbf{v} \subset \mathbf{V}$
$\boldsymbol{\psi}$ **(attr)**	A user specified state mapping function that maps **attr** to a state $S \in [1\ldots s]$.
P	A TAP, represented as a concatenation of states S_is for a given $\boldsymbol{\psi}$ such that $S_i \in [S_1..S_s]$
$P.\mathbf{u}$, $P.\mathbf{v}$	Set of entities associated with TAP P.
$\mathbf{P}$	Set of the TAPs discovered for a given state mapping function $\boldsymbol{\psi}$ **(.)** and time granularity τ.
α	Support threshold for a TAP pattern P
β	Minimum length of a TAP pattern P
λ	TAPs within λ deviation of each other are merged in a single TAP.

4.2 Discovering Insights from TAPs

Based on the principal of Occam's Razor, the probability of a large enough number of entities following a long enough temporal behavior due to their random congruence is small. Very often, the temporal behavior is guided by a rational exercise of choices, i.e., there exist distinguishing features for entities in sets $P.\mathbf{u}$ and $P.\mathbf{v}$ associated with a TAP P due to which they follow the temporal trend, represented by the TAP.

Entities in set $\mathbf{U}$ (set $\mathbf{V}$) are distinguishable from each other through a set of feature $\mathbf{F_U}$ ($\mathbf{F_V}$). Entity features help us expose deeper insights in the data. In Example 1, for TAP(ii), the entities (i.e., users) are found to belong to a particular age group. In Example 2, sales agents were found to belong to a particular geographical region and also dealing in a subset of drugs. These are called distinguishing features ($\mathbf{F_d}$) of entities, linked together with the aid of a TAP; $\mathbf{F_d} \subseteq (\mathbf{F_U} \cup \mathbf{F_V})$. For the entities linked with a TAP, it is relatively easy to find their most distinguishing features, as we explain below. A feature can either have values in a discrete range (e.g., gender), in a continuous range (e.g., price) or text (e.g. place).

The details of the process to find the distinguishing features are beyond the scope of this paper. However, for completeness, we describe the process to determine if a given feature, containing values from a continuous range, is a distinguishing feature.

Without loss of generality, let η be a feature for entities in set $P.\mathbf{u}$ such that η has values from a range $\eta.R$ ($\eta.R = \eta.max - \eta.min$); $\eta.max = \arg\max_{u \in P.\mathbf{u}}(u.\eta)$ where $u.\eta$ is the value of η for entity u. We follow the 80-20 rule to decide if η is a distinguishing feature, i.e., if a sub-range of length 20% of $\eta.R$ contains 80% of the entities in set $P.\mathbf{u}$, η is a distinguishing feature. Following are the steps:

Step 1: Divide the range $\eta.R$ in 100 parts. Create a histogram containing count of entities in each part.

Step 2: Pointers l_p (r_p) point to left (right) end of the histogram; l_p=1 and r_p=100. Fraction of entities *fr* in range (r_p-l_p+1) is 1.0.

Step 3: while (*fr*≥0.8**)** *fr=computeFraction*(l_p++, r_p--);
do {
fr_1= *computeFraction*(l_p+1, r_p);
fr_2= *computeFraction*(l_p, r_p-1);
if (fr_1>fr_2 & fr_1≥ 0.8) { l_p++; *fr*=fr_1}
else if (fr_2≥ 0.8){ r_p--; *fr*=fr_2}
} **while** (*fr*≥0.8)
if (r_p-l_p+1≤20) *addFeature* (*η*, l_p, r_p); //applying 80-20 rule;

computeFraction (.) computes the fraction of entities in set *P*.**u** in the range l_p, r_p. *addFeature* (*η*, l_p, r_p) adds the feature and predicate, (i.e., *η* belongs to the range corresponding to l_p, r_p) to set $\mathbf{F_d}$. Thus, we find the smallest range with at least 80% entities in set *P*.**u**.

A similar process can be applied for different features types (i.e., with discrete values or text values). Identifying deeper insights for a TAP is O (|$\mathbf{F_U}$| + |$\mathbf{F_V}$|). Set $\mathbf{F_d}$ contains distinguishing features and their predicate values, thus exposing deeper insights. However, the prerequisite to find such insights is to first discover TAPs. A TAP may be discarded if it exposes no insights, i.e., its $\mathbf{F_d}$ is empty.

5. IDENTIFICATION OF TAPs

To discover TAPs, we essentially 'cluster' the entities together, belonging to two distinct classes, which are following the similar temporal patterns. We now present the details of our algorithm.

5.1 Basic TAP Algorithm

For an EET data log, divided in *m quanta*, the algorithm to find TAPs runs in *m* phases. In phase 1, we initialize *s* buckets, corresponding to *s* states. While processing G_1, we add an edge in a particular state in its respective bucket. Therefore, we have at most *s* different buckets representing a total of *s* different pattern types, of length 1 each. We discard the buckets which contain less than *α* edges (*α* is the support threshold).

In phase 2, each non-empty bucket from phase 1 is subdivided into *s* buckets again and each edge is put into its respective bucket as per the state of that edge in G_2. At this stage, we have a maximum of s^2 buckets, containing patterns of length 2. Edges (or associated entities) in the same bucket at this stage have exactly same state for last 2 *quanta* (in phase 1 and phase 2). Again, the buckets with less than *α* entries are discarded.

The process continues till phase *m*. The number of buckets may continue to increase till $s^m > E_{max}/\alpha$. E_{max} is the maximum number of edges in a bipartite graph in the bipartite graphs sequence; E_{max} = $\text{argmax}_k(|E_k|)$; $1 \le k \le m$ where E_k is the set of edges in graph G_k (V_k, E_k). If the minimum support needed for a TAP is *α*, maximum possible distinct buckets could be at most E_{max}/α. Non-empty buckets in phase *k* group the similar patterns together of length *k*.

We must highlight that:

1) Number of distinct buckets and length of a TAP are independent of each other. Though the number of distinct buckets is bounded by E_{max}/α, the pattern length is not. For instance, if all the edges in a graph sequence follow the same pattern, they will always be placed in one bucket, irrespective of pattern length.

