Algorithms for Mining the Evolution of Conserved Relational States in Dynamic Networks

Rezwan Ahmed, George Karypis
Department of Computer Science & Engineering
University of Minnesota
Minneapolis, MN 55455
Email: {ahmed,karypis}@cs.umn.edu

***Abstract*—Dynamic networks have recently being recognized as a powerful abstraction to model and represent the temporal changes and dynamic aspects of the data underlying many complex systems. Significant insights regarding the stable relational patterns among the entities can be gained by analyzing temporal evolution of the complex entity relations. This can help identify the transitions from one conserved state to the next and may provide evidence to the existence of external factors that are responsible for changing the stable relational patterns in these networks. This paper presents a new data mining method that analyzes the time-persistent relations or states between the entities of the dynamic networks and captures all maximal non-redundant evolution paths of the stable relational states. Experimental results based on multiple datasets from real world applications show that the method is efficient and scalable.**

***Index Terms*—Dynamic network; relational state; evolution;**

I. Introduction

As the capacity to exchange and store information has soared, so has the amount and diversity of available data. To represent the relations between various entities in diverse applications and to capture the temporal changes and dynamic aspects of the underlying data, dynamic networks have been used as generic model due to its flexibility and availability of theoretical and applied tools for efficient analysis. Examples of some widely studied networks includes the friend-networks of popular social networking sites like Facebook, the Enron email network, co-authorship and citation networks, and protein-protein interaction networks. Analysis of temporal aspects of the entity relations in these networks can provide significant insight about the conserved relational patterns and their evolution over time.

Considerable effort has been made towards the development of efficient methods to analyze and extract useful information from static networks [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. On the other hand, the importance of dynamic networks has been recognized only recently; thus, the set of methods that are currently available for their analysis is considerably less developed than for static networks. Nevertheless, in recent years a considerable body of research has emerged on methods for finding patterns [17], clustering [18], characterizing network evolution rules [19], detecting related cliques [20], [21], and finding subgraph subsequences [22] in dynamic networks. Although the existing techniques can detect the frequent patterns in a dynamic network or track related patterns over time, they are not designed to identify stable relational patterns and do not focus on tracking the changes of these conserved relational patterns over time.

Our contribution in this paper is two folds. Firstly, we introduce a new class of patterns referred as the evolving induced relational states that are designed to analyze the time-persistent relations or states between the entities of the dynamic networks. Secondly, we present an algorithm to efficiently mine all maximal non-redundant evolution paths of the stable relational states of a dynamic network. We experimentally evaluate our algorithm using three real world datasets. First, we evaluate the performance and scalability of the algorithm on a large patent citation network by varying different input parameters. Second, we investigate some discovered evolving induced relational states from a trade network, an email communication network, and a patent citation network and provide a qualitative analysis of the information captured in them.

II. Definitions and Notation

A *graph* $G = (V, E, L[E])$ is composed of a set of nodes V modeling the entities of the network, a set of edges E modeling the relations between these entities, and a set of edge labels $L[E]$ modeling the type of the relations ($|E| = |L[E]|$). The relations between entities can either have a direction or not, leading to directed or undirected edges. An *induced subgraph* $G' = (V', E', L[E'])$ of $G = (V, E, L[E])$ is a graph such that $V' \subseteq V$, $E' \subseteq E$ and $\forall(u, v) \in E$ such that $v \in V'$ and $u \in V'$, $(u, v) \in E'$. For the rest of the discussion, any references to an induced subgraph will assume it is a connected induced subgraph. A *dynamic network* $\mathcal{N} = \langle G_1, G_2, \ldots, G_T \rangle$ is modeled as a finite sequence of graphs, where each G_t is a graph describing the state of the system at a discrete time interval t. The term *snapshot* will be used to refer to each of the graphs in the sequence. Snapshots are assumed to contain the same set of nodes, which will also be referred to as the nodes of $\mathcal{N}$ and denoted by $V(\mathcal{N})$. When nodes appear or disappear over time, the set of nodes of each snapshot is the union of all the nodes over all snapshots. Also, the nodes across the different snapshots are numbered consistently, so that the ith node of G_k ($1 \leq k \leq T$) will always correspond to the same ith node of $\mathcal{N}$.

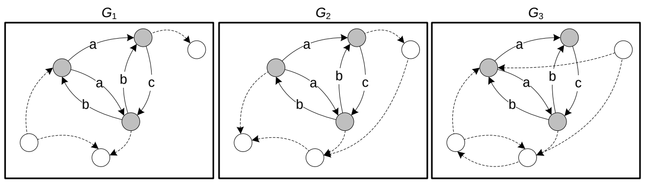


Fig. 1. Examples of relational states.

Due to its dynamic nature, the edges in $\mathcal{N}$ can over time appear, disappear, change label and/or direction. To capture the sequence of snapshots for which an edge exists in a consistent state, we define the *span sequence* of the edge as the sequence of maximal-length time intervals in which an edge is present in a consistent state. An edge (u, v) is in a *consistent state* over a maximal time interval $s\!:\!e$ if it is present in all snapshots $G_s, \ldots, G_e$ with the same label and direction and it is different in both G_{s-1} and G_{e+1} (assuming $s > 1$ and $e < T$). The span sequence of an edge will be described by a sequence of time intervals of the form $\langle s_1\!:\!e_1, \ldots, s_l\!:\!e_l\rangle$, where $s_i \leq e_i$ and $e_i \leq s_{i+1}$. In addition, we define the *span sequence* of a vertex as the sequence of maximal-length time intervals in which the vertex has at least one edge incident on it. The span sequence of the vertex will be represented as a sequence of time intervals in a similar fashion that of an edge.

A *persistent dynamic network* $\mathcal{N}^\phi$ is derived from a dynamic network $\mathcal{N}$ by removing all the edges from the snapshots of $\mathcal{N}$ that do not occur in a consistent state in at least ϕ consecutive snapshots, where $1 \leq \phi \leq T$. It can be seen that $\mathcal{N}^\phi$ can be derived from $\mathcal{N}$ by removing from the span sequence of each edge all the intervals whose length is less than ϕ.

An induced subgraph that involves the same set of vertices and whose edges and their labels remain conserved across a sequence of snapshots will be referred to as the *induced relational state* (IRS). The IRS definition is illustrated in Fig. 1. The three-vertex induced subgraph consisting of the dark shaded nodes that are connected via the directed edges corresponds to an induced relational state as it remains conserved in the three consecutive snapshots. The key attribute of an IRS is that the set of vertices and edges that compose the induced subgraph must remain the same in the consecutive snapshots. From this definition we see that an IRS corresponds to a time-conserved pattern of relations among a fixed set of entities (i.e., nodes) and as such can be thought as corresponding to a *stable* relational pattern. An IRS S_i will be denoted by the tuple $S_i = (V_i, s_i\!:\!e_i)$, where V_i is the set of vertices of the induced subgraph that persists from snapshot G_{s_i} to snapshot G_{e_i} ($s_i \leq e_i$) and it does not persist in G_{s_i-1} and G_{e_i+1} (assuming $s_i > 1$ and $E_i < T$). We will refer to the time interval $s_i\!:\!e_i$ as the *span* of S_i, and to the set of consecutive snapshots $G_{s_i}, \ldots, G_{e_i}$ as its *supporting set*. By its definition, the span of an IRS and the length of its supporting set are *maximal*. Note that for the rest of the paper, any references to subsequences of snapshots will be assumed to be consecutive. The induced subgraph corresponding to an IRS S_i will be denoted as $g(S_i)$. A snapshot G_t supports an induced relational state S_i, if $g(S_i)$ is an induced subgraph of G_t.

