Learning and Detecting Patterns in Multi-Attributed Network Data

Georgiy Levchuk, Jennifer Roberts, and Jared Freeman
Aptima Inc., 12 Gill Street, Suite 1400, Woburn, MA, USA 01801
georgiy@aptima.com

Abstract
Network analysis is a growing field across many domains, including computer vision, social media marketing, transportation networks, and intelligence analysis. The growing use of digital communication devices and platforms, as well as persistent surveillance sensors, has resulted in explosion of the quantity of data and stretched the abilities of current technologies to process this data and draw meaningful conclusions. Current tools either require significant levels of manual intervention (e.g., to prepare the data, to define patterns, or to draw conclusions from data) or are unable to generalize to new data sources and analysis needs.

In this paper, we present automated solutions to two major problems in network analysis: (a) finding patterns in the network data that contains high levels of noise and irrelevant information; and (b) learning repetitive patterns and dependencies between entities and attributes. Our modeling framework represents network data using multi-attributed graphs that can encode various discrete and continuous features and relationships between network entities. The pattern search and learning model is based on probabilistic multi-attributed graph matching, and implemented using distributed message passing algorithms. Our algorithms achieved high accuracy rates in learning and finding patterns in the data, are flexible to new domains and data types, and scale to large datasets using the Map-Reduce framework.

1. Introduction

1.1. Operational need and organization of this paper
One of the key challenges facing U.S. intelligence analysts in cyber operations and current military engagements in Afghanistan and elsewhere is the capture and maintenance of profiles, relationships and interactions that occur within the local populations. Analysts use this information to identify recruiters, members of militia and their supporters, weapons suppliers, potential places of operation, training grounds, and targets that insurgents might want to attack. Commanders on the ground also use this information to understand how local power is distributed and how to establish relationships that allow them to influence the attitudes and actions of the local population.

As the need for actionable intelligence has grown, the quantity of data available continues to increase faster than our ability to automatically analyze and extract mission-critical information. Adding sensors into the field without means for identifying suspicious patterns only introduces more noise into the system while offering limited returns. This increases information overload for analysts who sift through data manually, and hence potentially important data may not be processed in operationally relevant timelines. The analysts need automated decision support tools to infer meaning from multi-source intelligence, identify patterns, and discover suspicious activities, people, and groups. Since relational data representing influence, interactions, and dependencies among entities, events, and locations is of highest importance, these problems can be formalized as network analysis problems.

In this paper, we present automated solutions to two major problems in network analysis: (a) finding patterns in network data with high levels of noise and irrelevant information; and (b) learning repetitive patterns and dependencies between entities and attributes. Our algorithms can infer hidden attributes (e.g., profile characteristics or roles of actors), find patterns (groups of people involved in coordinated activity, or a pattern of person-event relations) in noisy data given a pattern library, and learn repetitive patterns (to populate the library) in supervised and unsupervised manner.

Algorithms presented in this paper are extensions of our previous work in probabilistic attributed network matching (Levchuk et al., 2007; Levchuk, Mendoza-Schrock, and Shebilske, 2009; Levchuk, Bobick, and Jones, 2010). To perform network search and inference, our baseline model finds the matching between model networks and data. Network matching algorithms return data subnetworks rank-ordered by probability of the match, and assign inference labels based on the semantics of matched patterns. Our first contribution presented in this paper is a new algo-
algorithm for pattern matching based on graph factorization and Loopy Belief Propagation (LBP), which provides improvements in convergence and scalability.

The default assumption of our network analysis model, as explained in Section 1.2, is that network patterns are specified by analysts based on domain knowledge or theories of group behavior. However, detailed manual pattern definition may not be possible due to expertise, time and manpower constraints. Our second and primary contribution presented in this paper is a new model for learning repetitive attributed network patterns from data in supervised and semi-unsupervised manner. This algorithm is based on iterative model network update and approximate pattern matching.

Our paper is organized as follows. First, we present a motivating example in section 1.2. Next, we summarize related research in section 1.3. We provide notations, describe our network matching problem formulation, and present the improved loopy belief propagation (LBP) algorithm in Section 2. We discuss the challenges encountered by LBP when data network contains multiple subnetworks matching the same model network, and describe our sampling-based solution approach to find multiple inexact matches in large networks. Section 3 details the network learning algorithm and describes future work in this area. Section 4 includes description of the data set and results of computational experiments we conducted to test the efficiency of our solutions in learning patterns and finding them in the network data with large number of irrelevant nodes and links.

1.2. Motivating example

When processing information about networks of people, places, and events, analysts would like to automatically detect entities of interest based on their relationships or unusual patterns of activities and interactions. Since the data is often noisy and ambiguous, and activities are performed by more than one entity, analyzing and classifying one entity at a time results in low recognition rates. Making inferences by finding the match between networks representing the patterns of interest, sometimes referred to as models or queries, and the observed networks, referred to as data, allows resolving gaps in the data and increasing semantic relevance of generated inferences.

Figure 1 illustrates an example of a network analysis problem where analysts are interested in finding an attacker, a reconnaissance/access person, a target, and a weapons supply location (Figure 1a) in a large network containing people, groups, events, and places. The data rarely contains exactly the information analysts are looking for; that is, the facilities in the data will not be marked as “target of attack” and people will not be marked as “attackers,” otherwise the network inference will be trivial. Instead, analysts must draw these inferences from the ambiguous entities, events and features in the data. Entities in the data (a subset of data depicted in Figure 1b) are associated with attributes, and several types of relations may exist between them. The analysts encode their queries using the attributed graph, where entities are people and location nodes, node attributes represent the features of people and events at the locations, and links indicate relationships and interactions, such as physical meetings, social and family relations, communications, and material and financial transactions. In Figure 1a, the search objective is encoded as a pattern that contains information about the attacker (engineer capable of assembling the explosive), the target (government facility), a supporter who may provide reconnaissance on and access to a target (a person who works at a facility), and the location of weapons assembly (a place where the attacker resides for a period of time and at which the loading activities may be observed).