2) A TAP may start from any of the graph instances G_k ($1 \le k \le m$). Therefore, to capture all the patterns, the patterns are tracked during each phase. Hence in phase 2, there are at most s^2 different patterns of length 2 but we also initialize *s* additional buckets to capture patterns of length 1 starting in phase 2 and so on and so forth (cf. Appendix A for analysis).

The pseudo code of the algorithm is given below.

Algorithm 1 TAP_Basic

Input: The EET Sequence *D*;
QuantumSize *τ*;
Minimum Pattern Length *β*;
Minimum Pattern Support *α*;
Output: A patternTree containing all the TAPs with support more than *α*.

1. PatternsTree(key, edgeSet) ←Φ /*HashTree containing patterns and associated edges */
2. Partitions $P \leftarrow D.length/\tau$;
3. Graph $G \leftarrow \Phi$
4. FuzzyMatching ← λ'

Procedure *TemporalPatternDoscovery*

5. **For** i←0 to P {
6. **While** (Partition(i)) {// we are in partition i
7. *t* ←ReadNextRecord();
8. $G \leftarrow$ updateGraph (*t*.EdgeSet, *t*.attributes, FuzzyMatching);
9. FindPatterns(*G, i*);
10. CleanupPatternsL(*β*);//Remove Patterns with length <*β*
11. **Return** *PatternTree*;

Procedure *FindPatterns* (*G*, *i*)

12. **For** j ← 0 to G_i.size() { //Graph G_i
13. Edge $e \leftarrow G_i$.next();
14. *e*.*S* ← findstate(*e*.attributes)/*Assign a state*/
15. *P* ← findPattern (*e*);
16. **If** (*P* != NULL) //Edge $e_{(u, v)}$ is part of a pattern
17. $P \leftarrow P + e.S$;
18. **Else**
19. $P \leftarrow e.S$; //A new pattern starts
20. *PatternTree*.add(*P*, edgeSet.add(*e*))
21. CleanupPatternsS(*α*); //Remove Patterns with Support<*α*
22. UpdateGraph () // Update the graph metadata collected so far.

Figure 4. Algorithm to identify patterns in an EET log

Function *FindPattern*(*Edge*) finds the pattern corresponding the edge *Edge* between two entities in the bipartite graph.

The memory overhead of TAP_Basic is bounded by O (E_{max}). The analysis of TAP_Basic is given in Appendix A. Further, the time complexity of the algorithm is independent of number of TAPs found in an EET log. It depends only on the number of edges in the bipartite graph sequence and the length of the log.

Next, we highlight the shortcomings of basic TAP discovery algorithm and present our methodology to improve it.

5.2 Improved TAP Algorithm

With the TAP_Basic algorithm, exactly the same set of entities, following a sequence of states, result in multiple TAPs as follows: Suppose in a given *quantum k*, edges {e_1, e_2,…,e_n} were in same state S_i and hence were grouped together in a pattern $P_u = S_i$. In quantum *k*+1, suppose these edges were again in a similar state, state S_j, and the pattern P_u is extended to P_u=S_iS_j. Let us say no other edge, besides these, is in state S_j. However, in TAP_Basic, an additional pattern P_v=S_j is initialized with exactly the same edges. Therefore, if there is a TAP of length $\beta' \ge \beta$, TAP_Basic ends up discovering (β'-β+1) patterns (β is minimum patterns length), with each subsequent pattern a suffix of its previous one. Since all these patterns have emerged due to the same set of edges, only the longest of these pattern is the correct TAP.

Example 5: For the TAP *P* in Example 4, if β =2, TAP_Basic will discover 5 TAPs, "$S_1S_1S_2S_2S_3S_3$", "$S_1S_2S_2S_3S_3$", "$S_2S_2S_3S_3$", "$S_2S_3S_3$" and "S_3S_3" all because of the same set of entities *P*.**u**, *P*.**v**.

However, if the support set(s) is (are) different for any of the patterns, then it is a valid pattern. For instance, if a new edge, e_{n+1}, is in state S_j in *quantum k*+1, P_v=S_j is a valid pattern. We capture these observations formally below:

Let $P_i.b$ is the quantum in which pattern P_i begins and $P_i.e$ is the quantum in which it ends; $1 \le P_i.b < P_i.e \le m$.

Def. 5.2.1 Concurrent Patterns: Two patterns P_i and P_j are concurrent if, without loss of generality (w.l.g.), $P_i.b \leq P_j.b$ and $P_i.e \geq P_j.e$.

Def. 5.2.2 Maximally Longest Pattern: Let **P** be a set of concurrent patterns; $\forall P_i, P_j \in \mathbf{P}$, $P_i.\mathbf{u} = P_j.\mathbf{v}$ and $P_i.\mathbf{u} = P_j.\mathbf{v}$. Let $|P_l| > |P_j| \ \forall P_{j \neq l} \in \mathbf{P}$; $P_l \in \mathbf{P}$. P_l is the maximally longest pattern.

Please note, concurrent patterns P_i and P_j with exactly the same support base, cannot have equal length (otherwise they would not be reported as two separate patterns in the first place).

The maximally longest pattern, followed by the same set of entities, is a TAP. Conversely, if the entity sets for any two different TAPs are not equal to each other, we ought to have them as two different TAPs. Therefore, at the end of a quantum, we check if any pattern is redundant due to Def. 5.2.2. A naïve method to check this results in $O(n^2)$ set comparisons; n is the number of TAPs in the quantum.

However, we can exploit the fact that for redundant patterns supported by exactly same set of edges, each pattern is progressively a suffix of its previous pattern (P_v is a suffix of P_u). We call it the *suffix property*. We can expedite the process of identifying redundant patterns due to the *suffix property* with the aid of Theorem 1. First we prove a lemma.

For a pattern P let's say E^P represents the set of edges associated with it, i.e. $\forall e_{(u,v)} \in E^P; u \in P.\mathbf{u} \wedge v \in P.\mathbf{v}$.

