Identifying Graphs From Noisy and Incomplete Data

Galileo Mark S. Namata Jr.
Department of Computer Science
University of Maryland
College Park, MD 20742 USA
namatag@cs.umd.edu

Lise Getoor
Department of Computer Science
University of Maryland
College Park, MD 20742 USA
getoor@cs.umd.edu

ABSTRACT

There is a growing wealth of data describing networks of various types, including social networks, physical networks such as transportation or communication networks, and biological networks. At the same time, there is a growing interest in analyzing these networks, in order to uncover general laws that govern their structure and evolution, and patterns and predictive models to develop better policies and practices. However, a fundamental challenge in dealing with this newly available observational data describing networks is that the data is often of dubious quality – it is noisy and incomplete – and before any analysis method can be applied, the data must be cleaned, and missing information inferred. In this paper, we introduce the notion of *graph identification*, which explicitly models the inference of a "cleaned" output network from a noisy input graph. It is this output network that is appropriate for further analysis. We present an illustrative example and use the example to explore the types of inferences involved in graph identification, as well as the challenges and issues involved in combining those inferences. We then present a simple, general approach to combining the inferences in graph identification and experimentally show the utility of our combined approach and how the performance of graph identification is sensitive to the inter-dependencies among these inferences.

Categories and Subject Descriptors

E.1 [**Data**]: Graphs and networks; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms

Algorithms

Keywords

data mining, statistical relational learning, social networks, classification, entity resolution, link prediction

1. INTRODUCTION

Data describing networks of various types, including social networks (e.g., friendship networks, affiliation networks), physical networks (e.g., transportation networks, computer networks), and biological networks (e.g., protein interaction networks, transcriptional regulatory networks) are increasingly becoming available. At the same time, there is a growing interest in analyzing these networks, in order to uncover general laws that govern their structure and evolution, and patterns and predictive models to develop better policies and practices. However, a fundamental challenge in dealing with this newly available observational data describing networks is that the data is often of dubious quality. Methods to directly acquire accurate and complete networks, if even possible, are often prohibitively expensive. Thus, more often data is gathered from indirect sources or high throughput experimental methods. This results in networks that are noisy and incomplete. If analysis is done directly on these networks it is likely to be biased and lead to faulty conclusions. Before any analysis method can be applied, the data must be cleaned, and missing information inferred. It is this output network that is appropriate for further analysis. In this paper, we introduce the notion of *graph identification*, which explicitly models the inference of a "cleaned" output network from a noisy input graph. In section 2, we present an illustrative example and use the example to explore the types of inferences involved in graph identification, as well as the challenges and issues involved in combining those inferences, in section 3. We then present a simple, general approach to combining the inferences for graph identification in section 4. We propose a novel synthetic data generator to evaluate this approach in section 5 and experimentally show how the performance of graph identification is sensitive to the inter-dependencies among these inferences. We present related work in section 6 and discuss our conclusions and future work in section 7.

2. MOTIVATING EXAMPLE

Suppose we wish to understand and analyze the social network of a large organization. Specifically, we wish to explore the network which identifies the individuals in the organization, the close friendships between those individuals, and the roles of the individuals. For large organizations, it may be very difficult, if possible, to gather such a network directly. What may instead be available for such an organization are the archived email communications. Using these communications, we can construct a communication network where nodes represent email addresses, edges represent a communi-