Figure 1: Example of network analysis

Several subnetworks in the data may contain information relevant to the roles and relations in the analyst’s query, but due to missing information, ambiguity, and variation in the observed evidence, we cannot expect to find many exact matches in the data. Instead, the users are interested in finding partial matches and rank-ordering the results. The example in Figure 1 resulted in two matches, neither of which perfectly match the original pattern. The first match,
depicted on the left in Figure 1c, assumes that the observed relation <friends> between Jon and Chris may mean that they could <team_up> for a hostile activity, while the second match, depicted on the right, assumes that sending money could be a pay-off for the team-up activity. The second match may be less relevant to the analyst’s query, because knowledge of Rick living at facility F2 may be dated, and loading activity may not have been observed at the facility within the last six months.

When analysts can succinctly describe the network pattern they are interested in, they can use network pattern matching algorithms to find relevant subsets of data. However, adversarial activity networks are often very complex and adversaries change their behavior over time to adapt to the environment and their opponents. As the result, analysts would benefit from automated methods that learn network patterns from data. Figure 1 depicts two instances of the same pattern, and we may expect that more examples of these activities occur in the data. However, obtaining multiple instances of hostile activities needed for robust machine learning can be challenging, because hostile activities occur less frequently than ordinary day-to-day interactions.

1.3. Related research

Network data exploitation tools currently available to analysts provide limited analytical capabilities and can be categorized into four groups. First, network visualization applications used by intelligence community (AXIS PRO, AnalystNotebook, Palantir, and StarLight) and researchers (Gephi, UCINet) enable easier viewing, editing, and browsing of large network data but require performing manual classification of the networks and their members. Second, clustering algorithms can identify similar entities or tightly connected communities, but are unable to find heterogeneous subnetworks. Third, methods for gathering statistics about the network can be used to analyze trends in network interactions and features over time; however, these solutions cannot find suspicious groups and activities when the amount of irrelevant data exceeds the amount of data related to hostile actions. Finally, database search engines and triaging systems allow querying and retrieval of relational data; these tools, however, are limited to finding only exact matches and require users to specify detailed queries.

In the last two decades, researchers have been developing solutions for network analysis in several diverse domains. For example, the SPARQL has emerged as a state-of-the-art RDF query language for graphs encountered in semantic web applications (Prudhommeaux, and Seaborne, 2007). SPARQL performs exact pattern matching, and thus is not appropriate in the domains of network exploitation where noisy, missing, and ambiguous data are common, and where analysts often need to find the set of best approximate matches.

Recently there have been studies on approximate matching with large graphs, e.g. SAGA (Tian et al., 2006), TALE (Tian and Patel, 2008), SIGMA (Mongiovì et al., 2010), and G-Ray (Tong et al., 2007). However, these algorithms use the measures of edit distance and missing links in models of objective function, and thus do not reason well about proximity between the attributes of nodes and are not scalable to larger data set sizes. To alleviate these challenges and speed up the query process, information propagation models have been proposed (Khan et al., 2011) that reason about the label of the node based on the labels of its neighbors. While efficient at pruning the search space, this model is sensitive to the labels of irrelevant entities interfering with the relevant features and thus may result in increased false alarm rates.

Traditional behavior profiling and network analysis techniques have severe limitations as well. Social network analysis (SNA) models apply simplified assumptions to compute structural graph measures for role classification of observed nodes (Xu and Chen, 2005; Monge and Contractor, 2003) and network state tracking (Coffman and Marcus, 2004; Chung et al., 2006). These models lose semantic and structural properties during the aggregation process and thus are unable to find small heterogeneous patterns in large network datasets. Many graph-theoretic models (Zheleva, Getoor, and Sarawagi, 2010; Getoor and Diehl, 2005) make inferences about the whole observed network, assuming that all data are generated by a single model describing dependencies between attributes of nodes and links, which prevents us from learning and detecting distinct patterns in network data.

Parallel plan recognition models including propagation network models (Shi et al., 2004) and factor-graph based methods (Davis, Olson, and Carley, 2008) can reason about multiple parallel observations and track evolution of hidden plans and goals. However, these models assume that the association of data entities with model entities has already occurred and that little or no irrelevant data are present, which is unrealistic in real-world applications.

The algorithms presented in this paper are extensions of the model developed in (Levchuk et al., 2007). We drew original inspiration for using probabilistic attributed graph matching for network learning and search applications from computer vision domain. Researchers in (Luo and Hancock, 2001; Wilson and Hancock, 1997) approached graph matching by finding a mapping of nodes between the model and data graphs that satisfied structural and attribute consistency and employing expectation-maximization algorithm. While researchers used only node attributes and modeled the objective function as a likelihood of data given the model, it gave us an inspiration to formulate the network search as a quadratic assignment problem, to
which several non-linear optimization techniques were applicable (Grande et al., 2008; Rangarajan, Yuille, and Mjolsness, 1999). We then reformulated the problem as optimizing a posterior probability of node-to-node mapping between the model and data networks (Levechuk, Bobick, and Jones, 2010), and were able to apply belief propagation (BP) algorithms extending the work on loopy BP methods in assignment problems (Bayati, Shah and Sharma, 2005; 2006).