Lemma 1: For two concurrent patterns, if one is not suffix of the other, w. l. g., the associated edge sets are disjoint.

Proof: Let P_u *and* P_v are two concurrent patterns starting in *quantum k*. Without loss of generality, let $|P_u| \geq |P_v|$ and that P_v is not a suffix of P_u. Since P_v is not suffix of P_u, let $k+i$ ($0 \leq i \leq k+|P_v|$) be the first quantum in which the state of P_u and P_v differs, i.e., $P_u[k+i].S \neq P_v[k+i].S$.

Since the state for two patterns is different in *quantum k+i,* $\forall e \mid e \in E^{P_u}, e \notin E^{P_v}$ and $\forall e \mid e \in E^{P_v}, e \notin E^{P_u}$. Therefore, the respective associated edge sets for both the patterns become disjoint in *quantum k+i.* For each subsequent *quantum j* $E_j^{P_u} \subseteq E_{k+i}^{P_u}$ and $E_j^{P_v} \subseteq E_{k+i}^{P_v}$. Therefore, the sets remain disjoint.□

Theorem 1: Only a pattern P_s, which is a unique suffix of another pattern P, can have the same set of supporting edges.

Proof: For a pattern P, let E^P represents its support set of edges. Let $P_s \supseteq P$ be a suffix of P. If P_s is also a suffix of another pattern P', i.e., $\exists P' \mid P_s \supseteq P'$ (P' is not a suffix of P and vice versa*)*, the support for P_s will be $E^{P_s} = E^P \bigcup E^{P'}$.

Since $P' \neq P$, the edge sets $E^{P'} \neq E^P$ (*Lemma* 1). Hence, $E^{Ps} \neq E^P$ *(or* $E^{P'}$*).* Hence P_s can have exactly same set of supporting edges *only if* it is a unique suffix of another pattern P. □

Hence, for two concurrent patterns, if one is not suffix of the other, the associated edge sets cannot be equal. Therefore, we just compare the subsets associated with only those pairs of TAPs (P_i , P_j) where pattern P_i is a unique suffix of pattern P_j. The complexity of this comparison is down to $O(n)$ (worst case) as there can be at most $\lfloor n/2 \rfloor$ such pairs of patterns from $O(n^2)$ (average case) where n is number of TAPs in set of TAPs **P**.

The algorithm in Figure 4 is modified as follows:

23. CheckRedundantPatterns(PatternTree);

Since our objective is to efficiently identify all such pairs of TAPs $P_i, P_j \in \mathbf{P}$, TAPs are maintained as a forest of tries for reverse TAP patterns. For $P = \prod_{i=1}^{n} S_i : k$, its reverser TAP $rP = \prod_{i=1}^{n} S_{n-i+1} : k$. For $P_1 = S_1S_2S_3$, $rP_1 = S_3S_2S_1$. Forest of tries is shown for s set of TAPs P_1 ($S_1S_2S_3$), P_2 ($S_1S_2S_2S_3$), P_3 ($S_1S_2S_2S_4$) and P_4 ($S_1S_1S_2S_3$) in Figure 5 (a). In this forest, if P_i is a suffix of P_j both TAPs will occur in the same branch. Hence, with the aid of this data structure we can efficiently identify a TAP which is a unique suffix of another TAP in a set of TAPs. P_1 is a unique suffix of P_4 in Figure 5 (a). Details are omitted.

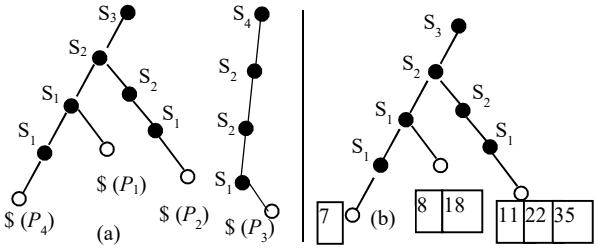


Figure 5: TAP PatternTree

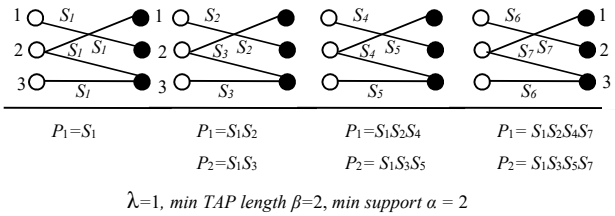
6. SIMILAR TAPs

Example 6: For the CDR data in Example 1, let there be a TAP $P = S_1S_1S_2S_2S_3S_3$ representing an increasing sequence of call durations between entities $P.\mathbf{u}$ and $P.\mathbf{v}$. If there is another TAP P' *concurrent* to TAP P such that $P' = S_1S_1S_1S_2S_2S_3S_3$, P' represents almost similar trend as P. Therefore, a user may want to merge the entities in set $P'.\mathbf{u}$ with $P.\mathbf{u}$ and $P'.\mathbf{v}$ with $P.\mathbf{v}$ together**.** For merged TAP P_m, $P_m.\mathbf{u} = P.\mathbf{u} \bigcup P'.\mathbf{u}$ and $P_m.\mathbf{v} = P.\mathbf{v} \bigcup P'.\mathbf{v}$.

P_m captures the temporal trend represented by P and P'. *How do we identify and represent* P_m*?* We now address this question.

For a given deviation threshold λ and a set of TAPs **P,** if TAPs in a subset of **P** are deviating by at most λ states from each other, they must be merged together into a single TAP. λ is called *Spatial deviation* threshold and is similar to edit distance [18]. We next present how to find similar TAPs with bounded *spatial* deviation.

There are two ways to manage the *spatial deviation*: a) at the *edge level*, i.e., edges that are within the edit distance λ of each other are associated together in a single TAP; b) at the *pattern level*, i.e., among the discovered TAPs, those within λ edit distance of each other are merged together.