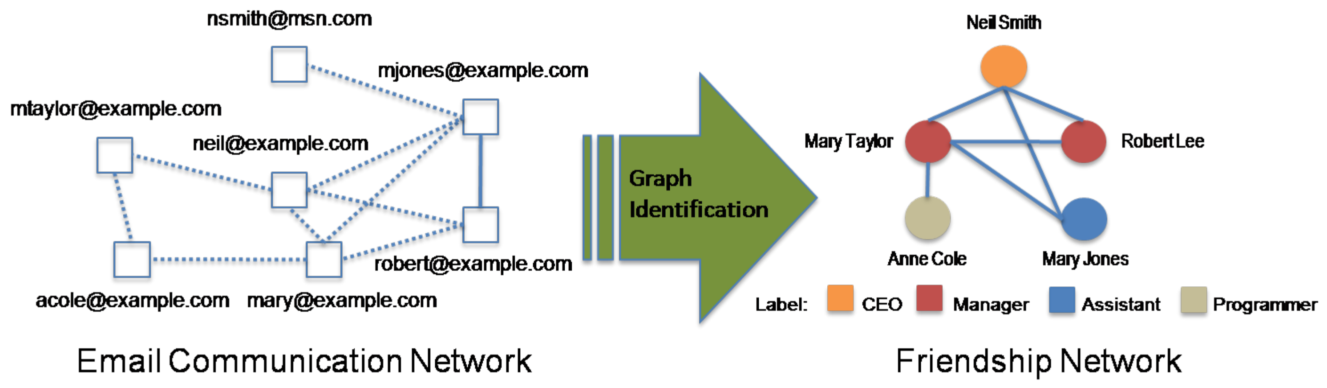

Figure 1: An illustration of the motivating example in section 2. The network on the right, the friendship network, is the output network appropriate for analysis where the nodes represent people, the edges represent friendships, and the color represent company roles. The network on the left is the network which is the available but noisy and incomplete network, the email communication network, where the nodes represent email addresses and edges represent communication between email addresses. Graph identification explicitly models the "cleaned" output friendship network from the noisy input email communication network.

cation between the email addresses, and attributes for these nodes and edges may include traffic statistics (e.g., frequency of communications) and content (e.g., presence of a word or phrase in an email). This available network, however, is noisy and incomplete for our analysis. To illustrate, consider the small example networks shown in Figure 1. The nodes in the communication network do not accurately reflect the individuals in the organization. If we perform analysis, substituting email address nodes for people nodes, even a simple statistic, like the number of individuals, would be inflated by the fact that people have multiple email addresses (i.e., *mary@example.com* and *mtaylor@example.com* both belong to Mary Taylor). Moreover, the communication network links are not the same as the social relationships between individuals (i.e., email communications exist between *robert@example.com* and *mjones@example.com* although their users, Robert Lee and Mary Jones, are not close friends) nor the attributes for our analysis (i.e., the email addresses are not explicitly annotated with roles). Although the communication network is not directly appropriate for our task, we can use the information in the communication network to infer the social network that we would like to analyze. This requires identifying the people and the correspondence of email addresses to people (these may be email addresses which have similar writing and communication patterns), friends (who are likely to email each other regarding social events), and their roles (reflected in the content of communications and/or with whom they communicate). We refer to this process, from the available noisy network to the network appropriate for our analysis as *graph identification*.

3. DEFINITION

Graph identification involves identifying the output graph from a given input graph, and involves constructing the mapping from the input to the output graph. The *output graph* ($\mathcal{G}_\mathcal{O}$) is the graph that is appropriate for further analysis. Ideally, the output graph can be acquired directly. In most cases, however, the output would be too difficult or expensive to directly acquire. What may be available, instead, is another graph which reflects the output graph but is too noisy and incomplete to directly use for our analysis. We refer to this available graph as the *input graph* ($\mathcal{G}_\mathcal{I}$). Given these graphs, we define *graph identification* as the general problem of inferring the desired output graph given a noisy input graph. The process of doing graph identification results in a mapping from elements (i.e., nodes, edges, and/or attributes) of the input graph to nodes in the output graph, the identification, or prediction, of edges that exist in the output graph, and mapping attributes of the nodes and edges in the output graph to values.

By its nature, graph identification is domain dependent. The specific inferences needed to perform graph identification will vary based on what the input and output graphs are and how the those graphs are related. In the scenario presented in section 2, for example, we are interested in the friendship network of individuals in a company as our output graph while our available input graph is the company's email communication network. We know, given our domain, that the mapping from the elements of the input graph to the nodes of the output graph is a many-to-one mapping from the email address nodes of the communication graph to the person nodes of the social network. Specifically, all individuals in our social network likely own at least one email address in the communication network and are likely to have multiple. Thus, we can identify the set of person nodes by mapping all email address nodes, which belong to the same person, to a single person node in the output graph. The problem of creating this type of mapping is commonly referred to as *entity resolution*[1]. Once we define the mapping between the nodes of the input and output graph, we can identify the set of edges that exist between the nodes of the output graph. In our domain, we can use the domain knowledge that people who are friends likely email each other, likely have similar attributes (e.g., interests) and likely communicate using terms common to that type of relationship (e.g., invitations to social events). We can thus use this knowledge to construct an edge prediction function for friendship edges which we can then apply over all people nodes in the output graph. The problem of defining (or learning) this existence function is commonly referred to as *link prediction*[10]. Finally, we know that individuals who fill a particular role are likely to discuss their role in their communications. We also know that social networks are often homophilic such that individuals who fill the same role are likely to be friends. We can use this knowledge to define a mapping for the attribute value of a person's role. The problem of predicting the values of these types of attributes is commonly referred to as *collective classification*[13]. Consequently, in our example in section 2, graph identification is performed by the application of entity resolution, link prediction, and collective classification over the input graph.