2. Network Matching Model

2.1. Definition of attributed network

To efficiently encode the semantic content and social structures in the networks, we represent each network as multi-attributed graph $G = (V_G, E_G, A_G)$ consisting of nodes $v \in V_G$, links $(v, w) \in E_G$, and their attributes $a_{vw} \in A_G$. Node attributes $a_{v}^N$ define profiles of activities, roles, actors, and geographic nodes while link attributes $a_{vw}^L$ define profiles of relations among those entities, encoding semantic relations, temporal and spatial dependencies, interactions, and influence among different nodes. Values of individual attributes in our representation can be single numbers, ranges, or distributions, to flexibly handle uncertainty and variability of behavior.

Data networks represent observations about specific entities and their relations in collected data. Model networks represent queries, hypotheses or inferences concerning (typically suspect) node-link structures of interest to the analyst. For example, data nodes can include actors (specific individuals or groups) and geographic areas (locations and facilities), while model nodes can include tasks (short-term activities performed by actors at geographic areas; e.g., weapons storage, reconnaissance) and roles (conceptual abstractions of behaviors, intents, and activities associated with actors or areas). We represent data networks using attributes $A_D = [a_{nm}^D]$, and model networks using attributes $A_M = [a_{km}^M]$. Data networks can range from hundreds to billions of nodes in size, while model networks generally consist of tens of nodes.

2.2. Graph matching formulation

We view the data network as partially generated by a set of hidden model networks. When the data network is noisy, its subgraphs will not perfectly match the model network and thus we must find subgraphs with the “best partial match” using probabilistic multi-attribute network analysis. A single match is represented as a set of assignments, called a mapping, from nodes in model network to nodes in data network (Figure 2).

Formally, a mapping is defined as vector $X = [X_k]$, where for each model node $k \in V_M$, the variable $X_k \in V_D$ represents a mapped node in data network. Similarly, a link $(k, m) \in E_M$ is mapped to a data link $(X_k, X_m) \in E_D$. The mapping not only returns a subnetwork in the larger data network that matches the query, but also represents joint inferences about the nodes and links in that subnetwork supported by the observed data $D$. For example, the variables of the model node $k$, such as its attributes $a_{km}$ or higher-level semantic label associated with the node, are interpreted as inferences made about the observed (data) node $X_k$. In the example of the first mapping on the left in Figure 1c, the model node with role label “attacker” and attribute <engineer> (which corresponds to the skills of the person that could be observed) is mapped to a real person named Chris. Multiple inferences are obtained from multiple attributes of the model node and by mapping different model nodes to a single data node.

![Network pattern matching and its variables](image)

Figure 2: Network pattern matching and its variables

General network matching is known to be NP-hard: for example, there are over $10^{63}$ subnetworks of 10 nodes in $10^7$-node data network, and clearly exhaustive search is infeasible. Hence, we must apply stochastic search to scale to such data sizes. Hence, we model the mapping vector $X$ as a set of dependent random variables, where their joint distribution conditioned on the observed data is often represented using a conditional Markov random field (Namata, Kok, and Getoor, 2011):

$$P(X|D) = \frac{1}{Z(D)} \prod_{c \in C} \varphi_c(x_c, d_c),$$

where $Z(D) = \sum_X \prod_{c \in C} \varphi_c(x_c, d_c)$ is a normalization factor independent of $x$, $C$ represents a set of subsets (cliques) of indices from $V_M$, and for a specific subset $c \in C$, the notation $X_c = [X_k, k \in c]$ represents a subset of corre-
sponding random variables while $x_c$ denotes their values, and $d_c$ represents a set of observations that correspond to data variables associated with nodes $x_c$ (i.e., the nodes and links in the data network to which the nodes/links from a subset $c \in C$ are mapped). The functions $\varphi_c(x, d)$ are called potentials and represent the correspondences between the model and data variables, and often represented using a log-linear combination over a set of weighted features $\varphi_c(x, d) = \exp(-w_c \cdot f_c(x, d))$. They can capture a range of dependences, including local and relational similarity metrics. The above formulation is used to solve two problems: (a) train the classifier to find the weights $w_c$ of potentials; and (b) find the mapping (i.e., find identities of actors) by maximizing the posterior probability:

$$\hat{x} = \arg\max_x P(x|D).$$

(2)

In multi-attributed network matching, the minimum assumption is that the node mapping depends on the mapping of neighboring nodes. The corresponding formulation results in pairwise Markov random field, where the potentials $\varphi_c(x, d)$ and corresponding feature functions $f_c(d)$ represent only individual decisions and pair-wise dependencies (Sen et al., 2008). This results in a simplified conditional posterior probability:

$$P(x|D) = \frac{1}{Z(D)} \prod_k \varphi_k^1(x_k) \prod_{km} \varphi_{km}^2(x_k, x_m).$$

(3)

Traditional network analysis models seek to find inferences about every observed nodes in the data network and thus define above formulation over a vector $X$ that represents the labels of all data nodes (Namata, Kok, and Getoor, 2011; Sen et al., 2008). This creates three problems. First, this requires large training corpus and makes trained classifier not able to deal with previously unseen data. Second, the complexity of classification stage increases since the number of the products in equation (3) is growing with the size of data network. Finally, the potentials must be defined well, and learned weights cannot be converted into “learned patterns” and interpreted by the analysts.

Instead of defining abstract potentials and learning their weights, we posed that there exist different patterns of changes to the LBP formulation. First, we replace the discrete mapping vector with 0-1 mapping matrix, which is further approximated with the continuous matrix. Replacing the mapping vector with a matrix is equivalent to smoothing the objective function and allowing the search for optimizing solution to be performed over a set of continuous variables instead of discrete ones. Second, we change the update equations in the LBP with incremental updates to have more gradual changes to the objective function. These enhancements are discussed in next section.