λ=1, *min TAP length* β=2, *min support* α = 2

Figure 6: TAP discovered based on 'edge level' deviation

In case of *edge level* deviation, missing edge(s) in the middle of a pattern are marked just another state deviation for the edge.

Both these approaches result in different sets of TAPs, as shown in Figure 6 which has a sequence of 4 bipartite graphs. The figure shows the TAP discovered based on edge level deviations. λ is set to 1, i.e., each edge can deviate at most in one of the *quanta.* In *quantum* 1, all the edges are in state S_1. In *quantum* 2*,* edges are in two different states (S_2 and S_3). Patterns P_1 and P_2 have support of

4 edges each (as each edge is still within the deviation threshold of one from the respective pattern and the pattern has a minimum support of 2). The process continues and we finally discover two patterns with support of 3 each, as shown in Figure 6. $E^{P1} = \{e_{11}, e_{12}, e_{21}\}$ and $E^{P2} = \{e_{12}, e_{21}, e_{33}\}$. Each edge in both the patterns is deviating in exactly one quantum (λ=1).

However, notice that none of the edges follow any of the TAPs end to end. One can actually construct a graph sequence, where for two completely different TAP, except the initial state, the support set of one TAP is a proper sub-set of the support set of the other TAP. Note that it is possible that the discovered TAP(s) may not actually exist in the data (as in this example).

TAPs discovered based on the *pattern level* deviation method, with λ=1, α=2, β (length)=2 are $P_1=S_1S_2$ and $P_2=S_1S_3$. While merging, the state followed by the TAP with greater support is assumed for the merged TAP. Ties are broken arbitrarily (as in this example).

With *edge level* deviation, we see the following shortcomings; a) exactly the same set of edges give rise to multiple TAPs (in Figure 6, P_1 and P_2 in quantum *2* have exactly similar support base); b) if a TAP is discovered due to *edge-level* deviation but not discovered otherwise, the discovered TAP is followed by less than α pairs of entities end-to-end (cf. Lemma 2); and c) deviation among only those edges can be tracked whose initial state is similar. This is an arbitrary restriction. The advantage of TAPs discovery based on *edge-level* deviation is that it helps discovers longer TAPs, exposing more meaningful temporal trends in the data.

For TAPs discovered based on *pattern level* deviation method, edges in the support set of a TAP follow it end to end. Since the TAPs are merged later, the TAPs need not have the same initial state. However, the inherent shortcoming of the *pattern level* deviation method is; since edges must follow the TAP end-to-end, the TAPs discovered based on *pattern level* deviation are shorter and they may fail to expose temporal trends in the data (in Figure 6, length of P_1 and P_2 is small for *pattern level* deviation).

Let P_E be a pattern discovered in the *edge level* deviation method and not discovered in *pattern level* deviation method and that $|P_E|=\beta$. Let $E^{\beta}{}_{PE}$ represent the set of edges associated with P_E. Let α be the support threshold. The initial state for the pattern P_E has to be similar for all the edges, therefore $|E^1{}_{PE}|\geq\alpha$ (otherwise pattern would not have been initialized). Therefore, the same pattern initializes in the *pattern level* deviation method too. Let's call that pattern P_P. Let S_n be the last state common in both P_E and P_P (in quantum k) and let S_{n+1} be the state after that in pattern P_E.

Lemma 2: For a given support threshold α, pattern P_E is followed by less than α edges end to end.

Proof: Omitted □

We adopt a hybrid approach. For a given λ, we first discover the patterns with edge deviation threshold $\lambda' \leq \lambda$. The discovered patterns are merged if they are within $\lambda - \lambda'$ edit distance of each other. If $\lambda' = \lambda$, only *edge level* deviation is captured in the TAPs. If $\lambda' = 0$, only *pattern level* deviation is captured. We first show how to find TAPs with *edge level* deviation followed by how to merge the patterns (to capture *pattern level* deviation).

6.1.1 Edge Level Deviation

In TAP_Basic, in Figure 4, we keep a *fuzzyMatching* counter. With the help of this counter, we discover the TAPs with *edge level* deviation. We set the *fuzzyMatching* counter to λ'. Therefore, similar TAPs, with edges deviating from each other within λ' are added to the same bucket.

fuzzyMatching counter is associated with each edge. While counting the support for a TAP *P,* in *CleanupPatternS*, we count the edges which are not in the same state as the TAP and their *fuzzyMatching* counter is positive. The counter is decremented by 1 for these edges. The edges could be either missing or could have assumed a different state during that phase. Specifically, while processing an edge, we find the TAP associated with the edge in the previous quantum. With each TAP, we keep a multi-set and the state of the edge is added to the multi-set in the current quantum. In *CleanupPatternS*, for each TAP, we extend it by the state with the highest count in its multi-set. For rest of the edges, the *FuzzyMatching* counter is decremented by 1. If the *fuzzyMatching* reaches 0 for an edge, the edge is dropped from the support set of that TAP. Thus, the entities associated with the TAP deviate from each other at most by λ' states.

Hence, we discover a set of TAPs where the entities associated with a TAP are deviating from each other by λ'. We next present our methodology to find similar TAPs by merging the TAPs, among the TAPs discovered based on *edge level* deviation, which are within $\lambda - \lambda'$ deviation from each other.

6.1.2 Pattern Level Deviation

The deviation between two TAPs is the same as the edit distance between two strings constructed over a given alphabet. For clarity, $(\lambda - \lambda')$ is denoted by just λ in the rest of the discussion.

Let **P** represent the set of all the *unique* TAPs discovered on a given EET log (as explained in Section 6.2, recurring TAPs are not reported as separate TAPs). Set **P** is divided into subsets $\mathbf{P}_i \subseteq \mathbf{P}$, such that $\forall P^i_m, P^i_n \in \mathbf{P}_i; d(P^i_m, P^i_n) \leq \lambda$ where $d(P^i{}_m, P^i{}_n)$ is the edit distance between two TAPs $P^i{}_m$ and $P^i{}_n$ belonging to subset $\mathbf{P}_i$. We call it the pattern partitioning problem, defined as follows:

Def. 6.1.2.1: For a given set of TAPs **P**, **P** is divided into K disjoint subsets $\mathbf{P}_i \subseteq \mathbf{P}$; $1 \leq i \leq K$, such that $\bigcup \mathbf{P}_i = \mathbf{P}$; $\forall i, j; \mathbf{P}_i \cap \mathbf{P}_{j(\neq i)} = \phi$ and $\forall P_m, P_n \in \mathbf{P}_i; d(P_m, P_n) \leq \lambda$.