III. Evolving Induced Relational State (EIRS)

An example of the type of evolving patterns in dynamic networks that the work in this paper is designed to identify is illustrated in Fig. 2. This figure shows a hypothetical dynamic network consisting of 14 consecutive snapshots, each modeling the annual relations among a set of entities. The four consecutive snapshots for years 1990 through 1993 show an induced relational state S_1 that consists of nodes {a, b, e, f}. This state was evolved to the induced relational state S_2 that occurs in years 1995–1999 that contains nodes {a, b, d, e, h}. Finally, in years 2000–2003, the induced relational state S_2 was further evolved to the induced relational state S_3 that contains the same set of nodes but has a different set of relations. Note that even though the sets of nodes involved in S_1 and S_2 are different, there is a high degree of overlap between them. Moreover, the transition from S_1 to S_2 did not happen in consecutive years, but there was a one year gap, as the snapshot for 1994 does not contain either S_1 or S_2. Such a sequence of induced relational states $S_1 \rightsquigarrow S_2 \rightsquigarrow S_3$ represents an instance of what we refer to as an *evolving induced relational state* (EIRS) and represents the types of patterns that the work in this paper is designed to identify.

EIRSs identify entities whose relations transition through a sequence of time-persistent relational patterns and as such can provide evidence of the existence of external factors responsible for these relational changes. For example, consider a dynamic network that captures the trading patterns between a set of entities. The nodes in this *trading network* model the trading entities (e.g., countries, states, businesses, individuals) and the directed edges model the transfer of goods and their types from one entity to another. An EIRS in this trading network can potentially identify how the trading patterns change over time (e.g., addition/deletion of edges or inclusion of new trading partners) signalling the existence of significant economic, political, and environmental factors that drive such changes (see §VI-C for some examples of such patterns in a inter-country trading network). Similarly, in a dynamic network that captures the annual citation structure of U.S. Patents (or other scientific publications), EIRSs can identify how stable knowledge transfer or knowledge sharing sub-networks among different science areas have evolved and thus facilitate the identification of transformative science developments that changed the state of these sub-networks.

The formal definition of an EIRS is as follows.

Definition 1 (Evolving Induced Relational State) *Given a dynamic network $\mathcal{N}$ containing T snapshots, a value ϕ ($1 \leq \phi \leq T$), and a value β ($0 < \beta \leq 1$), an evolving induced relational state of length m is a sequence of induced relational states $\langle S_1, S_2, \ldots, S_m\rangle$ that satisfies the following constraints:*

(i) *the supporting set of each induced relational state S_i contains at least ϕ consecutive snapshots in the persistent dynamic network $\mathcal{N}^\phi$ of $\mathcal{N}$,*

(ii) *for each $1 \leq i < m$, the first snapshot in S_{i+1}'s supporting set follows the last snapshot in S_i's supporting set,*

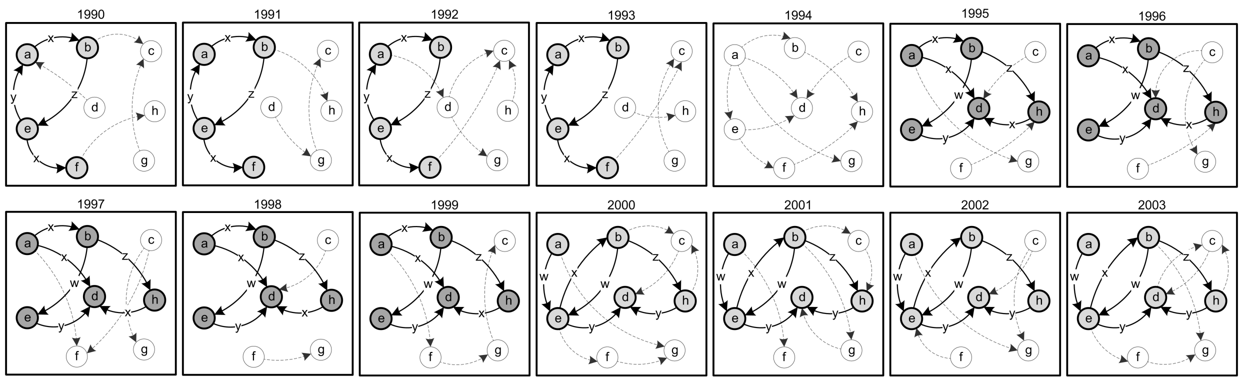


Fig. 2. An example of an evolving relational state.

(iii) *for each* $1 \leq i < m$, $g(S_i)$ *is different from* $g(S_{i+1})$, *and*

(iv) *for each* $1 \leq i < m$, $|V_i \cap V_{i+1}|/|V_i \cup V_{i+1}| \geq \beta$.

The value ϕ, referred to as the support of the EIRS, is used to capture the requirement that each induced relational state occurs in a sufficiently large number of consecutive snapshots and as such it represents a set of relations among the entities involved that are stable. The value β, referred to as the *inter-state similarity*, is used to enforce the minimum vertex-level similarity between the selected relational states. This ensures that the EIRS captures the relational transitions of a consistent set of vertices but at the same time allows for the inclusion of new vertices and/or the elimination of existing vertices, if they are required to describe the new relational state. The third constraint in the above definition is used to eliminate EIRSs that contain consecutive IRSs with identical induced subgraphs. This is motivated by our desire to find EIRSs that capture changes in the time-persistent relations. However, the above definition allows for the same induced subgraph to occur multiple times in the same EIRS, as long as these occurrences do not happen one after the other.

An important aspect of the definition of an EIRS is that it is defined with respect to the persistent dynamic network $\mathcal{N}^{\phi}$ of $\mathcal{N}$ and not $\mathcal{N}$ itself. This is because we are interested in finding how the persistent relations among a set of entities have changed over time and as such we first eliminate the set of relations that appear for a short period of time.

Given the above definition, the work in this paper is designed to develop efficient algorithms for solving the following problem:

Problem 1 (Maximal Evolving Induced Relational State Mining) *Given a dynamic network* $\mathcal{N}$ *containing* T *snapshots, a user defined support* ϕ *(*$1 \leq \phi \leq T$*), a inter-state similarity* β *(*$0 < \beta \leq 1$*), a minimum size of* $k_{\min}$ *and a maximum size of* $k_{\max}$ *vertices per IRS, and a minimum EIRS length* $m_{\min}$*, find all EIRSs such that no EIRS is a subsequence of another EIRS.*

Since the set of maximal EIRSs contains within all non-maximal EIRSs, the above problem will produce a succinct set of results. Also, the minimum and maximum constraints on the size of the IRSs involved is introduced to allow an application to focus on IRSs of meaningful size, whereas the minimum constraint on the EIRS length is introduced in order to eliminate short paths.

IV. Finding Evolving Induced Relational States

The algorithm that we developed for finding all maximal EIRSs (Problem 1) follows a two-step approach. In the first step, the dynamic network $\mathcal{N}$ is transformed into its persistent dynamic network $\mathcal{N}^{\phi}$ and a recursive enumeration algorithm is used to identify all the IRSs $\mathcal{S}$ whose supporting set is at least ϕ in $\mathcal{N}^{\phi}$. The $\mathcal{N}$ to $\mathcal{N}^{\phi}$ transformation is done by removing spans that are less than ϕ from each edge's span sequence and then removing the edges with empty span sequences. In the second step, the set of IRSs are mined in order to identify their maximal non-redundant sequences that satisfy the constraints of EIRS's definition.

A. Step 1: Mining of Induced Relational States

The algorithm that we developed to mine all induced relational states is based on a recursive approach to enumerate all (connected) induced subgraphs of a graph that satisfy minimum and maximum size constraints. In the rest of this section we first describe the recursive algorithm to enumerate all induced subgraphs in a simple graph and then describe how we modified this approach to mine the induced relational states in a dynamic network. The enumeration algorithms was inspired by the recursive algorithm to enumerate all spanning trees [23]. Our discussion initially assumes that the graph is undirected and the necessary modifications that apply for directed graphs are described afterwards. Also, any references to induced subgraphs assumes *connected* induced subgraphs.