2.3. Min-sum Loopy Belief Propagation

LBP algorithm finds marginal posterior distributions $b_k(i) = P(X_k = i|D)$ to maximize the joint mapping posterior (3). LBP operates on a factor graph constructed from a factorization structure in (1). Factor graph is a bipartite graph containing two types of nodes: factor nodes representing each subset $c \in C$, and variable nodes representing each node $k \in V_M$. Links between factor and variable nodes correspond to a membership of variable node in a subset $c$. LBP works by passing real-valued functions, called messages, between factor and variable nodes in a factor graph. These messages contain the “influence” that one variable exerts on another. LBP achieves its computational efficiency by reducing the complete marginalization to a sum of products of simpler terms than the ones appearing in the full joint distribution. It was shown (Weiss, and Freeman, 2001) that LBP finds a solution that is optimal in a large neighborhood of mappings (a property weaker that global maximum but stronger than local optimum).

The min-sum version of LBP algorithm (Pearl, 1988) modifies updates from summation and multiplication operations in traditional LBP to summation and maximization operations, resulting in improved computational efficiency. The updates include two steps. First, we update messages sent from factor node $k$ to variable node $i$: $l_{k \rightarrow i}(x_i) \propto \max_{x_k} \{\log \varphi_k(x_k, d_k) + \sum_{j \in N(k) \setminus \{i\}} l_{k \rightarrow j}(x_j)\}.

Second, we update marginal log-posterior variables that are sent from variable nodes $k$ to all adjoined factor nodes:

$$\mu_k(x_i) \propto \sum_{x_j \in X_i} e^{l_{k \rightarrow i}(x_i)} = \sum_{x_j} e^{\mu_j(x_j)} = 1.$$  

In the above updates, we normalize the messages to assure corresponding probabilities sum to 1: $\sum_{x_i} e^{\mu_k(x_i)} = 1$. At the end of algorithm iterations, we calculate the marginal posterior distribution as $b_k(i) = e^{\mu_k(i)}$. 

28
2.4. Reformulating matching objective function

We replace the mapping vector $X$ with mapping matrix $S = [S_{kl}]$, where $S_{kl} = 1$ if $X_k = i$, and 0 otherwise. This enables us to represent the posterior distribution (Levchuk, Bobick, and Jones, 2010) as

$$P(S|D, M) ∝ \prod_{k=1}^{m} p(d^D_k | a^M_{k,k}) \prod_{k=1}^{m} \prod_{l=1}^{m} p(a^D_l | a^M_{l,k}),$$ \hspace{1cm} (4)

where $a^D_l, a^M_{k,k}$ are respectively attribute vectors of observed (data) node $i$ and model node $k$; $a^D_l, a^M_{l,k}$ are the attribute vectors of data link $(i,j)$ and model link $(k,m)$; and conditional likelihood probabilities $p(d|m)$ are based on the attribute noise model. Note that the attributes on nodes and links can represent any features of social network interactions, including metadata, structural information (e.g., frequency of communications), and semantic information (topics, family connections, intents, and knowledge expressed by actors or in actor-to-actor interactions).

Taking a negative log of the new posterior distribution (4), we obtain the quadratic assignment objective function that defines a match between model and data network:

$$Q(S) = \sum_{i \in V^D} C_{kl} S_{kl} + \sum_{(km) \in E^M} C_{km} S_{kl} S_{mj},$$ \hspace{1cm} (5)

where coefficients are node and link mismatches (Figure 2b):

$$C_{kl} = -\log p(a^D_l | a^M_{k,k}) C_{km} = -\log p(a^D_l | a^M_{l,k}).$$ \hspace{1cm} (6)

The number of summation components in objective function (5) is equal to the number of nodes and links in model network, and thus is typically small. Converting the original problem into an assignment formulation allows us to use a min-sum version of LBP, and enables simpler updates during learning model attributes from historic network data corpus. The function $Q(S)$ is a measure of the mismatch between the model and data network under mapping $S$. As the result, we can interpret its minimum value $\min_S Q(S)$ as a “weak” distance (non-symmetric and not conforming to triangle inequality) between model and data networks.

2.5. Finding network matching via LBP algorithm

Using LBP algorithm, we approximate marginal posterior distributions of mapping from the model nodes to data nodes

$$b_k(i) = P(s_{kl}|D, M)$$ \hspace{1cm} (7)

by passing “messages”, that carry beliefs about matches of model nodes and links in the data, between variable nodes and factor nodes defined for the links in model network.

Values $b_k(i)$ are then used to generate the final mapping vector $X_k$. In case a single mapping should exist in the data, we can compute $X_k = \arg \max_{b_k(i)}$. Approaches to derive multiple matches are described in section 2.7.

From equation (4), the factor nodes and corresponding factor graph in the problem of matching model network to a data network are defined for each link in the model network, while variable nodes correspond to the original nodes in model network with variables in those nodes correspond to assignment of data nodes to corresponding model node. For each link in the model network factor nodes are created; each factor node sends out two messages, each being an array of double variables computed iteratively. Figure 3 shows an example of model network, corresponding factor graph, and the messages that are passed from factor nodes to update marginal posterior distribution.

For pair-wise Markov random fields (3), we simplify min-sum algorithm to perform the following steps. First, for each link factor we update two factor messages, representing the marginal log-probabilities of matching model link $(m,k)$ to the data link that ends in node $j$, $f_{(m,k)}(j)$, or starts in node $j$, $r_{(m,k)}(j)$, and send them to variable nodes:

$$f_{(m,k)}(j) \propto \max_i (-C_{mkij} + \mu_m(i) - r_{(m,k)}(i))$$ \hspace{1cm} (8)

$$r_{(m,k)}(j) \propto \max_i (-C_{mkij} + \mu_k(i) - f_{(m,k)}(i))$$ \hspace{1cm} (9)