The patterns within a subset $\mathbf{P}_i$ are merged together as they are all within λ deviation of each other. The Minimum Pattern Partitioning (MPP) problem is to "divide the set **P** into the smallest such subsets", i.e., K is minimized.

Theorem 2: The Minimum Pattern Partitioning Problem is NP-complete.

Proof: 'Minimum Clique Partitioning' problem (MCP) [17] is defined as follows: Given a graph $G(V, E)$, a minimum clique partition of G is a partition of V into minimum disjoint subsets $C_1, C_2 \ldots C_K$ such that the sub graph induced by $C_i, 1 \leq i \leq K$ is a clique.

Given a partition of $G(V, E)$, it is easy to verify that there are at most K subsets of V, each node belongs to exactly one subset of V and the union of the subsets is set V.

We now show that MCP is polynomial time reducible to MPP, $MCP \leq_p MPP$. For a given set of TAP strings **P**, we induce a graph $G(V, E)$ on **P** as follows: Each TAP $P_i \in \mathbf{P}$ is a node $v_i \in V_G$. For a pair of nodes $v_i, v_j \in V_G$ edge $e(v_i, v_j) \in E_G$ if $d(P_i, P_j) \leq \lambda$. P_i is the TAP pattern associated with node v_i.

Naturally, for a clique C_i in induced graph $G(V, E)$, all nodes $v_j \in C_i$, corresponding TAP patterns will be within λ distance of each other. Therefore, in order to obtain a smallest partitioning of S, we need to partition the graph $G(V, E)$ into the smallest number of *cliques* such that each node is part of exactly one clique in graph G. Therefore MPP is true *iff* MCP is true. □

MCP has the same complexity as Minimum Graph Coloring [17] problem. Greedy algorithm for MCP problem for a graph $G(V, E)$ is $O(\Delta).OPT$ where OPT is optimal number of colors and Δ is the maximum degree of a node in V (greedy algorithm uses Δ+1 colors

[17]). We now present a greedy algorithm for Minimum Pattern Partitioning (MPP) problem with the same complexity.

Algorithm: mpp_Greedy

Input: 1. Set **P** of TAP patterns. $|\mathbf{P}| = n$ and $P_i \in \mathbf{P}$ is a TAP
2. Deviation threshold λ.

Output: A partition of **P** into disjoint subsets such that patterns within a partition are within λ edit distance of each other.

1. Induce a graph $G(V, E)$ on **P**.
   i) $\forall \; P_i \in \mathbf{P}, v_i \in V$; **ii)** $\forall (P_i, P_j) \in \mathbf{P}$, if $d(P_i, P_j) \leq \lambda$, $e(v_i, v_j) \in E$
2. $l \leftarrow 0$; $S \leftarrow V$;
   while $(S \neq \Phi)$
   l++;
   n ← getNode (S); $S \leftarrow S - \{n\}$;
   Initialize clique C_l; $C_l \leftarrow C_l \bigcup n$;
   x ← FindCommonNeighbor (C_l, G);
   if x $\in S$
   $S \leftarrow S - \{x\}$; $C_l \leftarrow C_l \bigcup \{x\}$;
   else /*x belongs to an existing clique*/
   $C_m \leftarrow$ x.clusterId;
   if $(|C_l| + 1 > |C_m|)$
   $C_l \leftarrow C_l \bigcup \{x\}$; $C_m \leftarrow C_m - \{x\}$;
3. FindCommonNeighbor (Clique C)
   In graph G, returns a node which is common neighbor to all the nodes in clique C and is not already a part of clique C.
4. For each discovered clique C_i
   i) Create the corresponding subset $\mathbf{P}_i \in \mathbf{P}$. Set $\mathbf{P}_s \leftarrow \mathbf{\Phi}$
   ii) The pattern $P_m \in \mathbf{P}_i$ with highest support in subset $\mathbf{P}_i$ is the representative pattern for the subset.
   iii) $\mathbf{P}_s \leftarrow \mathbf{P}_s \bigcup P_m$

Thus set **P** is reduced to set $\mathbf{P}_s$, that represents similar TAPs.

Many a times, the sequence of states as well as length for a group of TAPs is exactly same but they do not start, hence do not end, in the same quanta. Such TAPs are called recurring TAPs. In next present our methodology to discover recurring TAPs.

6.2 Discovering Recurring TAPs

Recurring TAPs expose the lasting insights in the data. A TAP may have more significance for some applications if TAP itself is *recurring*. Naturally, two instances of a recurring TAP are separated by at least one quantum (otherwise there would not be two instances of that TAP in the first place).

In order to identity recurring patterns, we exploit the *pattern tree* in Figure 5. Pattern tree is maintained as trie structure for reverse TAPs. This structure can be exploited to maintain the temporal information about the TAPs efficiently as follows: Each leaf node in the original trie structure (Figure 5(a)) is marked with a special character. We replace it with an expandable array, *qArray,* where each entry in the array notifies the starting *quantum number* for an instance of the corresponding TAP when it occurs (Figure 5(b)). If any node in the *pattern tree* has no such associated *qArray,* it implies that no reverse TAP ends at that node. For a pattern, the *qArray* of the node corresponding to the last state in the reverse TAP is updated with the *quantum* number in which the TAP started. In Figure 5(b), we show the *pattern tree* for 3 patterns P_1 ($S_1S_2S_3$), P_2 ($S_1S_2S_2S_3$) and P_4 ($S_1S_1S_2S_3$). Let m=40. We see that P_2 *and* P_3 are recurring patterns. The *pattern tree* is updated as soon as a pattern is discovered. By looking at the time distributions users can identify recurring TAPs of their interest.