1) Induced Subgraph Enumeration: Given a graph $G = (V, E, L[E])$, let $G_i = (V_i, E_i, L[E_i])$ be an induced subgraph of G (V_i can also be empty), V_f be a subset of vertices of V satisfying $V_i \cap V_f = \emptyset$, and let $F(V_i, V_f)$ be the set of induced subgraphs of G that contain V_i and zero or more vertices from V_f. Given these definitions, the complete set of induced subgraphs of G is given by $F(\emptyset, V) \setminus \emptyset$. The set

$F(V_i, V_f)$ can be computed using the recurrence relation.

$$F(V_i, V_f) = \begin{cases} V_i, & \text{if } \mathrm{sgadj}(V_i, V_f) = \emptyset \\ & \text{or } V_f = \emptyset \\ F(\{u\}, V_f \setminus u) \cup F(\emptyset, V_f \setminus u), & \text{if } V_i = \emptyset \wedge V_f \neq \emptyset \\ \text{where } u \in V_f & \\ F(V_i \cup \{u\}, V_f \setminus u) \cup F(V_i, V_f \setminus u), & \text{otherwise,} \\ \text{where } u \in \mathrm{sgadj}(V_i, V_f) & \end{cases} \tag{1}$$

where $\mathrm{sgadj}(V_i, V_f)$ (*subgraph-adjacent*) denotes the vertices in V_f that are adjacent to at least one of the vertices in V_i.

To show that Equation 1 correctly generates the complete set of induced subgraphs, is sufficient to consider the three conditions of the recurrence relation. The first condition, which corresponds to the initial condition of the recurrence relation, covers the situations in which either (i) none of the vertices in V_f are adjacent to any of the vertices in V_i and as such V_i is the only induced subgraphs that can be generated, or (ii) V_f is empty and as such V_i cannot be extended further. The second condition, which covers the situation in which V_i is empty and V_f is not empty, decomposes $F(\emptyset, V_f)$ as the union of two sets of induced subgraphs based on an arbitrarily selected vertex $u \in V_f$. The first is the set of induced subgraphs that contain vertex u (corresponding to $F(\{u\}, V_f \setminus u)$) and the second is the set of induced subgraphs that do not contain u (corresponding to $F(\emptyset, V_f \setminus u)$). Since any of the induced subgraphs in $F(\emptyset, V_f)$ will either contain u or not contain u, the above decomposition covers all possible cases and it correctly generates $F(\emptyset, V_f)$. Finally, the third condition, which corresponds to the general case, decomposes $F(V_i, V_f)$ as the union of two sets of induced subgraphs based on an arbitrarily selected vertex $u \in V_f$ that is adjacent to at least one vertex in V_i. The first is the set of induced subgraphs that contain V_i and u (corresponding to $F(V_i \cup \{u\}, V_f \setminus u)$) and the second is the set of induced subgraphs that contain V_i but not u (corresponding to $F(V_i, V_f \setminus u)$). Similarly to the second condition, this decomposition covers all the cases with respect to u and it correctly generates $F(V_i, V_f)$. Also the requirement that $u \in \mathrm{sgadj}(V_i, V_f)$ ensures that this condition enumerates only the connected induced subgraphs[*]. Since each recursive call in Equation 1 removes a vertex from V_f, the recurrence relation will terminate due to the first condition. Finally, since the three conditions in Equation 1 cover all possible cases, the overall recurrence relation is correct.

In addition to correctness, it can be seen that the recurrence relation of Equation 1 does not have any overlapping subproblems, and as such, each induced subgraph of $F(V_i, V_f)$ is generated only once, leading to an efficient approach for generating $F(V_i, V_f)$. Constraints on the minimum and maximum size of the induced subgraphs can be easily incorporated in Equation 1 by returning $\emptyset$ in the first condition when $|V_i|$ is less than the minimum size and not performing the recursive exploration for other two cases.

[*]Given a connected (induced) subgraph g, it can be grown by adding one vertex at a time while still maintaining connectivity; e.g., a MST of g (which exists due to its connectivity) can be used to guide the order by which vertices are added.

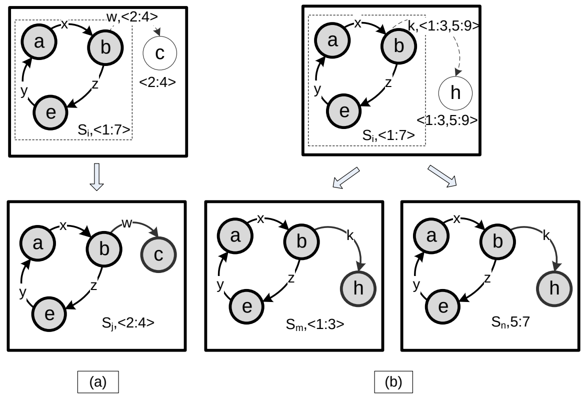


Fig. 3. Adding a vertex to an induced relational state.

2) Induced Relational State Enumeration: There are two key challenges in extending the induced subgraph enumeration approach of Equation 1 in order to enumerate the IRSs in a dynamic network. First, the addition of a vertex to an IRS is different from adding a vertex to an induced subgraph as it can can result in multiple IRSs depending on the overlapping spans between the vertex being added and the original IRS. Consider an IRS $S_i = (V_i, s_i\!:\!e_i)$, a set of vertices V_f such that $V_i \cap V_f = \emptyset$, and a vertex $v \in V_f$ that is adjacent to at least one of the vertices in V_i. If v's span sequence contains multiple spans that have overlaps greater than or equal to ϕ with S_i's span, then the inclusion of v leads to multiple IRSs, each supported by different disjoint spans. Fig. 3 illustrates the vertex addition process during an IRS expansion. Fig. 3(a) shows a simple case of vertex addition where an IRS $S_1 = (\{a, b, e\}, 1\!:\!7)$ is expanded by adding adjacent vertex c having a single span of $2\!:\!4$. The resultant IRS is $S_2 = (\{a, b, e, c\}, 2\!:\!4)$ that contains all the vertices and only the overlapping span of S_1 and c. Fig. 3(b) shows a more complex case where the vertex h that is added to S_1 has the span sequence of $\langle 1\!:\!3, 5\!:\!9 \rangle$. In this case, the overlapping spans $1\!:\!3$ and $5\!:\!7$ form two separate IRSs $S_3 = (\{a, b, e, h\}, 1\!:\!3)$ and $S_4 = (\{a, b, e, h\}, 5\!:\!7)$, each of which needs to be considered for future expansions in order to discover the complete set of IRSs.

Second, the concept of removing a vertex from V_f used in Equation 1 to decompose the set of induced subgraphs needs to be re-visited so that to account for the temporal nature of dynamic networks. Failure to do so, will lead to an IRS discovery algorithm that will not discover the complete set of IRSs and the set of IRSs that it discovers will be different based on the order that it chooses to add vertices in the IRS under consideration. This is illustrated in the example of Fig. 4. The IRS $S_1 = (\{a, b\}, 0\!:\!12)$ is expanded to $S_2 = (\{a, b, c\}, 1\!:\!5)$ by adding the adjacent vertex c. In terms of Equation 1, this corresponds to the third condition and leads to the recursive calls of $F(\{a, b, c\}, V_f \setminus c)$ and $F(\{a, b\}, V_f \setminus c)$ (i.e., expand S_2 and expand S_1). It is easy to see that the set of IRSs that will be generated from these recursions will not contain $S_4 = (\{a, b, c, d\}, 6\!:\!11)$, since it can only be generated from S_2 but its span does not overlap with S_4's

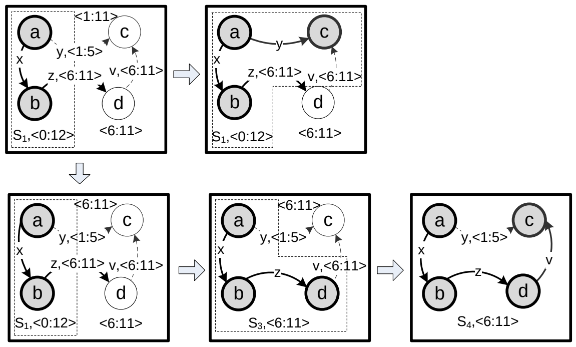


Fig. 4. Updating span sequence of a vertex.

span. One the other hand, if S_1 is initially expanded by adding vertex d, resulting in the IRS $S_3 = (\{a, b, d\}, 6\!:\!11)$, then the recursive calls of $F(\{a, b, d\}, V_f \setminus d)$ and $F(\{a, b\}, V_f \setminus d)$ will generate S_4 and S_2, respectively. Thus, based on the order by which vertices are selected and included in an IRS, some IRSs may be missed. Moreover, different vertex inclusion orders can potentially miss different IRSs.