Second, we use received messages in variable nodes to update node messages $\mu_m(i)$ that represent marginal log-posterior probability of mapping model node $m$ to data node $i$:

$$\mu_m(i) \propto -C_{mi} + \sum_{(lm) \in E^M} f_{(lm)}(i)$$ \hspace{1cm} (10)

$$+ \sum_{(m,l) \in E^M} r_{(l,m)}(i)$$

![Figure 3: Factor graph for pattern matching](image)

2.6. Improving performance and scalability of LBP

We simplify the updates in (8-9) by first computing the values

$$\tau_{(m,k)} = \max_i \left( \mu_m(i) - r_{(m,k)}(i) \right),$$ \hspace{1cm} (11)

$$\tau_{(m,k)} = \max_i \left( \mu_k(i) - f_{(m,k)}(i) \right)$$

and then updating the messages as
attributes and parameters of subnetworks that occur multiple times in the data corpus. Attributes of the model network \( A_M = \{a_{lk}M\} \) can be learned iteratively in supervised or unsupervised manner. In supervised situation, we are given a set of \( N \) observations of network states \( D(t), t = 1, ..., N \) with attributes \( A_D(t) = \{|a_{lk}(t)|\} \), which is called training network corpus. We then estimate the parameters of the model that most likely generated such data by maximizing posterior probability of the model given its observed instances:

\[
A_M = \arg \max_A P(A | A_D(t), t = 1, ..., N)
\]

In the unsupervised situation, we first cluster the network instances into groups \( T_k; \cup_{k=1}^K T_k = \{1, ..., N\} \) based on their similarity computed using the network mismatch function descried in section 2.4. Then we learn a network pattern for each subset \( T_k \):

\[
A_{m,k} = \arg \max_A P(A | A_D(t), t \in T_k).
\]

To compute objective function in (15), we either need the knowledge of network mapping between each data instance \( D(t) \) and model network \( M \), or need to perform marginalization over all possible mappings between the model network and its observed instances:

\[
P(A | A_D(t), t \in T_k) = \sum_{S_k} P(A, S_k | A_D(t), t \in T_k)
\]

To perform parameter learning, we will use a variant of expectation maximization (EM) algorithm that will treat mappings as unobserved variables and iteratively update model network parameters.

### 3.2. Expectation maximization algorithm

Given the set of observed variables \( X \), set of unobserved variables \( Z \), and a set of parameters that must be learned \( \theta \), the expectation maximization (EM) algorithm seeks to find maximum likelihood estimate (MLE) of the marginal likelihood \( P(X|\theta) = \sum_Z P(X,Z|\theta) \). Since computing this function is often intractable, the EM algorithm performs iteratively two steps:

- **Expectation step** (E step) computes the expected value of log-likelihood function with respect to conditional distribution of \( Z \) given \( X \) under current estimate of parameters \( \theta^n \):

  \[
  Q(\theta | \theta^n) = E_{Z \mid X, \theta^n} [\log P(X,Z|\theta^n)]
  \]

- **Maximization step** (M step) finds the parameter that maximizes the above function

  \[
  \theta^{n+1} = \arg \max_\theta Q(\theta | \theta^n).
  \]

The EM algorithm terminates either after fixed number of iterations or when no parameter changes are obtained.

### 3.3. Multi-attributed network learning algorithm

We apply the concept of EM for network pattern learning, modifying its implementation from working with likelihood function to using the posterior distribution. This is
dictated by the need to avoid dealing with large number of components in the expectation step. Informally, the EM is applied iteratively to compute the “best” values of \( \theta \) given current estimates of parameters \( \theta^t \) (E step), and then use just computed values of \( Z \) to compute a better estimate of parameters \( \theta^{t+1} \).

In the context of learning attributed network patterns, the observed variables are network training corpus \( X = \{ A_D(t), t = 1, ..., N \} \), the unobserved variables are the mappings \( Z = \{ S_i, t = 1, ..., N \} \) between model network \( M \) and data network instances \( D(t) \) in the corpus, and parameters to be estimates are the parameters of attributed model network being learned \( \theta = A_M \). Our algorithm proceeds as follows:

**Step 1:** Initialize model network attributes \( A_M^0 \). This can be done by selecting one of the networks in a corpus or doing an averaging among them.

**Step 2:** Find the set of mappings \( S_i[r] \) near the maximum of posterior distribution \( P(S_i[r] | A_M^0, A_D(t)) \).

**Step 3:** Find the new model attributes using these mappings to optimize

\[
Q(A | A_M^t) = E_{S_i[A_D(t)],A_M^t}[\log P(A,S_i[r] | A_M^t, A_D(t))]
\]

\[
\propto \sum_{tr} p_{tr}(\sum_{k} C_{k1}^r S_{k1}^r + \sum_{(k,m),(i,j)} C_{km,i}^r S_{km,i}^r S_{m1}^r),
\]

where \( p_{tr} = P(S_i[r] | A_M^t, A_D(t)) \) is a posterior probability of mapping \( S_i[r] \) conditioned on model and data networks. In the case of Gaussian error model, the mismatch functions are weighted L2 norms: \( C_{km,i}^r \propto (a_{km}^r - a_{ij}^D(t))^T \Sigma^{-1}(a_{km}^r - a_{ij}^D(t)) \).

In a simple Euclidean case (i.e., modeling attribute error with Gaussian distribution), we obtain that model node and link attributes are updated via weighted sum of data attributes mapped to corresponding model nodes and links, where the weights are probabilities \( p_{tr} \) of the mappings found in step 2:

\[
a_{kk}^M = \sum_{tr} p_{tr} \sum_{k} S_{k1}^r a_{k1}^D(t) \quad (17)
\]

\[
a_{km}^M = \sum_{tr} p_{tr} \sum_{(k,m),(i,j)} S_{km,i}^r S_{m1}^r a_{ij}^D(t) \quad (18)
\]

The steps 2-3 are executed until the convergence criterion is reached. Step 2 seems to be equivalent to solving a network matching problem, where the network mismatch coefficients are changing with the change in model network attributes, and the straightforward execution will result in such approach being computationally intractable. However, since EM will change the model attributes only gradually (assured by employing the incremental update as defined in (14)), we perform a limited set of LBP iterations and sampling to generate set of mappings \( S_i[r] \), making the algorithm computationally feasible.