7. EXPERIMENTAL RESULTS

We have experimented with synthetic as well as real data. Our real data is the sales data of a pharmaceutical company. The goal of the experiments over real data is to expose interesting insights in the sales data provided with the help of TAPs. The goal of the experiments over synthetic data is to test the performance of our algorithm under different parameter values. The experiments over synthetic data demonstrate the scalability of our approach by confirming our analysis (see Appendix A). Experiments were carried out on Core2 Duo 2.4GHz, 4GB RAM Windows 8 machine. Java is used as programming language.

7.1 Insight gained with Real Data

We analyze the sales data of a big pharmaceutical company (described in Example 2). The data is anonymized and transaction time stamps are replaced with sequence numbers. We analyzed the transaction amount involved in different transactions and defined a mapping function **ψ** as follows: **ψ** puts a transaction into one of the four states (i.e., s=4) based on the amount of transaction ($\$t_a$) namely S_1, i.e., *RED* if $t_a \leq 200$), S_2, i.e, *YELLOW* if $200 < t_a \leq 2000$, S_3, i.e., *BLUE* if $2000 < t_a \leq 20000$, and S_4, i.e., *GREEN* if $t_a \geq 20000$. Naturally, one can change **ψ** to get different TAPs. We do not change **ψ** in our experiments. Hence, there are three parameters to vary, a) quantum length τ. We define τ based on number of transactions in a single quantum, b) minimum pattern support α and c) minimum pattern length β. No projection function is specified. Individual TAPs exposed interesting insights (as in Example 2). Here we discuss the insights gained due to their collective behavior.

Some of the trends observed had no surprises: For example with decreasing α, the total number of TAPs increases. But, there were others which were instructive. On studying the discovered TAPs, we realized that based of their markedly distinct characteristics, TAPs can be divided into two classes:

1) Monochromatic TAP (mTAPs): mTAPs, involving a single state signify a relatively steady relation between the respective entities. We call them monochromatic since the sequence can be represented by just one color; and **2) non-Monochromatic TAP (nmTAPs):** nmTAPs, involving multiple states signifying that the relationship (i.e., transaction amount) between the associated entities is varying significantly. They are called non-monochromatic as the sequence is represented by multiple colors.

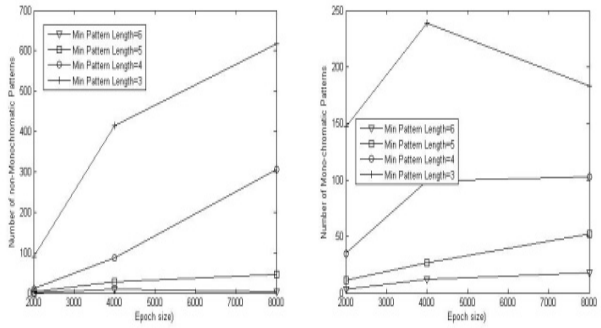


Figure 7: Variation in the (a) non-monochromatic and (b) monochromatic patterns

We have plotted the number of mTAPs and nmTAPs in Figure 7 for varying β and quantum size τ. Support α for these patterns is kept fixed at 4 for all the runs. As we see, the real data contains a large number of TAPs, verifying our premise.

Observe that, the number of nmTAPs increases at a faster rate with increasing quantum length compared to mTAPs (Figure 7), signifying that mTAPs are more steady. Hence, their overall number remains low. On the other hand, nmTAPs keep getting formed and dissolved. Figure 8 and Figure 9 depict the distribution of the TAPs of different lengths with varying quantum sizes.

Another interesting insight is that the *average support for mTAPs was more than twice that of the average support for nmTAPs*.

Further, a significant number of mTAPs are of length 5 or longer, whereas the number of nmTAPs is close to zero with β=6 (Figure 7). Hence, the number of nmTAPs is high but support and length of such TAPs is low. Hence, we infer that *if the relationship is not steady, it may not last long*. One very interesting insight is that *for nmTAPs relationship between associated entities was unsteady even if the transaction amount between them is increasing*. To understand this phenomena, we sought an answer to the question:

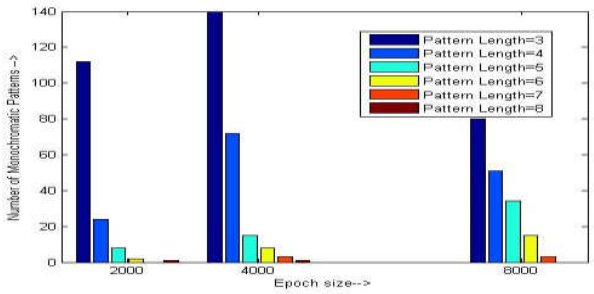


Figure 8: Distribution of monochromatic TAPs

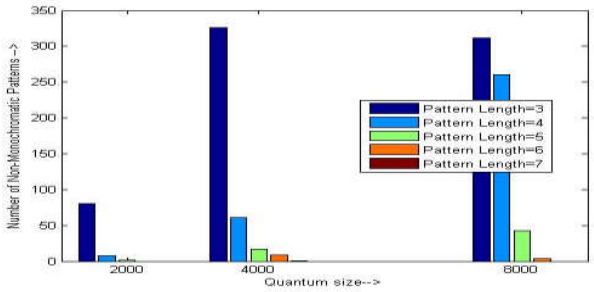


Figure 9: Distribution of non-monochromatic TAPs

Is there any correlation between different TAPs and products? We sorted the TAPs based on their support (τ=4000, α=4, β=3). For the mTAPs, the sales agents involved in first 80% of the TAPs were selling a subset of 67 products. For nmTAPs, 95 products were being sold for first 80% TAPs. Naturally, *in case of nmTAPs, a few drugs were initially bought by the respective accounts but later on stopped.* By studying the set of drugs involved in such nmTAPs, actionable insights can be found. For example, by analyzing which drugs are not being bought consistently, the pharmaceutical company can work on fixing the issues involved. Therefore, TAPs help improve the effectiveness of the sales. Further, TAPs expose that the sales team must focus more on the unsteady relationships. In our sales data, 142 out of 400 products were having $\leq$ 3 transactions (hence these products were not part of any TAP).