To address both of these issues the algorithm that we developed for enumerating the complete set of IRSs utilizes a recursive decomposition approach that extends Equation 1 by utilizing two key concepts. The first is the notion of the set of vertex-span tuples that can be used to grow a given IRS. Formally, given an IRS $S_i = (V_i, s_i\!:\!e_i)$ and a set of vertices V_f in $\mathcal{N}^\phi$ with $V_i \cap V_f = \emptyset$, the $\text{irsadj}(S_i, V_f)$ is the set of vertex-span tuples of the form $(u, s_{u_j}\!:\!e_{u_j})$ such that $u \in V_f$ and $(V_i \cup \{u\}, s_{u_j}\!:\!e_{u_j})$ is an IRS whose span is at least ϕ. Note that $\text{irsadj}(S_i, V_f)$ can contain multiple tuples for the same vertex if that vertex can extend S_i in multiple ways (each having a different span and possibly an induced subgraph with different sets of edges). The tuples in $\text{irsadj}(S_i, V_f)$ represent possible extensions of S_i and since a vertex can occur multiple times, it allows for the generation of IRS with the same set of vertices but different spans (addressing the first challenge).

The second is the notion of vertex-span deletion, which is used to eliminate the order dependency described earlier and generate the complete set of IRSs. The key idea is when a tuple $(u, s_{u_j}\!:\!e_{u_j})$ is added into S_i, instead of removing u from V_f, only remove the span $s_{u_j}\!:\!e_{u_j}$ from u's span sequence in V_f. Vertex u will only be removed from V_f iff after the removal of $s_{u_j}\!:\!e_{u_j}$ its span sequence becomes empty or the remaining spans have lengths that are smaller than ϕ. Formally, given V_f, a vertex $u \in V_f$, and a vertex-span tuple $(u, s_{u_j}\!:\!e_{u_j})$, the *span-deletion* operation, denoted by $V_f|(u, s_{u_j}\!:\!e_{u_j})$, updates the span sequence of u by removing the $s_{u_j}\!:\!e_{u_j}$ span from its span sequence, eliminating any of its spans that become shorter than ϕ, and eliminating u if its updated span sequence becomes empty. The span-deletion operation is the analogous operation to vertex removal of Equation 1. To illustrate how this operation can address the second challenge, consider again the example of Fig. 4. Once c is added into S_1, the span that it used (i.e., $1\!:\!5$) will be deleted from its span sequence, resulting in a new span sequence containing $\langle 6\!:\!11\rangle$. Now the recursive call corresponding to $F(\{a, b\}, V_f|(c, 1\!:\!5))$ will be able to identify S_4.

Given the above definitions, the recursive approach for enumerating the complete set of IRSs can now be formally defined. Let $S_i = (V_i, s_i\!:\!e_i)$ be an IRS, V_f a set of vertices in $\mathcal{N}^\phi$ with their corresponding span-sequences in $\mathcal{N}^\phi$ such that $V_i \cap V_f = \emptyset$, and $H(S_i, V_f)$ be the set of IRSs that (i) contain V_i and zero or more vertices from V_f and (ii) their span is a sub-span[†] of $s_i\!:\!e_i$. Given the above, the complete set of IRSs in $\mathcal{N}^\phi$ is given by $H((\emptyset, 1\!:\!T), V(\mathcal{N}^\phi)) \setminus \emptyset$. The recurrence relation for $H(S_i, V_f)$ is:

$$H(S_i, V_f) =$$

$$\begin{cases} S_i, & \text{if } \text{irsadj}(S_i, V_f) = \emptyset \\ & \text{or } V_f = \emptyset \\ \\ H((\{u\}, s_u\!:\!e_u), V_f|(u, s_u\!:\!e_u)) & \\ \cup\, H((\emptyset, 1\!:\!T), V_f|(u, s_u\!:\!e_u)), & \text{if } V_i = \emptyset \wedge V_f \neq \emptyset \\ \text{where } u \in V_f \text{ and } s_u\!:\!e_u \text{ is a span of } u & \\ \\ H((V_i \cup \{u\}, s_u\!:\!e_u), V_f|(u, s_u\!:\!e_u)) & \\ \cup\, H(S_i, V_f|(u, s_u\!:\!e_u)), & \text{otherwise.} \\ \text{where } (u, s_u\!:\!e_u) \in \text{irsadj}(S_i, V_f) & \end{cases} \tag{2}$$

The above recurrence relation shares the same overall structure with the corresponding recurrence relation for enumerating the induced subgraphs (Equation 1) and its correctness can be shown in a way similar to that used for Equation 1. Due to space limitations, we omit the complete proof of Equation 2, and only focus on discussing the third condition, which represents the general case. This condition decomposes $H(S_i, V_f)$ as the union of two sets of IRSs based on an arbitrarily selected vertex-span tuple $(u, s_u\!:\!e_u) \in \text{irsadj}(S_i, V_f)$. Since the span $s_u\!:\!e_u$ is a maximal overlapping span between u and S_i, the new IRS containing vertices $(V_i \cup \{u\}$ has the span of $s_u\!:\!e_u)$. With respect to vertex-span tuple $(u, s_u\!:\!e_u)$, the set of IRSs in $H(S_i, V_f)$ can belong to one of the following three groups: (i) the set of IRSs that contain u and have a span that is a sub-span of $s_u\!:\!e_u$; (ii) the set of IRSs that contain u and have a span that is disjoint with $s_u\!:\!e_u$, and (iii) the set of IRSs that do not contain u. The $H((V_i \cup \{u\}, s_u\!:\!e_u), V_f|(u, s_u\!:\!e_u))$ part of the third condition generates (i), whereas the $H(S_i, V_f|(u, s_u\!:\!e_u)$ part generates (ii) and (iii). What is missing from the above groups is the group corresponding to the set of IRSs that contain u and have a span that partially overlaps with $s_u\!:\!e_u$. The claim is that this cannot happen. Consider an IRS $S_j = (V_j, s_j\!:\!e_j) \in H(S_i, V_f)$ that contain u and without loss of generality, assume that $s_u < s_j < e_u < e_j$. Since we are dealing with induced subgraphs and stable topologies (i.e., from the definition of an IRS), the connectivity of u to the vertices in V_i remains the same during the span of $s_u\!:\!e_u$ and also during the span of $s_j\!:\!e_j$, which means that the connectivity of u to the vertices in V_i remains the same during the entire span of $s_u\!:\!e_j$. This is a contradiction, since $(u, s_u\!:\!e_u) \in \text{irsadj}(S_i, V_f)$ and as such is a maximal length span of stable relations due to the fact that $(V_i \cup \{u\}, s_u\!:\!e_u)$

[†]The *sub-span* of a span corresponds to a time interval that is either identical to the span or is contained within it.