## 3.4. Generating network training corpus

Oftentimes in unsupervised network learning, a training corpus of subnetworks is not available. Instead, we are given a single large data network \( D \) with attributes \( A_D \). We can then segment this network into subnetworks for the training corpus using one of the following approaches. First, many large data networks contain disconnected subnetworks, which are natural candidates for the inclusion in training corpus. Second, the observations can often be segmented in time and space, thus providing us with subnetworks to generate training network corpus. Finally, we developed an algorithm to perform data segmentation using **mapping of structural primitives**. This approach is based on a generative model of the network formation, which assumes that structurally there is a limited set of primitive (or elementary) subnetworks that could occur in the data. An example of seven such subnetworks is shown in Figure 4. We then generate a training network corpus by finding matches of primitive network elements in the data network (using only structural attribute information) and using the examples of structural primitives to learn the patterns in unsupervised manner. This approach is useful when we do not know the attributes of the networks but can hypothesize the structure of dependencies between the entities.

![Figure 4: Structural network primitives](image)

### 4. Computational Study

We used two datasets to conduct computational experiments to assess the scalability and accuracy of network search and learning algorithms. The first dataset was randomly generated using the procedure described in section 4.3 so that it contained embedded instances of pre-specified patterns. As the result, we knew which patterns should have been detected and learned from dataset 1, and we used this dataset and the measures described in section 4.2 to assess the accuracy of our approach. The second dataset was a generic network with a typical scale-free structure and sparsely populated entities, but with no correlation to real-world entities or networks. Studying this dataset provided us with an understanding of the data statistics, such as the ability to decompose the data into disconnected subnetworks, which facilitated network learning and enabled improved scalability. While the “ground truth”
for this data was not available, we treated some of the attributes as true and assessed the ability of our algorithms to find them in the data.

4.1. Outputs of network matching and learning for computational experiments

We use the outputs of network matching algorithms to generate the following two inferences:

- The pattern inference, defining where we find a query, is the tuple \((m, p(M_m))\) associated with the whole data network or its segmented (e.g., disconnected) subnetwork, where \(k\) is the index of model network \(M_m\), and \(p(M_m)\) is a probability, which we compute as a maximum soft-assign probability of the mapping for this pattern

\[
p(M_m) = \max_p P(S[r]|D, M_m)
\]

\[
= \frac{1}{\gamma_m} e^{-\beta_m \max_r q(S[r]|D, M_m)},
\]

where \(\{S[r]\}\) is the set of feasible mappings with the minimum mismatch obtained by network matching algorithm, and \(\beta_m, \gamma_m\) are pattern-specific scaling coefficients.

- The node attribute inference, defining where we find a query, is the tuple \((j, p(a[j]))\) associated with the data node \(i\), where \(j\) is the index of attribute, and \(p(a[j])\) is the probability of this inference, which can be computed as \(p(a[j]) = \max_{i, \lambda, K_j = 1} s_{mu}[r] \times P(S[r]|D, M_k)\), i.e., maximum mapping probability that mapped model node with the attribute \(j\) to the data node \(i\). One of the attributes is the type of model node, which is not observed in the data and must be recovered by the algorithm. For example, a sink network depicted in Figure 4b has one “center” node and five neighbor nodes that may look exactly the same in terms of their descriptor attributes and links to the center node. It is thus important, when retrieving a data subnetwork matching this pattern, to know which of the subnetworks’ nodes is the center and which other ones are the neighbors, but it may not matter which of the specific neighbors (2-6) they map to.

The set of inferences above is accepted if their corresponding probabilities are above the threshold, varying which we can assess the network matching algorithm and select optimal decision acceptance range.

The network learning algorithm returns the network with attributes and the corresponding data mappings, which are used to generate the three inference statements described above. We can then find a mismatch between learned network(s) and true pattern(s) that generated the data.

4.2. Algorithm performance measures

We evaluated our network matching and learning algorithms using four standard evaluation metrics: Precision, Recall (true positive rate), and Accuracy (Makhoul et al., 1999), computed as follows:

- Precision is the percentage of inference statements made by the algorithm that are correct;
- Recall is the percentage of true inferences found by algorithm;
- Accuracy is the number of data variables correctly inferred to be part of a pattern or correctly classified as irrelevant data divided by the total number of data variables.

Precision measures the exactness of the returned network search results – i.e., how often the retrieved subnetworks and their variables are correct. Recall measures the completeness of the predictions – i.e., how many true matches we can find. False positive rate is degree of specificity, - i.e., the percentage of irrelevant variables returned as the result of network search. Accuracy is the aggregate metric of the algorithm’s performance.

4.3. Random network generation model and experiment design

The random data network generation algorithm we developed proceeds in the following steps. First, we select patterns from primitive structure library defined in Figure 4. Then, we generate a number of attributed networks for each structural primitive, obtaining true patterns. Next, we generate multiple instances of each pattern, obtaining true subnetworks. Then, we connect these instances with several links to form a fully connected graph. Finally, we add errors to this network by adding a uniform error to attribute values on nodes and links within a range of certain percentage of the true value. Finally, we add irrelevant nodes and links to this network to obtain a final observed data network. The generation of attributes on nodes and links, as well as attributes of irrelevant data, is done uniformly and guided by a set of parameters defining the number of attributes and their min and max value ranges. We control the amount of irrelevant data using min-max ranges for irrelevant nodes and links. The data network generation process is depicted in Figure 5.