As we increase the quantum size, the number of TAPs increases, as shown in Table 2. We also see that a large number of TAPs are recurring. For quantum size 8000 transactions, we see, 88 TAPs are recurring and 142 TAPs occur only once (α=4, β=3). For quantum size 8000, maximum frequency of a recurring TAPs was as high as 53 and the mean frequency of recurring TAPs was close to 7.

Table 2: Recurring TAPs

Quantum Size →	2000	4000	8000
#Unique TAPs	42	107	142
#Recurring TAPs	30	51	88

Spatial Deviation: To study the spatial deviation, we varied the deviation threshold λ from 2 to 4. Figure 10 presents the number of TAPs after merging, on the sales data. β=5 for the TAPs. A number of TAPs merge with each other. However, since a large fraction of these TAPs were mTAPs, the impact of increasing λ was limited (mTAPs with length β will never merge if λ<β). For the sales data, interesting insights can be discovered, for example, by studying how the distribution of different products varies across merged TAPs for different values of λ.

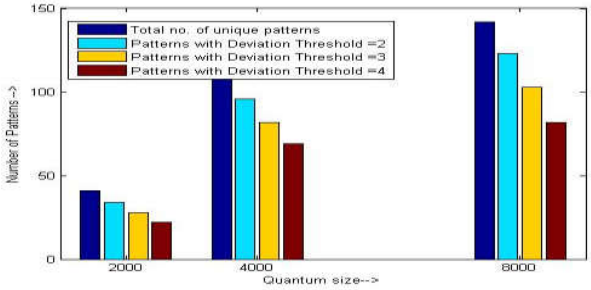


Figure 10: Patterns after Merging (Spatial Deviation)

7.2 Efficiency and Scalability Studies with Synthetic Data

We generate synthetic data with two distinct classes of entities, users and items. Data is generated such that a subset of users shows a *temporal* correlation with a subset of items. Thus, we generate the synthetic data as follows: First a distinct number of subsets are created for both users and items. The size of a subset is chosen from a uniform random distribution with mean μ_N and μ_M for users and items respectively. In our experiments, we create $|Ns|$ and $|Ms|$ distinct subsets of user sets and item sets respectively. Since our objective is to create a transaction workload where users and items show a correlation, we create a link structure between users and item subsets. A subset each of type *user* and type *item* is picked uniformly randomly and linked with the other to create an association between a subset of users and items. We create L_m such linkages where $L_m = max(|N_s|, |M_s|)$. Since this association needs to be spread over multiple *quanta,* a pattern length (n) corresponding to this association is picked from a Poisson distribution with mean μ. The pattern is then spread over those consecutive *quanta*, starting from the current quantum i to $i+n$. Further, in each quantum, a certain fraction of transactions do not get associated with any TAP mimicking real world scenario.

In Figure 11, we show the impact of support threshold on the processing time. For a TAP P, its support is defined as min ($|P.\mathbf{u}|$, $|P.\mathbf{v}|$), $|P.\mathbf{u}|$, $|P.\mathbf{v}|$ be the number of users and items associated with P. We vary the μ_N and μ_M to vary $|P.\mathbf{u}|$, $|P.\mathbf{v}|$. We plot the time taken of the TAP algorithm w.r.t. size of max (**u, v**). We see in Figure 11, *processing time is independent of the support threshold for the TAPs*. As shown in Appendix A, the processing time does not depend on the size of user sets or item sets. It depends just on the number of records in the data log and maximum number of edges in an instance of the bipartite graph sequence.

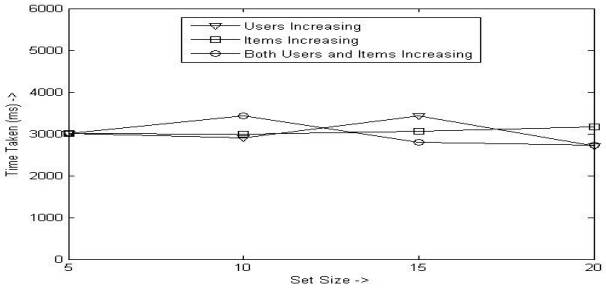


Figure 11. Processing Time Vs. TAP Support

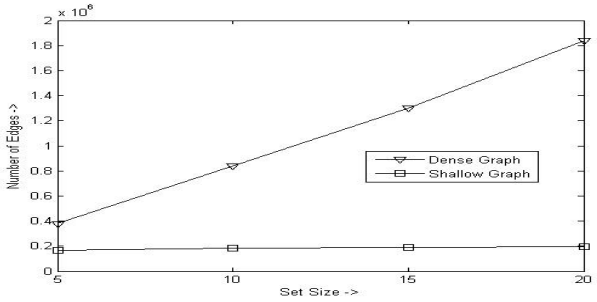


Figure 12. Pattern Support Size Vs. Edges in the Log

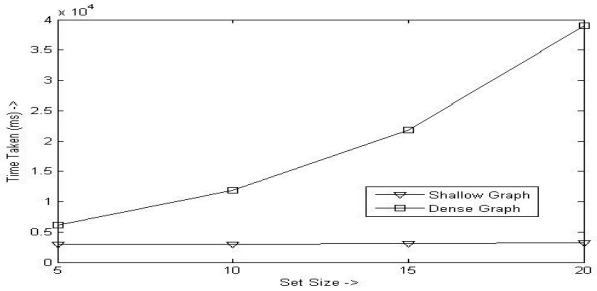


Figure 13. Pattern Support Size Vs. Processing Time

Next we examine the *impact of the size of the input data log*. We prepared two types of logs: i) Shallow logs with one-to-one relationship between a user and item ii) Dense logs with many-to-many relationship between users and items. For log of type (ii), each item in a transaction has an edge with each user in the transaction. Therefore, the density of edges in log of type (ii) is significantly higher. We fix the number of users and items in each quantum and vary the average size of the transactions in each type of logs. As we can see, the number of edges will not vary in the Shallow log, once the number of transactions in a quantum is fixed since the relationship between users and items is one-to-one, as shown in Figure 12 and 13. However, the number of edges increases linearly with average transaction size for Dense logs.