Algorithm 1 mdfs($u, t, d[], p$)

```
1: /* $u$ is the current node */
2: /* $t$ is the current time */
3: /* $d[]$ is the discovery array */
4: /* $p$ is the current path */
5: $d[u]$ = t++
6: push $u$ into $p$
7: if $adj(u) = \emptyset$ and $|p| >$ minimum EIRS-length then
8:     record $p$
9: else
10:    for each node $v$ in ($adj(u)$ sorted in increasing end-time order) do
11:        if $dt[v] < d[u]$ then
12:            mdfs($v, t, d, p$)
13: pop $p$
```

is an IRS and the span of an IRS is maximal. Thus, the two cases of the third condition in Equation 2 cover all possible cases and it correctly generate the complete set of IRSs.

Also, since each recursive call modifies at least the set V_f, none of the recursive calls lead to overlapping subproblems, ensuring that each IRS is only generated once.

3) Handling Directed Edges: To handle directed edges, we consider each direction of an edge separately, such that a directed edge $a \rightarrow b$ is listed separately from $a \leftarrow b$. The direction of an edge is stored as part of the label along with the span sequence of that edge. The direction of an edge at a certain span is coded as 0, 1 or 2 to represent $a \rightarrow b$, $a \leftarrow b$ or $a \leftrightarrow b$. Note that the ordering of the vertices in an edge (a, b) are stored in increasing vertex-number order (i.e, $a < b$). Using the above representation of an edge, we determine the direction of all the edges of an IRS during its span duration.

B. Step 2: Mining of Maximal Evolution Paths

The algorithm that we developed to identify the sequence of IRSs that correspond to the maximal EIRSs is based on a modified DFS traversal of a directed acyclic graph that is referred to as the *induced relational state graph* and will be denoted by G^{RS}. G^{RS} contains a node for each of the discovered IRSs and a root node r. For each pair of IRSs $S_i = (V_i, s_i\!:\!e_i)$ and $S_j = (V_j, s_j\!:\!e_j)$, G^{RS} contains a directed edge from the node corresponding to S_i to the node corresponding to S_j iff $V_i \neq V_j$, $|V_i \cap V_j|/|V_i \cup V_j| \geq \beta$ and $e_i < s_j$ (i.e., constraints (ii)–(iv) of Definition 1). The modified DFS algorithm which we will refer to as mdfs, uses only a discovery array $d[]$ to record the discovery times of each node, which are initially set to 0. The DFS traversal starts from the root node and proceeds to visit the rest of the nodes. The pseudocode is shown in Algorithm 1.

The mdfs algorithm also keeps track of the current path from the root to the node that it is currently at. If that node has no outgoing edges, then it outputs that path. The sequence of the relational states corresponding to the nodes of that path (the root node is excluded), represent an EIRS.

The key difference between mdfs and the standard DFS algorithm is the method for selecting which node to visit next (line 4). Given a node u, the mdfs algorithm selects among its adjacent nodes the node v that has the earliest end-time and $d[v] < d[u]$. The selection of the earliest end-time is done to allow mdfs to find maximal paths first, where the selection of v that satisfies $d[v] < d[u]$ is done to eliminate forward edges but still allow for the exploration of cross edges. The elimination of forward edges is done so that to eliminate paths that are sub-paths of previously identified longer paths (and as such will lead to non-maximal EIRSs). The inclusion of cross-edges is done so that to allow the mdfs algorithm to find the complete set of maximal EIRSs. Due to space constraints, we do not provide the formal proof that the set of paths generated by mdfs($r, 0, d, \emptyset$) represent the complete set of maximal EIRSs that we are after. However, we hope that the discussion above provides evidence as to the correctness of the claim.

The discussion so far assumes that G^{RS} has been fully materialized. However, this can be expensive as it requires pairwise comparisons between a large number of IRSs in order to determine if they satisfy the constraints (ii)–(iv) of Definition 1. For this reason, our algorithm starts with the mdfs traversal from the root and materializes the portions of G^{RS} that it needs during the traversal. This allows it to reduce the rather expensive computations associated with some of the constraints by not having to visit forward edges. Moreover it utilizes the minimum EIRS length constraint to further prune the parts of G^{RS} that it needs to generate.

For a given node u, the mdfs algorithm needs the adjacent node of u that has the earliest end-time. Let e_u be the end-time of node u. Since a node (i.e., an IRS) is required to have at least a span of length ϕ, we start u's adjacent node search among the nodes in G^{RS} that have the end-time of $e_k = e_u + \phi$. According to constraint (iv) of Definition 1, a certain minimum threshold of similarity is desired between two IRSs of an EIRS. Thus, it is sufficient to compare u with only those nodes that have at least a common vertex with u. We index the nodes of G^{RS} based on the vertices so that the similar nodes of u can be accessed by looking up all the nodes that have at least one vertex in common with u. If a node v is similar to u and $d[v] < d[u]$, we add the node to the adjacency list. If the search fails to detect any adjacent node, we initiate another search looking for nodes that have an end-time of $e_k + 1$ and continue such incremental search until an adjacent node of u is found or all possible end-times have been explored.

V. Related Work

Over the last ten years, considerable research effort has been devoted to developing algorithms to find patterns in static graphs and networks. This research has resulted in the development of algorithms for finding different types of patterns such as paths [24], [25], trees [6], [7], induced subgraphs [8], arbitrarily connected subgraphs [26], [10], and various types of cliques [27], [28], [29]. While most of these methods have been developed for mining databases containing relatively small graphs, algorithms have also been developed to identify subgraphs with a large number of embeddings in a single large graph (i.e., static network) [11].

Since dynamic networks have only recently emerged as an important research area, the problem of formally defining the conserved relational states and developing algorithms for identifying their evolution patterns have not been well studied. Desikan et. al. [30] analyzed the importance of mining temporally evolving Web graphs, but did not provide any algorithmic solution for detecting the stable patterns or their evolution. Borgwardt et. al. [17], [31] introduced the notion of the *dynamic subgraph*, which extends the traditional notion of the subgraph to include the sequence of subgraphs that exist in a consecutive sequence of snapshots and developed algorithms to identify the set of dynamic subgraphs that occur frequently in a dynamic network. Even though this work provides a similar definition to the stable relational states, it uses a heuristic based approach to detect the frequent patterns and it does not capture the evolution of those patterns over time. Jin et. al. [32] focused on the problem of finding recurrent patterns in dynamic networks in which a time series is associated with each node and introduced the notion of the *trend motif*, which is a connected subgraph in which each node's time series exhibits a consistent trend over a time interval. Even though the trend motif formulation is general, their work focused only on developing algorithms for finding frequently occurring trend motifs that show either an increasing or a decreasing trend. Berlingerio et. al. [19] also provided an algorithm to detect frequent subgraphs in time-evolving graphs for deriving graph-evolution rules that satisfy a minimum confidence constraint. Although these evolution rules are used to characterize the changes in the overall evolving graph, it does not determine the evolution of the conserved frequent patterns. Robardet [21] represented the frequent patterns of a graph as *pseudo-cliques* and proposed an algorithm that first mines each graph snapshot of a dynamic graph for local patterns and then combines these with patterns from previous snapshot based on some constraints to form evolving patterns. Inokuchi et. al. [33], [22] solved a similar problem of finding frequent induced subgraph subsequences from graph sequence data and capturing the changes of a subgraph over the subsequence. However, their approach does not focus on determining stable induced subgraphs and does not discover the evolution patterns of the conserved relational states.

A related body of research has investigated the task of identifying and tracking patterns in biological networks [34], [35], [31] and evolving communities in social networks [36], [37]. The problem of evolutionary clustering [18], [38] is to find clusters in a dynamic network in the form of snapshots, such that clusters at any given time groups similar entities while also preserving the grouping of entities in past snapshots. Even though these methods provide valuable insights on the evolution of dynamic networks, the nature of conserved patterns and their evolution focused in this paper is different from the general evolution addressed in other papers.