Although the specific inferences in graph identification is domain dependent, at its core graph identification is still the problem of performing three general inferences: the node mapping, edge existence function, and attribute value mapping. An important aspect of graph identification is how these three inferences should be applied and how they interact. One method for doing graph identification involves applying a collection of local predictors for the three inferences (e.g., applying entity resolution (ER), link prediction (LP), and collective classification (CC) models). Another method for doing graph identification involves defining a joint model, and extracting the output graph with the most likely configuration. For the rest of this paper, we will explore the former method. We will examine a simple, general way the local predictors can be combined, discuss issues involved using this method of combination, and show, in general, how inter-dependent these inferences are.

4. PIPELINE GRAPH IDENTIFICATION

One method to perform graph identification is to apply a collection of local predictors. The benefit of this method is that we can use any previously defined models for each of the individual predictors. For our scenario in section 2, for example, this method allows us to apply a previously learned local entity resolution, link prediction, and collective classification models. When applying local predictors, however, we have to address the additional challenge of how to combine these predictors. First, we need to consider what order the predictors are applied. This is important for any approaches which focus on a single pass, sequential application of the predictors. In these approaches, order defines the set of previous predictions a predictor can use for its inference. In our motivating example, in particular, application of link prediction prior to collective classification provides the collective classification models predicted links to define relational features. Conversely, application of collective classification prior to link prediction allows the link prediction to use the labels of incident nodes in its features. Another issue to consider is when and how the predictions should be committed and information shared between the predictors. Predictions can be applied all at once or be applied iteratively. Since our predictors are strongly inter-dependent it may be beneficial to perform each task partially and exchange information more frequently. We may also want to vary how much information is shared between predictors. Committing only the most likely predictions at each iteration[12] may mitigate the propagation of incorrect information between the predictors. Predictions can also be shared as hard or soft predictions. Depending on the predictors used for each inference, there may be information about the probability or confidence of the predictions that maybe useful to share among the predictors. Graph identification also provides challenges in that there are two graphs whose information we can use in our predictors. When predicting over the nodes of the output graph, we not only have the attributes and edges of the output graph to use, we also have the additional information provided by the parts of the input graph mapped to those nodes. For example, the words used in email communications of a subset of email addresses may be used to infer the label of the person they belong to. We need to figure out how best to combine and exploit the information in both graphs during both learning and inference. Finally, we need to consider constraints between the predictions. In our motivating example, we can have constraints like all friends must have shared email communications or that individuals with a certain role must be friends. We must consider how and when to enforce these constraints in the final output graph.

In this paper, we present and analyze the most direct way of applying these inferences, in a pipeline [17]. In the *pipeline approach*, we apply the predictors one at a time and in sequence where all predictions are committed as hard predictions at the end of each turn and are available for use in the next predictor in the pipeline. Referring to the example in section 2, this means applying entity resolution, link prediction, and a collective classification in turn, as defined in section 4.1.

4.1 Example Pipeline

Let $\mathcal{G}_{\mathcal{I}}(\mathcal{V}_{\mathcal{I}}, \mathcal{E}_{\mathcal{I}})$ represent the input graph, and $\mathcal{G}_{\mathcal{O}}(\mathcal{V}_{\mathcal{O}}, \mathcal{E}_{\mathcal{O}})$ represent the output graph where $\mathcal{V}_{\mathcal{I}}$ and $\mathcal{V}_{\mathcal{O}}$ are the sets of nodes and $\mathcal{E}_{\mathcal{I}}$ and $\mathcal{E}_{\mathcal{O}}$ are the sets of edges for the corresponding graphs. For all input graph nodes, $v_I \in \mathcal{V}_I$, let $entity(v_I) = v_O$ denote the unknown output graph node, $v_O \in \mathcal{V}_O$, which v_I corresponds to (e.g., the person who owns a given email address). In this instance of the pipeline approach, we begin by applying an entity resolution model, ER_{model}, to create a set of disjoint clusters, $c \in C$, of input graph nodes such that a pair of input graph nodes, v_I^1, v_I^2, are in the same cluster if $entity(v_I^1) = entity(v_I^2)$ and in different clusters if $entity(v_I^1) \neq entity(v_I^2)$. We then create a node in the output graph mapped from each cluster resulting in the full set of output graph nodes. Next, we consider all pairs of the newly created output graph nodes, $v_O^j, v_O^k \in V_O$, and apply the LP_{model} over these pairs. The LP_{model} defines an indicator function such that:

$$LP_{model}(v_O^j, v_O^k) = \begin{cases} 1 & \text{if an edge should exists between } v_O^j \text{ and } v_O^k, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We create an output graph edge between all output graph nodes where $LP_{model}(v_O^j, v_O^k) = 1$. Finally, we apply the CC_{model} over the output graph nodes, $v_O \in \mathcal{V}_O$ in order to predict the value of the label attribute, $v_O.A$. Given a set of possible values, L, of the target attribute, A, the CC_{model} defines a function, $CC_{model}(v_O) = l$ where $l \in L$. For all $v_O \in \mathcal{V}_O$, we assign $v_O.A = CC_{model}(v_O)$.

To further illustrate the inferences involved in this approach, we now present a specific instance of the approach over the motivating problem. In this instance, we assume a supervised scenario where we have test and train pairs of input and output graphs, as well as the mapping between

Algorithm 1 Example Pipeline Graph Identification

```
Input: $\mathcal{G}_I$, $ER_{model}$, $LP_{model}$, $CC_{model}$
Output: $\mathcal{G}_O$
1: Apply $ER_{model}$ on $\mathcal{G}_I$ (i.e., cluster nodes)
2: Create nodes $\mathcal{V}_O \in \mathcal{G}_O$ mapped from the clusters
3: Apply $LP_{model}$ over possible edges between $\mathcal{V}_O$
   (i.e., apply edge existence)
4: Create edges $\mathcal{E}_O \in \mathcal{G}_O$ for existing edges
5: Apply $CC_{model}$ on $\mathcal{V}_O$ and $\mathcal{E}_O$ (i.e., predict labels)
6: Set labels for all $\mathcal{V}_O$ and $\mathcal{E}_O$ to their predicted value
```

them. We also use three commonly used predictors and sets of features for the ER, LP, and CC models. For entity resolution, we use collective relational clustering (CRC) [1] which iteratively creates subsets of references, each corresponding to a single entity, using a weighted combination of the similarity of local attributes and neighborhoods of the references. In CRC, for a given pair of nodes we use the normalized similarity between the node entity attributes of the input graph nodes for the feature similarity (e.g., similarity of the email address strings[6]) and the Jaccard-Coefficient similarity[1] of their neighborhoods (e.g., email address nodes adjacent to a given node with a communication edge) for the relational similarity. We vary the α parameter, controlling the weighting between the local and neighborhood similarities, and threshold parameter, the minimum similarity of two reference clusters predicted to refer to the same entity, in CRC based on the training graphs and the mapping between them. For link prediction and collective classification, we use the aggregate values of input graph nodes mapped to each output graph node. We take the aggregate by setting the value of an attribute a for an output graph node with the mapping $v_o = \{v_i^1, v_i^2, ..., v_i^j\}$ as:

$$v_o.a = mode(v_i^1.a, v_i^2.a, ..., v_i^j.a) \tag{2}$$

where *mode* takes the most common value of an attribute a among the mapped input graph nodes. In link prediction, we use these aggregates to create a feature which measures the percent similarity of the aggregate values of a given output graph node pair. Specifically, given two output graph nodes and their mapped input graph node subsets, $v_o^1 = \{v_i^{11}, v_i^{12}, ..., v_i^{1j}\}$ and $v_o^2 = \{v_i^{21}, v_i^{22}, ..., v_i^{2k}\}$, we compute:

$$similarity(v_o^1, v_o^2) = \frac{\sum_{a \in A} \delta(v_o^1.a, v_o^2.a)}{|A|} \tag{3}$$

where A is the set of all attributes of the input graph nodes, $|A|$ is the set size of A, and $\delta(x, y)$ is an indicator function which returns 1 if $x = y$ and 0 otherwise. We use this similarity as a feature in logistic regression[20] model to predict whether or not an edge exists. Finally, for collective classification, we use the iterative classification algorithm (ICA)[11], using logistic regression for the bootstrap and relational classifiers. For a given output graph node, we use aggregates of the attributes over the mapped input graph nodes in the bootstrap classifier (i.e., $\forall a \in A, v_o.a$). For the relational classifier, we use these aggregates, as well as the percent of neighboring nodes (i.e., nodes adjacent to a given node with a friendship edge) which have a specific label.