In Figure 13, the average processing time for Dense and Shallow graphs varies in accordance with our analysis (cf. Appendix A).

The *results validate our bound that the time taken to process the graph increases in $O(E.\log(E_{max}))$ where E is the total number of edges in the bipartite graph sequence and E_{max} is the maximum number of edges in a instance of the sequence.*

8. CONCLUSION

In this paper, we presented a new class of patterns called *temporal association patterns,* for discovering temporal association between entities belonging to two distinct classes. We mapped the problem to discovering patterns over a sequence of bipartite graphs and presented a novel technique to identify TAPs. Our technique is generic and applicable on a wide class of data logs. Our experimental results showed that our technique is scalable. We applied our technique on real data and exposed many interesting trends hidden in the data. One of our future research directions is to study TAPs among entities belonging to more than two classes.

9. REFERENCES

[1] Laxman, S., Shastri P.S,.: A Survey of Temporal Data Mining, *Sadhna,* Vol. 31, Part 2, April 2006.

[2] Yan X, Han J, Afshar R: CloSpan: Mining closed sequential patterns in large datasets. In: In SDM, 2003, San Fransisco, CA.

[3] Wang J., Han J.: BIDE: Efficient mining of frequent closed sequences. In ICDE, 2004, Boston, MA.

[4] Pasquier N., et al.: Discovering Frequent Closed Itemsets for Association Rules. In ICDT 1999, Jerusalem, Israel.

[5] Laxman S, Sastry P S, Unnikrishnan K P.: Discovering frequent episodes and learning hidden markov models: A formal connection. IEEE TKDE. Vol 17: 1505–1517, 2005.

[6] Atallah M J, Gwadera R, Szpankowski W.: Detection of significant sets of episodes in event sequences. In ICDM, 2004, Brighton, UK.

[7] Casas-Garriga G.: Discovering unbounded episodes in sequential data. In 7th PKDD, Croatia, 2003.

[8] Cao H, Cheung D W, Mamoulis N.: Discovering partial periodic patterns in discrete data sequences. In: 8th PAKDD, Sydney, 2004.

[9] Meger N, Rigotti C.: Constraint-based mining of episode rules and optimal window sizes. In: 8th PKDD, 2004, Pisa, Italy

[10] Lee C., Lin C. , Chen M. : On Mining General Temporal Association Rules in a Publication Database,. In: ICDM 2001.

[11] Bettini C., Wang X. S., Jajodia, S.: Testing Complex Temporal Relationships Involving Multiple Granularities and Its Application to Data Mining. In: PODS, 1996.

[12] Cormode G, Srivastava D, Yu Ting, Zhang Q.: Anonymizing bipartite graph data using safe groupings, VLDB Journal (2010) 19:115–139

[13] Agarwal R., Srirant R.: Fast Algorithms for Mining Association Rules. In: 20th VLDB, 1994, pp 487–499.

[14] Jian Pei, Jiawei Han, Mao R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In KDD, Boston, USA, 2000

[15] Zaki M J.: Efficient enumeration of frequent sequences. In: 7th CIKM 1998.

[16] V. Vazirani, Approximation Algorithms. Springer-Verlag, Berlin, 2001. http://www.nada.kth.se/~viggo/wwwcompendium/

[17] http://www.nada.kth.se/~viggo/problemlist/compendium.html

[18] https://en.wikipedia.org/wiki/Edit_distance

[19] Huang Y., et al.: Telco Churn Prediction with Big Data, in SIGMOD 2015.

[20] H. Tong, et al.: Proximity Tracking in time-evolving Bipartite Graphs, in SDM 2008

[21] J. Sun, D. Tao, C. Faloutsos: Beyond Streams and Graphs: Dynamic Tensor Analysis, in KDD 2006.

10. Appendix A: TAP Algorithm Analysis

10.1 Space Complexity Analysis

The maximum number of patterns for a given phase could be at most s^m ($\geq E_{max}/\alpha$). However, pattern may start in different phases. Therefore the maximum number of distinct pattern at any stage is $s + s^2 + \ldots s^m$. Hence, total memory footprint for storing patterns is

$= s + s^2 + \ldots s^m$

$< E_{max}/\alpha \times (s + 1 + 1/s + \ldots 1/s^{m-2})$ (since $s^{m-1} < E_{max}/\alpha$)

$< E_{max}/\alpha \times (s + 1 + 1/s + 1/s^2 + \ldots 1/s^{m-2} + \ldots\ldots)$

$< E_{max}/\alpha \times s + E_{max}/\alpha \times (1/s + 1/s^2 + \ldots 1/s^{m-2} + \ldots\ldots)$

$< E_{max}/\alpha \times (s^2/s\text{-}1)$

$O(s \times E_{max}/\alpha)$

If s/α is small, the memory footprint is $O(E_{max})$. Further, it is clear from the description of our algorithm (Figure 4) that we keep at most one quantum of graph data in memory. Besides, we keep a bounded number of patterns in the memory, as analyzed above. Therefore, the total memory foot print of our algorithm is $O(E_{max}) + O(E_{max}) = O(E_{max})$.

10.2 Time Compexity Analysis

In the algorithm in Section 5.1, , in phase 1 an edge is processed only once (put in one of the s states). In phase 2, an edge is processed twice, put in 2 different buckets, so on and so forth.

- Therefore, we see that each edge is processed at most 'm' times where m is such that $s^{m-1} < E_{max}/\alpha$ (or $s^m > E_{max}/\alpha$). Therefore, $m < \log E_{max}$
- Since the total number of edges in m instances of Graph G is bounded by $m.E_{max}$. Therefore, total running time of the algorithm is $O(m.E_{max}.\log E_{max})$.