VI. Experimental Design & Results

A. Datasets

We evaluated our algorithm using datasets from a patent citation network, a trade network and an email communication network. The scalability of our algorithm was assessed on the patent citation dataset (as it was the largest) whereas all three datasets were used in the qualitative assessment of the identified EIRSs.

TABLE I
Dynamic Network Datasets.

Dataset	#Vertices	Avg. #Edeges	Span
Patent N2	84152	45465	34
Patent N3	84152	63633	34
Patent N4	84152	79358	34
Trade	192	23	53
Enron	130	88	30

#Vertices denotes the total number of vertices in the dynamic network. Avg. #Edeges denotes the average number of edges per snapshot in the dynamic network. Span denotes the total number of snapshots in the dynamic network.

a) Patents Citation Network: This is a citation network derived from the United States Patent and Trademark office's (USPTO) bibliographic information for the patents granted from 1976 to 2009. The nodes correspond to the primary art areas associated with the patents based on USPTO's classification and the edges correspond to aggregated citations between art areas. For example, if patent A of art area α cites patent B of art area β in year x, then a directed edge is added from α to β in the graph representing year x. Citations that produce self references are removed. The classification of USPTO art areas is hierarchical and forms a tree structure that can be up to 16 levels deep. The patents are assigned the classes corresponding to the leaf nodes of the tree. In our experiments, in order to obtain art areas that contain a sufficiently large number of patents, we rolled up the classification tree to the third level, and all the patents underneath each third-level node was assigned the class of that node. The snapshots of the dynamic network that we created correspond to the citation network of each year, leading to a dynamic network consisting of $2009 - 1976 = 34$ snapshots. Since the vertices at each snapshot can potentially be connected to all other vertices, we pre-processed each snapshot in order to derive a set of dynamic networks that contain the most important set of outgoing edges (i.e. references) from each node. This is done as follows. For each vertex of each snapshot, we first choose the 20 most frequent edges. The frequency of an edge (a, b) is defined as the number of patent-to-patent references (P_1, P_2) such that P_1's class is a and P_2's class is b. Then, for each edge (a, b) we calculate its lift (i.e., $w(a, b) = p(b|a)\backslash p(b)$) to use as its weight. Based on these weights, we construct three dynamic network datasets N2, N3 and N4 by selecting the highest weighted 2, 3 and 4 edges for each vertex in each snapshot of the network. The size and density of the networks is presented in Tbl. I.

b) Trade Network: This is a trade network that models the yearly export and import relations of 192 countries from 1948 to 2000 based on the Expanded Trade and GDP Data [40]. The nodes model the trading countries and the direct edges model the export or import activity between two countries for a certain year. The snapshots of the dynamic network that we created corresponds to the trade network of each year, leading to a dynamic network consisting of $2000 - 1948 = 53$ snapshots. If the export amount from

TABLE II
NETWORK DENSITY.

Dataset	#IRS	#EIRS	ETime	PTime	TotalTime
N2	56725	2495	5	0	13
N3	1864660	45448	287	160	458
N4	16696859	103335	3226	16662	19901

The number of vertices in an IRS is 4 to 8, the maximum allowed vertex difference between two successive IRSs is 1, and ϕ is 4. Run times are in seconds. #IRS denotes the number of discovered IRSs. #EIRS denotes the number of discovered EIRSs. Etime denotes the amount of time spent to enumerate IRSs. PTime denotes the amount of time spent for path enumeration. TotalTime denotes the total amount of time spent to determine all EIRSs in the network.

country A to country B in a given year is more than 10% of the total export amount of A and total import amount of B for that year, a directed edge $A \rightarrow B$ is added to that year's trade graph.

c) Email Communication Network: This is a communication network that models the email traffic between employees of the ENRON company [41]. The nodes model the employees who are labeled by their rank and a node id to uniquely identify two nodes with same label (i.e. multiple employees with the same rank). The directed edges model the communications between the employees. To represent the dataset as a dynamic network, the entire time span was divided into 30 equal-size intervals. If an email was sent from node a to node b at a certain time interval, a directed edge $a \rightarrow b$ is added to the graph representing that time interval. This representation contains 130 nodes and about 90 edges per snapshot.

B. Performance Results

We evaluated the performance and scalability of our algorithm for mining the maximal EIRSs using the N2, N3, and N4 datasets from the patent citation network. Our evaluation is designed to assess how the density of the networks and the various parameters associated with the EIRS definition impacts the performance of the algorithm. All experiments are conducted on a Linux cluster with 6-core Intel Xeon X7542 "Westmere" processors at 2.66 GHz.

Note that in order to better assess how the interstate similarity component in the definition of the EIRS impacts the performance of the algorithm in all the experiments presented in this section, instead of using $|V_i \cap V_j|/|V_i \cup V_j|$ as a measure of inter-state similarity (constraint (iv) of Definition 1), we used the number of different vertices between V_i and V_j as a measure of distance. This allows us to explicitly increase/decrease the complexity of the mining problem by changing the number of different vertices that is allowed between successive IRSs.

1) Network Density: The performance of the algorithm for the three datasets is shown in Tbl. II. The datasets N2, N3 and N4 contain the same number of vertices 84152, but their density in terms of the number of edges present in the network increases by ∼1.4 times from N2 to N3 and by ∼1.2 times from N3 to N4. The results in Tbl. II show that as the graph density increases the number of EIRSs found increases (i.e. the number of EIRSs increased from 2495 in N2 to 45448 in N3 to 103335 in N4). At the same time, the total runtime to discover

TABLE III
INTER-STATE SIMILARITY.

ISDiff	$k_{\min}$	$k_{\max}$	#IRS	#EIRS	ETime	PTime	TotalTime
1	5	8	3037440	315	343	51	407
2	5	8	3037440	86731	342	116	472
3	5	8	3037440	75592341	346	409	768
1	6	8	2962777	2	338	46	398
2	6	8	2962777	5289	338	93	444
3	6	8	2962777	9781632	337	223	574

Dataset N4 is used for this experiment and $\phi = 5$. ISDiff denotes the inter-state distance capturing the maximum allowed vertex difference between two IRSs. $k_{\min}$ denotes the minimum number of vertices allowed in an IRS. $k_{\max}$ denotes the maximum number of vertices allowed in an IRS. Rest of the column labels are described in Tbl. II

TABLE IV
MINIMUM SPAN (ϕ) STUDY.

ϕ	QEdges	#IRS	#EIRS	ETime	PTime	TotalTime
5	5521	190491	112	21	1	34
4	9515	744053	18788	105	33	148
3	21726	5761948	1190067	1289	4729	6028

Dataset N3 is used for this experiment, the number of vertices in an IRS is 5 to 7, and the maximum allowed vertex difference between two linked IRSs is 1. QEdges denotes the number of edges in $\mathcal{N}^{\phi}$, and rest of the column labels are described in Tbl. II

the EIRSs increases from 13 seconds to process N2 to 458 seconds for N3 and 19901 seconds for N4. Even though the IRS enumeration step is mostly the time consuming process, as the number of IRS increases the time needed to traverse the IRS graph and discover the EIRSs starts to increase. For N4, about 84% of the total runtime was spent discovering the maximal paths. For sparse graphs, IRS enumeration time dominates the computation. For denser graphs, the direction graph building requires the most amount of computing.

2) Inter-state Distance: Tbl. III shows the performance of the algorithm for different values of the maximum allowed number of different vertices.

The number of different vertices is varied from 1 to 3 for two different sets of IRSs (i.e., a set of IRSs with 5 to 8 vertices and other set of IRSs with 6 to 8 vertices).