5. EXPERIMENT

In order to study the pipeline approach of graph identification, we experiment to see the strengths and weaknesses this approach has for different types of networks. First, we developed a novel synthetic data generator which allows us to create an input and output graph modeled after the communication and social networks presented in section 2. In our generator, we can control the reference, link existence, and label ambiguity in the inference so that we can vary the ability of each predictor, ER, LP, and CC, respectively, to make an accurate inference. We generate networks with different ambiguity levels (Low, Med, High) for each type of ambiguity. We provide specific details of the synthetic data generator in section 5.1. We perform graph identification by applying Algorithm 1 and using the models and features described in section 4.1. We train predictors, for all combinations of ambiguity (27 in total), on one graph and test on another graph to compute the F1 performance, averaged over six runs, for each predictor. We also explored the effects of order, as discussed in section 4, by using a variant which exchanges the order we apply LP and CC (lines 4 and 5 in Algorithm 1). For both versions, we also explored the challenge of how to make use of both the input and output graphs by varying how the LP and CC were trained. We note that in this example, there are two ways to train LP and CC. The first is to train the models directly over the edges and labels of the training output graph. Another way to train LP and CC is to use the correspondence between the output and input graph to transfer equivalent edges and labels to the input graph. Specifically, an equivalent edge is created between two input graph nodes if the known corresponding output graph nodes also share an edge. Similarly, an input graph node is assigned a label if the known corresponding output graph node has that label. We then train the models over the modified training input graph.

5.1 Synthetic Data Generator

To evaluate the performance improvements of our optimizations, we developed a novel synthetic data generator that creates a noisy network with ambiguous references which need to be merged to entities, missing labels which need to be classified, missing edges which need to be predicted, and the graph structure and attributes commonly used for those types of inferences. The graph and the attributes created by this synthetic data generator is modeled after the motivating problem presented in the paper where the desired output graph is a social network where nodes are people, edges are close friendships between those people, and attributes represent a trait of that person (e.g., role). Intuitively, the generator works by creating a synthetic output graph which mimics the structure and attributes of real world social networks. The generator then creates communication network input graph from the social network output graph by adding different types of noise common to these types of network. The algorithm for the synthetic data generator is shown in Algorithm 2.

The synthetic data generator begins by creating the structure of the network (i.e., the set of nodes and edges of the output graph). A number of network generation models have been proposed which create networks which exhibit properties, observed in many real world networks. For our experiments, we implemented the widely used Forest Fire generation model [9] which models many of these proper-

Algorithm 2 Synthetic Data Generator

```
Output: Output Graph ($\mathcal{G}_O$), Input Graph ($\mathcal{G}_I$)
 1: $\mathcal{G}_{temp} \Leftarrow$ Generate network structure
 2: Add node labels to $\mathcal{G}_{temp}$
 3: Add node attributes based on node labels to $\mathcal{G}_{temp}$
 4: Add node attributes based on neighboring nodes to $\mathcal{G}_{temp}$
 5: Add node entity attributes to $\mathcal{G}_{temp}$
 6: $\mathcal{G}_O \Leftarrow \mathcal{G}_{temp}$ {Set clean graph as output graph}
 7: $\mathcal{G}_{temp} \Leftarrow$ Create ambiguous references for nodes in $\mathcal{G}_{temp}$
 8: Remove node labels from $\mathcal{G}_{temp}$
 9: Randomly change values of attributes from $\mathcal{G}_{temp}$
10: Randomly remove edges from $\mathcal{G}_{temp}$
11: Add random edges between some node pairs to $\mathcal{G}_{temp}$
12: $\mathcal{G}_I \Leftarrow \mathcal{G}_{temp}$ {Set noisy graph as input graph}
```

ties including heavy tailed degree distribution, "small world" phenomenon, and densification over time. We used a *forward burn probability* of 0.4 and a *backward burn probability* of 0.2. This creates the output graph nodes (people nodes) and output graph edges (friendship edges).

Once the initial network structure is generated, we add three sets of attributes to the nodes corresponding to the three types of inferences we will perform on the graph. The first set is for use with collective classification and includes the labels and attributes based on those labels. We use the label generation method described in [14] (2 labels, with 1/10 of the graph initially labeled randomly) to create the "role" label of the people nodes where "role" has a high positive autocorrelation (i.e., people who are friends like have the same role). We then create binary attributes based on those labels using the method described in [3] (5 attributes per label where secondary probability is set to 0.45 while the primary probability is varied to control *label ambiguity*). The second set of attributes is used for link prediction and consist of between 1 and 100 attributes (varied to control *link existence ambiguity*) generated using the method described in [14]. These attributes were generated for link prediction with the intuition that nodes with similar attributes are likely to share an edge. The last set of attributes are used for entity resolution and represent attributes that imply, non-uniquely, the entity it refers to (e.g., first name references non-uniquely imply who the individuals are as multiple individuals may have the same first name). To generate these attributes, we use the method described in [1] and vary p_a to control the *reference ambiguity.* The resulting network is our synthetic output graph (friendship network).