As the result of this workflow, we have ground truth about the actual patterns that generated the data network, how many instances of those patterns are in the data, and where those instances are located. Since the data network is fully connected, we need to perform unsupervised network learning using the segmentation via structural primitives. We start by assuming that any of the structural model networks could be in the data, and thus segment the data network into a training corpus using the process described in section 3.4. We then learn the attributes using the algorithm presented in section 3.3, potentially learning multiple model networks with the same structure distinguished only by the networks’ node and link attributes. We make a final decision to accept or reject a learned model network based on the frequency (number of mappings to different data
subnetworks) and quality (mismatch with these subnetworks) of the model’s occurrences in the data.

Accordingly, we designed an experiment to test both learning and matching algorithms. We tested the following criteria:

- **Node type detection accuracy**: given the ground truth about where the true subnetworks are, we score the ability of our algorithms to find them by analyzing the accuracy of node type associations. The node type is an attribute that is unobserved but represents the label that indicates whether the node is relevant, what pattern it was generated from, and what role it has in the pattern. We use node attribute statements to compute the corresponding accuracy measures.

- **Pattern attribute learning quality**: we compare learned patterns against the true patterns that produced the data. The comparison metric is a value between 0 (worst match) and 1 (best match), obtained by converting the minimum network mapping mismatch (5) to the 0-1 mismatch score similar to the probability computation (19). We note that the data may contain structural patterns that were not intentionally used to generate the data network. This will occur because of establishing connections between true subnetworks as well as due to introduction of irrelevant nodes and links. While such occurrences will neither be frequent nor would exhibit the attribute consistency, we still might retrieve them as part of the learning process.

### 4.4. Experimental results on synthetic data

Figure 6 shows the performance assessment rates for learning and detecting hidden patterns and their node types from the 100 Monte-Carlo simulations using randomly generated attributed graphs with embedded primitive network structures. These results show that our pattern learning and matching model was able to find on average 80% of relevant nodes of the patterns hidden in the data, while producing less than 9% incorrect retrievals.

Two pattern structures presented challenges to our model, which was due to both network noise generation process and structural ambiguity of these patterns. First, a correct multi-path pattern structure was very difficult to learn because the generative model that initialized the structural primitives did not create an extensive set of alternative multi-path structures. Random network generation process added links to connect the true patterns as well as irrelevant links. This process created new paths in the data graph; as the result, many subnetworks in data network appeared conceptually similar to the multi-path pattern, but different in path length, number of multiple paths and the attributes on nodes and links. While several multi-path networks were detected in the data (which was structurally...
accurate), they were not the true ones. A similar problem was encountered with single-path networks. Those networks were easier to detect since the length of a single path was distinguishing this pattern from other node subsets.

Figure 7 shows the average attribute-based similarity (higher numbers in the matrix represent closer networks) between true hidden patterns and learned models. Most of the true patterns did match the learned hidden subnetworks. Moreover, we discovered patterns in the data structurally similar to true patterns (e.g., 2-level “tree” network was structurally similar to “star”), while some learned patterns were only partially similar to true patterns (“path” networks 2 and 3). During our experiments, learning “multi-path” patterns was not successful due to lack of strong constraints on the structure of that pattern (i.e., number and length of multiple paths between start and finish nodes). Also, learning “path” network proved challenging due to existence of multiple repetitive paths in the data network not matching the true networks in terms of attributes. As was shown in right bars in Figure 6, learning “clique” pattern was also challenging; we discovered this was due to convergence issues in LBP algorithm, and plan to address this in our future research. One of the key aspects affecting pattern learning was the model used to inject noise in the data network. By adding the links between hidden patterns and irrelevant data changed the structural consistency of the “observed” subnetworks and thus affected the learning and irrelevant data changed the structural consistency of data network. By adding the links between hidden patterns and irrelevant data changed the structural consistency of the “observed” subnetworks and thus affected the learning algorithm in situations the hidden patterns occurred rarely in the data. We plan to address these issues in our future research.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Similarity</th>
<th>Learned Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ring Source Sink Clique Path1 Path2 Path3 Grid Tree</td>
</tr>
<tr>
<td>True Patterns</td>
<td></td>
<td>0.00003 0.00004 0.9868 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Star</td>
<td>0.00003</td>
<td>0.99643 0.00004 0.98868 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Hub</td>
<td>0.00004</td>
<td>0.00000 0.00000 0.98578 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Clique</td>
<td>0.00000</td>
<td>0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Path1</td>
<td>0.00157</td>
<td>0.00000 0.00000 0.00197 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Path2</td>
<td>0.00012</td>
<td>0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Path3</td>
<td>0.00007</td>
<td>0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Grid</td>
<td>0.00010</td>
<td>0.00002 0.00001 0.00206 0.00008 0.00007 0.00006 0.00020 0.00002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Learned Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ring Source Sink Clique Path1 Path2 Path3 Grid Tree</td>
</tr>
<tr>
<td>True Patterns</td>
<td>0.00003 0.00004 0.9868 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Star</td>
<td>0.00003 0.99643 0.00004 0.98868 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Hub</td>
<td>0.00004 0.00000 0.98578 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Clique</td>
<td>0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Path1</td>
<td>0.00157 0.00000 0.00000 0.00197 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Path2</td>
<td>0.00012 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Path3</td>
<td>0.00007 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000</td>
</tr>
<tr>
<td>Grid</td>
<td>0.00010 0.00002 0.00001 0.00206 0.00008 0.00007 0.00006 0.00020 0.00002</td>
</tr>
</tbody>
</table>

4.5. Provided dataset and experimental design

Our customers provided us with a generic dataset that was event-centric in that all entity nodes were related to an event node. The entities consisted of individuals, their roles (e.g., aide, cell leader, field commander, member), groups, and events with which these entities were associated. Although the dataset was realistic in its structure and sparseness, it bore no correlation to real world events or individuals. The dataset contained thousands of nodes and links, and provided us with the opportunity to test developed tools in realistic situations, understand the features of the data that afford scalability, learn repetitive networks patterns in an unsupervised manner, and establish a validation process without access to ground truth.