We observed that the number of discovered EIRSs increases as the maximum allowed vertex difference increases (i.e. the similarity threshold decreases). For the $5-8$ set, the increase in the maximum allowed vertex difference from 1 to 3 causes an increase in discovered EIRSs from 315 to 75592341. However, as we increase the maximum allowed vertex difference, the EIRSs will start containing unrelated IRSs in their path, since the similarity threshold between the IRSs are lower, which may represent less interesting EIRS. The decrease in similarity threshold also increases the total runtime, the path enumeration step takes longer to process more edges between IRSs.

3) Minimum Span: The performance of the algorithm for different values of minimum span (ϕ) is shown in Tbl. IV. The value of ϕ represents the minimum length requirement for an induced relational state to be in consistent state and for this experiments is in terms of years.

From these results, we observed that as the value of ϕ decreases, the number of discovered EIRSs and the runtime increases. The value of ϕ controls the number of edges that can qualify to be part of the $\mathcal{N}^{\phi}$ and the lower the value of ϕ is the more number of edges will qualify. In this case, we

TABLE V
IRS SIZE STUDY.

$k_{\min}$	$k_{\max}$	#IRS	#EIRS	ETime	PTime	TotalTime
5	5	212267	60	25	4	40
5	6	1039741	21059	108	41	158
5	7	4265612	46609	493	642	1144
5	8	16639693	57583	3221	15617	18852

Dataset N4 is used for this experiment, the maximum allowed vertex difference between two linked IRSs is 1 and ϕ=4. The column labels are described in Tbl. II

see that the number of qualified edges increase from 5521 to 21726 for ϕ=5 and ϕ=3. As the number of qualified edges in $\mathcal{N}^{\phi}$ increase, the number of IRSs increases and resulting in discovering higher number of EIRSs. Similarly, the runtime increase as the value of ϕ decreases from 34 seconds to 6028 seconds for ϕ=5 and ϕ=3, since the algorithm needs to process more number of IRSs to find relations and their evolution paths. This parameter is an important factor in finding EIRSs in different datasets, since the conserved state of a pattern is likely to be different depending on the type of the data.

4) IRS Size: We analyzed the performance of the algorithm for different sizes IRSs in Tbl. V. The size of an IRS is represented as the minimum and maximum number of vertices allowed in an IRS. For example, the size of 5-8 means that an IRS can contain minimum of 5 vertices and maximum of 8 vertices. We observe that as the size increases, the number of discovered EIRSs increases. Since larger range in size allows more number of IRSs to be detected, the chance of finding higher number of EIRSs is increases. In this experiment, the number of IRSs and EIRSs found for size 5 is 212267 and 60. When the size was increased to 5-8, the number of IRSs increased to 16639693 and in turn resulted with 57583 EIRSs.

C. Qualitative Analysis

In this section we present some of the EIRSs that were discovered by our algorithm in order to illustrate the type of information that can be extracted from the dynamic networks by focusing on how stable relations changed over time.

The EIRSs that are presented correspond to some of the EIRSs that show the highest change between the different IRS involved. In particular, given an EIRS, we computed the ratio of the total number of unique edges (i.e., relations) in all of its constituent IRSs over the total number of edges in the same IRSs. We refer to this quantity as the *total drift*. Note that this is just one of the many ways that can be used to assign a quantitative interestingness measure to an EIRS and other measures can be derived by looking at the nodes, relational inversions, cyclicality, etc.

In Fig. 5 we present an EIRS generated from the trade network capturing trade relations between some of the European countries over 30 years period and the chosen ϕ=3. The total drift for this EIRS is $12/18 = 0.67$. The EIRS mainly captures trade relations between Belgium, Netherlands, Germany and France. The other countries, such as Luxembourg, Italy and United Kingdom participate for a partial period of time. Based on the illustration, initially (during 1963 to 1967) Belgium and Netherlands were strong trade partners as they exported and imported from each other. The period 1970-1973 shows that the countries were heavily trading between each other. By evaluating the historical events, political and economic situation of that period, we could find the cause of higher trade activity. The periods 1980-1982 and 1984-1991 captures how France's trade relations with Belgium and Germany became one sided as France only imported from those countries. The cause of such changes could be that France was exporting to other countries or Belgium and Germany decided to import from some other countries. In Fig. 6 we present another EIRS generated from the trade network capturing a stable trade relation of USA with other countries over 35 years period. The total drift for this EIRS is $7/16 = 0.44$. We notice that USA and Canada have strong trade relations over a long period of time. Even though the strong tie in trading seems obvious due to the geographical co-location of the countries, it is interesting that the algorithm could discover such relation from the historical data. The EIRS also captures steady relation between USA and Japan.

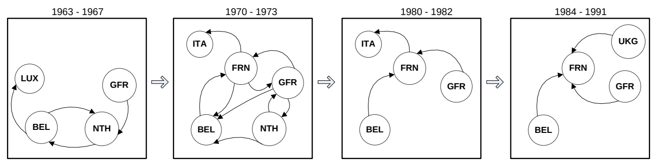


Fig. 5. An EIRSs capturing a trade relation between EU countries. The nodes in the figure are LUX=Luxembourg, BEL=Belgium, GFR=German Federal Republic, NTH=Netherlands, ITA=Italy, and UKG=United Kingdom.

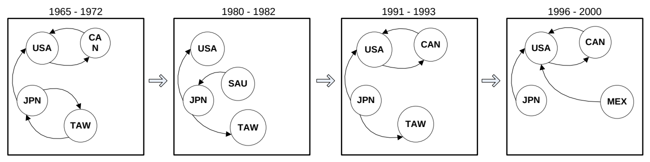


Fig. 6. An EIRSs capturing a trade relation of USA. The nodes in the figure are USA=United States of America, CAN=Canada, JPN=Japan, TAW=Taiwan, SAU=Saudi Arabia, and MEX=Mexico.

In Fig. 7 we present an EIRS generated from the email communication network capturing email exchange patterns among a group of employees over a period of time. The total drift for this EIRS is $7/12 = 0.58$. Although it is difficult to understand the communication of the employees without the message content, the direction of the communication can be found in this EIRS. We see that VP[id:33] had always initiated the conversation and was very active in email communication to have stable relations over all the captured periods. It is interesting to notice that VP[id:146] is not present in last period (25-28). One can investigate and confirm whether he/she was replaced or terminated and the cause for such actions.

In Fig. 8 we present an EIRS generated from the patent

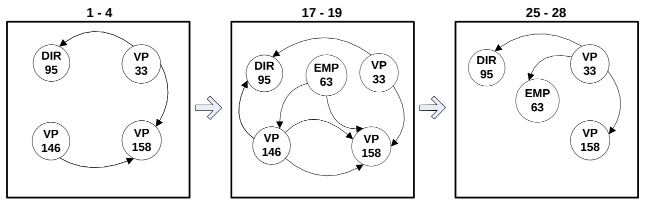
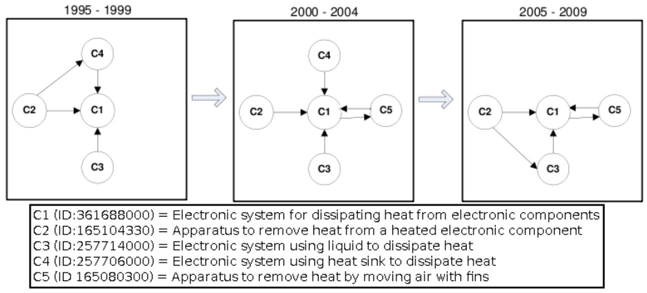


Fig. 7. An EIRSs capturing Enron email traffic pattern.