We create an input graph from our output graph by creating a noisy version of the output graph. We add noise in four ways. First, we add ambiguous references (email addresses) to the input graph by adding a random number of nodes (between 1 and 3) for a percentage of the nodes in the graph (25% of the original nodes). Each input graph node initially has the same attributes and labels as the corresponding node in the output graph and we also create edges similar to those of the output graph by ensuring all input graph nodes have an edge "equivalent" to the edges of the corresponding output graph nodes. Equivalent edges are created by adding at least one edge from an input graph node, corresponding to a node v_o^j of the output graph, to a input graph node, corresponding to an output graph node v_o^k, if v_o^j and v_o^k share an edge. Once the reference nodes are generated, we add noise to the attributes of those nodes by removing the "role" labels of all the nodes and randomly changing, as appropriate, the values of the other attributes. Finally, we add edge noise to the graph by randomly removing a percent of the existing edges (20% of the current number of edges) and adding edges between randomly selected pairs of nodes in the graph (adding 50% more edges) where the resulting edges are our communication edges. The resulting noisy network is our synthetic input graph (communication network).

5.2 Results and Discussion

We evaluate the performance of the ER, LP, and CC models using the average F1 performance of each predictor over the different networks. For entity resolution, we use the method of calculating F1 over ER predictions as described in [1] where we consider all possible pairs of input graph nodes and whether or not each pair is accurately mapped to the same or different output graph node. For link prediction and object classification, however, we cannot compute the F1 performance directly over the nodes of the predicted output graph because the set of output graph nodes will vary based on the entity resolution performance. We address this by evaluating link prediction and object classification over predictions mapped onto the input graph. A predicted edge is mapped between nodes in the input graph if the predicted output graph nodes of the two input graph nodes have a predicted edge between them. Moreover, the predicted edge between two nodes in the input graph is a true positive edge if an edge exists between the mapped true output graph nodes and a false positive edge otherwise. Similarly, the predicted label of a node in the input graph is the label of the node it is mapped to in the predicted output graph and the true label of that node is the label of the node it is mapped to in the true output graph.

The results are presented in Table 1 and Table 2. For clarity, we only show results where we vary one type of ambiguity while holding the others at medium. First, in general, we see that good performance of predictors early in the pipeline result in improved performance of later predictors. In fact, the best performances are seen when the entity ambiguity is low resulting in ER performing well. Good ER performance results in a more accurate set of person nodes, and thus a more accurate set of mappings for use by the features of LP and CC. We see the same trend in CC performance when link existence ambiguity is low in Table 1. LP performance improves which results in more accurate links for the relational features used in CC. Note though the improvement in LP performance does not affect ER and the improvement in CC performance does not affect LP and ER. This is a weakness in the pipeline approach in that the flow of information is only one way. Ideally, in graph identification, the models and features used by the predictors should be able to make use of predictions from all other predictors to improve its performance (e.g., use labels and predicted edges in the ER feature and relational similarity, use predicted labels in LP relational features). An obvious extension to address this weakness is an iterative pipeline approach where the pipeline is repeatedly applied over the network. We performed an initial study of the iterative pipeline approach but the results were inconclusive indicating a naive iterative approach may not be enough. This is a subject of future work. Next, we note that when predictors early in the pipeline perform poorly, the effect is reduced performance for all predictors later in the pipeline. In fact, the resulting reduction in performance can be drastic as shown when we increase reference ambiguity. Although link existence and label ambiguity is held constant, poor performance by ER early in the pipeline results in substantial drop in LP and CC performance as both are forced to make predictions over people nodes whose mapped email address nodes inaccurately and incompletely reflect a person and that person node's friendships and attributes. Thus, in graph identification, we need to be aware of what the expected performance of each predictor is and how each predictor will impact the other predictors in the overall inference.

Comparing the results from the two ways of training our LP and CC models, shown in Table 1, we see a general trend where LP performs better when trained over the output graph. On the other hand, we see that CC generally performs better when trained over the input graph. The improvement in both cases demonstrates two things. First, the improvement shows the importance of understanding and exploiting information in both the input and output graphs in the inferences. Second, the differences over the two sets of results, where the output graph predicted over models trained over the output graph has better LP performance but worse CC performance than the alternative, demonstrate the complexity of the inference interactions, as well as the difficulty in comparing the quality of one predicted output graph from another.