First, we observed that the dataset was decomposable into many disconnected subnetworks. This provided us the opportunity to define a network training corpus, where the items in the corpus were disconnected subnetworks identified within our data. We used the unsupervised process described in section 3.1 to aggregate the disconnected subnetworks into clusters with similar structures and attributes, Some elements in a cluster became inputs for pattern learning, while some were reserved for testing. Using the EM algorithm described in section 3.3, we were able to learn attributed networks with highly discriminative features.

We divided the disconnected subnetworks randomly into training and testing subsets, and performed cross validation. After learning patterns from the training subset of the data, we performed network matching of the learned patterns on the test subset of the data. Understanding that the provided dataset contained temporal data and was somewhat richer than data normally available to an analyst, we designed an experiment that manipulated two variables: the amount of training data given to the network learning algorithm, and percentage of missing attributes that the algorithm needed to find. We then tested the following criteria:

- **Node attribute detection accuracy**: withdrawing a portion of the attributes from the testing data, we assessed the accuracy of node attribute inferences generated by our algorithm over the missing attribute(s).

- **Pattern learning efficiency**: we gave the algorithm only a portion of the disconnected networks from the training subset. We assessed the ability of our models to learn discriminative and generalizable patterns when the training data is scarce by computing how similar the learned pattern was to the centroid of a cluster of disconnected networks.

![Figure 7: Average similarity between true and learned patterns. The value closer to 1 represents that learned pattern is close to true network. Cells corresponding to expected true semantic matches are highlighted.](image)

![Figure 8: Experimental results on provided dataset. The x-axis represents a decision threshold to accept the inference statement. Results are shown for 80-20 and 50-50 cross-validation](image)
4.6. Experimental results on provided data

Figure 8 shows the accuracy measures for node attribute and pattern detection achieved in provided dataset, averaged across different missing attributes and 100 Monte-Carlo runs. The x-axis represents the decision threshold; increasing the decision threshold will result in accepting fewer inference statements about the pattern type/id for a subset of nodes and detecting the missing attribute with the observed node in social network. The decision is based on the probability of the attributed network matching. In these experiments, for 80-20 cross-validation (80% of data used for training the models and 20% for testing) and for acceptance threshold of 0.7, we achieved a recall of over 91% and precision of 78%. In other words, the model is able to find 86% of missing attributes while producing less than 22% incorrect attribute associations. Our model correctly discovered almost all patterns (recall above 95%) in provided dataset. The precision rate for pattern detection was low (65%) because several of learned patterns were similar (in terms of their attributes): while the network detection algorithm generated multiple inferences in this case, the ground truth was defined with only one pattern type per node group.

5. Conclusions and Future Research

In this paper, we described an approach to learn and detect network patterns and missing attributes in heterogeneous network data that may include different attributes and connection types. Our algorithms are able to learn patterns in supervised and semi-supervised manner, where the structural primitives for the patterns of interest are known and used to segment the data for further unsupervised training. Although the details are not described in this paper, our algorithms are able to scale to data networks with hundreds of thousands of nodes and links for in-memory implementation from seconds to minutes run time, and can process larger sets of data using Map-Reduce and the multi-core cloud.

In this paper, we reported the network learning and detection accuracies from two datasets. The first synthetic dataset with known ground truth about the patterns and attributes in the data was used for analyzing scalability and assessing the ability of the algorithm to find hidden patterns and types of the nodes in the data. We achieved on average 80% recall (were able to find 80% of the nodes from patterns hidden in the data) and 9% precision (less than 9% of the inferred node-pattern associations were false alarms). We similarly achieved high accuracy of pattern detection and node attribute inference on provided dataset (see Section 4.6). However, some pattern structures proved challenging to learn and find in the noisy data. We plan to address challenges of learning and matching patterns with particular structural properties, as well as dependency of accuracy of the complexity of the pattern structure and the noise properties, in our future research.

During this research, we encountered three challenges that warrant further investigation. First, network pattern matching must find multiple inexact matches, resulting in a complex optimization problem requiring finding several local optima. The modification of the Loopy Belief Propagation algorithm addresses some of these requirements; however, it has convergence issues for some of the network patterns. Currently, we are experimenting with additional stochastic search algorithms to improve the convergence of network matching solutions. Second, the network learning algorithm can proceed in unsupervised manner, but it requires that a data set be segmented either naturally, which is the case with many real-world problems in which the observations decompose into disconnected networks, or by identifying structural primitives that capture network topologies related to the search objectives. The second approach requires developing a network generative grammar, and our results, as well as the research in network motifs (e.g., Alon, 2007), indicate the feasibility of this approach. Finally, analysts must attach meaning to network patterns that are learned in an unsupervised manner. Our algorithms can return the frequency with which the pattern occurs in the data, and learned attributes provide the descriptive information about the patterns. However, defining the functional information about the network purely based on specific structural or high-level attribute statistics is still an open research question, and researchers argue that such high-level information does not provide sufficient insights into the function or intent of the network (Ingram, Stumpf, and Stark, 2006). Supplementing the structural and relational attributes with modeling the dynamics of the networks (i.e. the evolution of its state over time) promises more inferential power and is a significant further challenge for pattern learning and detection in datasets that contain high ambiguity and large amounts of irrelevant data.

6. Acknowledgements

DISTRIBUTION STATEMENT: this paper has been approved for public release; distribution is unlimited.

7. References