C1 (ID:361688000) = Electronic system for dissipating heat from electronic components
C2 (ID:165104330) = Apparatus to remove heat from a heated electronic component
C3 (ID:257714000) = Electronic system using liquid to dissipate heat
C4 (ID:257706000) = Electronic system using heat sink to dissipate heat
C5 (ID 165080300) = Apparatus to remove heat by moving air with fins

Fig. 8. An EIRSs capturing patent class relations. The nodes in figure denote patent classes and the legend captures the USPTO definition of the classes C1 to C5.

citation network capturing the evolution of the relations between some patent classes over a 15 years period. The total drift for this EIRS is $7/14 = 0.5$. Based on the illustration, the patent classes C2, C3 and C4 were citing C1 during 1995-1999. We can interpret that as the patents of class C1 are earlier inventions and later the patents of class C2, C3, and C4 used the ideas found in the patents belonging to C1. Over time the relations changed as class C5 appears in later years and both C1 and C5 are citing each other. It is possible that patents of class C5 represent a newer technology that cited earlier patents of class C1 as reference and the newer patent of class C1 are using C5 as reference. This captures a complex relational dependence between entities in a dynamic network. Moreover, we also observed that class C4 disappeared in the period of 2005-2009. This could indicate that the technology introduced in the products of class C4 is no longer used.

VII. Conclusion

In this paper we presented an algorithm for finding all maximal non-redundant evolution paths of the induced relational states in a dynamic network. This can be used to discover the transitions of the conserved relational states over time and to better understand the cause of such changes in the stable patterns in a dynamic network. Our experimental evaluation on multiple real world datasets show that the algorithm is able to discover interesting evolution paths from all datasets and can scale well to large and dense dynamic networks.

Acknowledgement

This work was supported in part by NSF (IIS-0905220, OCI-1048018, and IOS-0820730) and by the DOE grant USDOE/DE-SC0005013 and the Digital Technology Center at the University of Minnesota. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

References

[1] D. Boyd and N. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 11, October 2007.

[2] A. Chapanond, M. S. Krishnamoorthy, and B. Yener, "Graph theoretic and spectral analysis of enron email data," *Comput. Math. Organ. Theory*, vol. 11, no. 3, pp. 265–281, 2005.

[3] X. Liu, J. Bollen, M. L. Nelson, and H. Van de Sompel, "Co-authorship networks in the digital library research community," *Information Processing and Management*, vol. 41, no. 6, pp. 1462–1480, 2005.

[4] X. Hu, "Mining and analysing scale-free protein protein interaction network," *Int. Journal of Bioinformatics Research and Applications*, vol. 1, no. 1, pp. 81–101, 2005.

[5] Y. Koren, S. C. North, and C. Volinsky, "Measuring and extracting proximity graphs in networks," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 3, p. 12, 2007.

[6] M. J. Zaki, "Efficiently mining frequent trees in a forest," in *ACM KDD 02*. 2002, pp. 71–80.

[7] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient substructure discovery from large semi-structured data," in *Proc. of the 2nd Annual SIAM Symposium on Data Mining*, 2002, pp. 158–74.

[8] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Proc. of the 4th European Conf. on Principles of Data Mining and Knowledge Discovery*. Springer-Verlag, 2000, pp. 13–23.

[9] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraph in the presence of isomorphism," in *Proc. of the 3rd IEEE Int. Conf. on Data Mining (ICDM)*, 2003, pp. 549–552.

[10] M. Kuramochi and G. Karypis, "An efficient algorithm for discovering frequent subgraphs," *IEEE TKDE*, vol. 16, no. 9, pp. 1038–1051, 2004.

[11] ——, "Finding frequent patterns in a large sparse graph," *Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 243–271, 2005.

[12] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[13] J. Han, M. Kamber, and A. K. H. Tung, *Spatial Clustering Methods in Data Mining: A Survey*. Taylor and Francis, 2001.

[14] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE TKDE*, vol. 17, no. 8, pp. 1036–1050, 2005.

[15] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *KAIS*, vol. 14, no. 3, pp. 347–375, 2008.

[16] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[17] K. M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther, "Pattern mining in frequent dynamic subgraphs," in *IEEE ICDM*. Washington, DC, 2006, pp. 818–822.

[18] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *ACM KDD*. 2006, pp. 554–560.

[19] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis, "Mining graph evolution rules," *Machine Learning and Knowledge Discovery in Databases*, pp. 115–130, 2009.

[20] L. Cerf, T. Nguyen, and J. Boulicaut, "Discovering relevant cross-graph cliques in dynamic networks," *Foundations of Intelligent Systems*, pp. 513–522, 2009.

[21] C. Robardet, "Constraint-based pattern mining in dynamic graphs," in *IEEE ICDM*. 2009, pp. 950–955.

[22] A. Inokuchi and T. Washio, "Mining frequent graph sequence patterns induced by vertices," in *Proc. of 10th SIAM Intl Conf. on Data Mining*, 2010, pp. 466–477.

[23] D. B. West, *Introduction to Graph Theory*. Prentice Hall., 2001.

[24] S. Kramer, L. De Raedt, and C. Helma, "Molecular feature mining in hiv data," in *ACM KDD*, 2001.

[25] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Prefixspan: Mining sequential patterns by prefix-projected growth," in *ICDE*, 2001, pp. 215–224. [Online]. Available: citeseer.nj.nec.com/pei01prefixspan.html

[26] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraph in the presence of isomophism," in *IEEE ICDM*, 2003.

[27] H. Hu, X. Yan, H. Yu, J. Han, and X. Zhou, "Mining coherent dense subgraphs across massive biological networks for functional discovery," in *ISMB*, Ann Arbor, MI, 2005, pp. 213–221.

[28] J. Pei, D. Jiang, and A. Zhang, "On mining cross-graph quasi-cliques," in *ACM KDD*. 2005, pp. 228–238.

[29] X. Yan, X. J. Zhou, and J. Han, "Mining closed relational graphs with connectivity constraints," in *ACM KDD*. 2005, pp. 324–333.

[30] P. Desikan and J. Srivastava, "Mining temporally evolving graphs," in *WEBKDD Workshop*, vol. 22. Citeseer, 2004.

[31] B. Wackersreuther, P. Wackersreuther, A. Oswald, C. B
"ohm, and K. Borgwardt, "Frequent subgraph discovery in dynamic networks," in *Proc. of the 8th Workshop on Mining and Learning with Graphs*. 2010, pp. 155–162.

[32] R. Jin, S. McCallen, and E. Almaas, "Trend motif: A graph mining approach for analysis of dynamic complex networks," in *IEEE ICDM*. 2007, pp. 541–546.

[33] A. Inokuchi and T. Washio, "A fast method to mine frequent subsequences from graph sequence data," in *IEEE ICDM*. 2008, pp. 303–312.

[34] C. You, L. Holder, and D. Cook, "Learning patterns in the dynamics of biological networks," 2009.

[35] M. Koyutürk, Y. Kim, S. Subramaniam, W. Szpankowski, and A. Grama, "Detecting conserved interaction patterns in biological networks," *Journal of Computational Biology*, vol. 13, no. 7, pp. 1299–1322, 2006.

[36] T. Berger-Wolf and J. Saia, "A framework for analysis of dynamic social networks," in *ACM KDD*. 2006, pp. 523–528.

[37] M. Lahiri and T. Berger-Wolf, "Mining periodic behavior in dynamic social networks," in *IEEE ICDM*. 2008, pp. 373–382.

[38] L. Tang, H. Liu, J. Zhang, and Z. Nazeri, "Community evolution in dynamic multi-mode networks," in *ACM KDD*. 2008, pp. 677–685.

[39] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrieval," *Information Processing and Management*, no. 24, 1988.

[40] K. S. Gleditsch, "Expanded Trade and GDP Data," *Journal of Conflict Resolution*, vol. 46, no. 5, pp. 712–724, 2002.

[41] W. W. Cohen, "Enron email dataset," Website, 2005, http://www.cs.cmu.edu/ enron/.