In the comparison between approaches varying the order the inferences are performed, shown in Table 2, we see that order can substantially impact the overall performance. Both LP and CC performance declines when applying CC prior to LP. The decline is particularly noticeable for CC performance when the label ambiguity is high. When label ambiguity is high, the classifier used must rely on relational features more. However, given that the edges LP predicts, over which the nodes are homophilic, are not available the classifier is unable to accurately predict the label. This in turn affects the performance of LP which uses the predicted label in its predictions. Although the inter-dependence of these inference have the potential to positively affect their performance as a whole, the inverse is also true. In graph identification, the impact of a poorly performing predictor must be mitigated when combining the inferences.

6. RELATED WORK

There have been a number of machine learning problems defined over network data[7]. In this work, we discussed how some of these problems (i.e., entity resolution, link prediction, and entity resolution) correspond to types of inferences involved in graph identification. We note however, that these problems only infer specific parts of a graph. They do not, individually, address all the inferences involved in inferring both the structure and attributes of a whole graph. There are previous work which infer more of the graph by combining many of these inferences. Many attempts perform pairwise combinations of these inferences such as combining model link prediction and collective classification[4, 5, 19] or combining entity resolution and collective classification[2]. There have also been work on creating a general framework which allows for a general combination of these problems[16, 17, 8, 15, 18]. To our knowledge, none of the previous work have explored the benefits, issues, and challenges in performing this combination to explicitly infer a full graph.

7. CONCLUSION AND FUTURE WORK

In this paper, we introduce the notion of graph identification. We discuss the types of inferences involved in graph identification and explore those inferences using an illustrative example problem. We discuss the types of approaches applicable for graph identification and explored one approach which uses a combination of local predictors. We discuss the issues we must consider when using this approach and present a simple, general approach to applying these inferences in graph identification. We then experimentally evaluate the general approach and show the importance of the inter-dependence of these inferences is in accurately predicting the output graph. In future work, we plan to further explore these inter-dependencies by exploring alternate ways to combine local predictors and comparing those methods to a full joint approach to graph identification. We are also exploring methods for comparing the quality of two predicted output graphs. and are in the process of collecting real world datasets we can evaluate our approaches over.

8. ACKNOWLEDGMENTS

We are grateful to Mustafa Bilgic for help with the synthetic data generator. The work was supported by NSF Grant No. 0746930.

9. REFERENCES

[1] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1:1–36, 2007.

[2] I. Bhattacharya, S. Godbole, and S. Joshi. Structured entity identification and document categorization: two tasks with one joint model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 25–33, 2008.

[3] M. Bilgic and L. Getoor. Effective label acquisition for collective classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 43–51, 2008.

[4] M. Bilgic, G. M. Namata, and L. Getoor. Combining collective classification and link prediction. In *ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, pages 381–386, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Y. Choi, E. Breck, and C. Cardie. Joint extraction of entities and relations for opinion recognition. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 431–439. Association for Computational Linguistics, 2006.

[6] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, August 2003.

[7] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations Newsletter*, 7:3–12, 2005.

Table 1: Average and standard deviation of F1 performance of ER, LP, and CC in the pipeline approach of various levels of ambiguity comparing two ways of training the data over the output and input graphs. In general, we see improvement in performance of a predictor improves performance of predictors later in the pipeline. However, improvement of predictors later in the pipeline do not impact the performance of predictors early in the pipeline. With the two w ays of training, we see that in training over either the output graph or the modified input graph can affect overall performance.

		Train Over Output Graph			Train Over Input Graph		
Ambiguity	Level	*ER (F1)*	*LP (F1)*	*CC (F1)*	*ER (F1)*	*LP (F1)*	*CC (F1)*
Reference	Low	0.758 ± 0.093	0.554 ± 0.089	0.715 ± 0.065	0.758 ± 0.093	0.547 ± 0.110	0.750 ± 0.039
	Med	0.627 ± 0.169	0.439 ± 0.148	0.677 ± 0.078	0.627 ± 0.169	0.427 ± 0.170	0.719 ± 0.057
	High	0.213 ± 0.033	0.102 ± 0.034	0.517 ± 0.110	0.213 ± 0.033	0.078 ± 0.026	0.447 ± 0.189
Link Existence	Low	0.627 ± 0.169	0.500 ± 0.161	0.705 ± 0.066	0.627 ± 0.169	0.467 ± 0.166	0.729 ± 0.071
	Med	0.627 ± 0.169	0.439 ± 0.148	0.677 ± 0.078	0.627 ± 0.169	0.427 ± 0.170	0.719 ± 0.057
	High	0.627 ± 0.169	0.101 ± 0.057	0.653 ± 0.052	0.627 ± 0.169	0.301 ± 0.135	0.599 ± 0.181
Label	Low	0.627 ± 0.169	0.439 ± 0.148	0.797 ± 0.069	0.627 ± 0.169	0.427 ± 0.170	0.811 ± 0.055
	Med	0.627 ± 0.169	0.439 ± 0.148	0.677 ± 0.078	0.627 ± 0.169	0.427 ± 0.170	0.719 ± 0.057
	High	0.627 ± 0.169	0.439 ± 0.148	0.517 ± 0.085	0.627 ± 0.169	0.427 ± 0.170	0.556 ± 0.045

Table 2: Average and standard deviation of F1 performance of ER, LP, and CC in the pipeline approach of various levels of ambiguity comparing how the order the inferences are run affects the performance. In the results, we see that we have better performance in collective classification when we order the predictors as ER, LP, and CC. CC is able to make use of the LP edges in this case while CC must rely only on local attributes and noisy communication edges in the other case.

		Order: ER, LP, CC			Order: ER, CC, LP		
Ambiguity	Level	*ER (F1)*	*LP (F1)*	*CC (F1)*	*ER (F1)*	*LP (F1)*	*CC (F1)*
Reference	Low	0.758 ± 0.093	0.547 ± 0.110	0.750 ± 0.039	0.758 ± 0.093	0.545 ± 0.109	0.514 ± 0.294
	Med	0.627 ± 0.169	0.427 ± 0.170	0.719 ± 0.057	0.627 ± 0.169	0.425 ± 0.169	0.585 ± 0.187
	High	0.213 ± 0.033	0.078 ± 0.026	0.447 ± 0.189	0.213 ± 0.033	0.078 ± 0.026	0.327 ± 0.322
Link Existence	Low	0.627 ± 0.169	0.467 ± 0.166	0.729 ± 0.071	0.627 ± 0.169	0.467 ± 0.166	0.585 ± 0.187
	Med	0.627 ± 0.169	0.427 ± 0.170	0.719 ± 0.057	0.627 ± 0.169	0.425 ± 0.169	0.585 ± 0.187
	High	0.627 ± 0.169	0.301 ± 0.135	0.599 ± 0.181	0.627 ± 0.169	0.294 ± 0.130	0.585 ± 0.187
Label	Low	0.627 ± 0.169	0.427 ± 0.170	0.811 ± 0.055	0.627 ± 0.169	0.425 ± 0.169	0.780 ± 0.053
	Med	0.627 ± 0.169	0.427 ± 0.170	0.719 ± 0.057	0.627 ± 0.169	0.425 ± 0.169	0.585 ± 0.187
	High	0.627 ± 0.169	0.427 ± 0.170	0.556 ± 0.045	0.627 ± 0.169	0.426 ± 0.170	0.326 ± 0.322

[8] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of link structure. *Machine Learning*, 3:679–707, 2003.

[9] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery Data*, 1(1):2, 2007.

[10] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Intl. Conf. on Information and Knowledge Management*, 2003.

[11] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the International Conference on Machine Learning*, 2003.

[12] L. McDowell, K. M. Gupta, and D. W. Aha. Cautious inference in collective classification. In *AAAI*, pages 596–601, 2007.

[13] G. M. Namata, P. Sen, M. Bilgic, and L. Getoor. Collective classification for text classification. In M. Sahami and A. Srivastava, editors, *Text Mining: Classification, Clustering, and Applications.* Taylor and Francis Group, 2009.

[14] M. J. Rattigan, M. Maier, and D. Jensen. Exploiting network structure for active inference in collective classification. Technical report, University of Massachusetts Amherst, 2007.

[15] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[16] D. Roth, K. Small, and I. Titov. Sequential learning of classifiers for structured prediction problems. In *Artificial Intelligence STAT*, April 2009.

[17] D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In *Proc. of CoNLL-2004*, pages 1–8. Boston, MA, USA, 2004.

[18] B. Taskar, A. Pieter, and D. Koller. Discriminative probabilistic models for relational data. In *Conf. on Uncertainty in Artificial Intelligence*, 2002.

[19] B. Taskar, M.-F. Wong, P. Abbeel, and D. Koller. Link prediction in relational data. In *Advances in Neural Information Processing Systems*, 2003.

[20] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, October 1999.