GRDB: A System for Declarative and Interactive Analysis of Noisy Information Networks

Walaa Eldin Moustafa, Hui Miao, Amol Deshpande, Lise Getoor
Department of Computer Science, University of Maryland, USA
{walaa, hui, amol, getoor}@cs.umd.edu

ABSTRACT

There is a growing interest in methods for analyzing data describing networks of all types, including biological, physical, social, and scientific collaboration networks. Typically the data describing these networks is observational, and thus noisy and incomplete; it is often at the wrong level of fidelity and abstraction for meaningful data analysis. This demonstration presents GRDB, a system that enables data analysts to write declarative programs to specify and combine different network data cleaning tasks, visualize the output, and engage in the process of decision review and correction if necessary. The declarative interface of GRDB makes it very easy to quickly write analysis tasks and execute them over data, while the visual component facilitates debugging the program and performing fine grained corrections.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query languages*; H.2.8 [**Database Management**]: Database Applications—*Data Mining*

Keywords

Social Network Analysis, Graph Data, Datalog

1. INTRODUCTION

In today's world, networks abound. Examples include social networks, communication networks, financial transaction networks, gene regulatory networks, disease transmission networks, ecological food networks, sensor networks, and more. There is a growing interest in real-time methods for analyzing such network data for scientific discovery, anomaly detection, vulnerability prediction, and assessing the potential impact of interventions. Although observational data describing these networks can often times be obtained, an inherent problem with much of this data is that it is noisy and incomplete, and at the wrong level of fidelity and abstraction for meaningful data analysis. Thus there is a need for methods which extract and infer "clean" annotated networks from noisy observational network data. This involves inferring missing attribute values **(attribute prediction)**, adding missing links and removing spurious links between the nodes **(link prediction)**, and eliminating duplicate nodes **(entity resolution)**.

While methods have been proposed for doing each of these extractions/inferences in isolation, there has been little work on fully integrated approaches. The little work that has been done has been ad hoc, domain-specific, and typically performed outside a declarative data management framework. This makes it cumbersome to store and compare results of different approaches, or to handle dynamic updates to the underlying observation network. Furthermore, the sizes of real-world networks are growing at a rapid pace, with networks with millions of nodes and edges becoming ubiquitous. To support analysis and cleaning of these networks, a framework is needed for efficiently storing, managing, and analyzing such large, dynamic network data.

In this demonstration, we present GRDB, a system that enables efficient, declarative, visual, and interactive analysis and cleaning of large-scale information networks. Our goal is two-fold. First, we wish to provide a declarative framework for specifying common operations required in cleaning and extracting networks, a mechanism for combining them in various ways, and an implementation for efficiently applying them to a large observational network. Second, we wish to allow the system users, or data analysts, to inspect the system decisions visually, interact with them, and correct them if necessary. The four main challenges in building such a framework are: (a) network analysis is heavily dependent on the actual graph structure and typically requires traversal of the node neighborhoods and computation of structural features; (b) most network analysis techniques are inherently *iterative*, and require repeated passes over the graph; (c) network cleaning and analysis often needs to be "collective" (where a decision in one part of the network affects the information flow in other parts of the network); and (d) since decisions are collective, errors in some decisions might affect further future decisions, and hence, it becomes necessary to avoid wrong predictions, especially in early iterations, to keep them from propagating.

The key to our approach is to *decouple* the graph traversal operations from the modification operations; the traversal operations are typically computationally expensive, especially for large disk-resident graphs. We present a declarative analysis language based on *Datalog*, and show how it can be used to cleanly achieve such decoupling. This decoupling enables us to develop a framework for declarative analysis over large networks, and facilitates efficient execution, by allowing us to push much of the computation inside a database system. Further, the declarative framework allows us to efficiently incorporate, and propagate through the analysis task, dynamic updates to the network data. In addition, we involve data analysts into the decision loop, present them with de-

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

cisions before applying them to the data, and allow them to change the predictions before proceeding with further decisions. We have implemented a prototype system called GRDB that supports our declarative framework with user feedback. Our preliminary results illustrate the computational and usability advantages of our system.

2. NETWORK INFERENCE OPERATIONS

The operations that are commonly required in cleaning network data include filling in missing information, correcting inaccurate information, and consolidating and reconciling redundant information. In this work, we frame these operations as *prediction* problems, and use machine learning classification algorithms to perform them. The predictions are made at the granularity of a *prediction element*, which may be a node whose label is to be predicted, or a pair of nodes to be linked, etc., depending on the task. We make a distinction between *local* classification algorithms that make predictions based only on the attributes of the prediction element, and *collective* classification algorithms where predictions for a prediction element can depend on the output of other classifiers corresponding to other prediction elements. The prediction problems supported can be broadly classified into three categories:

Attribute prediction: The goal here is to predict the value for an attribute of a node. Predictions can be made based on the values of other attributes of the node (local classification) or based on the neighbors' predicted attribute values (collective attribute prediction). The underlying assumption in attribute prediction is that the links between nodes carry important information for inferring the attribute values [6]. In many cases, there is auto-correlation between the labels of the nodes, which means that linked nodes are likely to share the same attribute values, but other, more complex correlations can be modeled and exploited [12].

Link prediction: Here the task is to predict the edges in the network [8, 11]. The link prediction problem can be formulated as a classification problem where we associate a binary variable for each pair of nodes that is *true* if a link exists between the two nodes and *false* otherwise. The prediction can depend on structural features computed based on the network (e.g., the number of common neighbors, etc.) and attribute values of nodes. Link prediction is difficult since it involves a large class skew (a priori, an edge is typically much more likely to *not* exist).

Entity resolution: Here the task is to identify when two nodes in the graph are referring to the same real-world entity. In that case, the nodes should be merged, and their attributes and links should be updated accordingly. Common approaches use a variety of similarity measures, often based on approximate string matching criteria [5, 4]. These work well for correcting typographical errors and other types of noisy reference attributes. More sophisticated approaches make use of domain-specific similarity measures and often learn such mapping functions from resolved data. Other approaches take graph structure and similarity into account [1, 7] and allow dependencies among the resolutions, e.g., *collective* entity resolution [3]. Other approaches adopt a constraint programming approach to express and solve the entity resolution problem [2].

Finally, recently a new approach [10] has been proposed to perform joint inference between the different types of tasks that uses *coupled collective classifiers* to propagate information among solutions to the problem.

3. PROPOSED FRAMEWORK

The workflow of our proposed framework is depicted in Figure 1. Data analysts start by specifying the prediction domain and features (Section 3.1), and then specify the type of prediction (attribute prediction, link prediction, or entity resolution), the *prediction* and *confidence functions*, and the *iteration* mechanism (Section 3.2). Our specification language for defining inference tasks builds upon Datalog. A detailed explanation and discussion of the language and other data model aspects can be found in [9]. Two example programs for performing attribute prediction and entity resolution are shown in Figure 2. We will use these programs as running examples for our discussion. The programs are used to clean scientific collaboration data extracted from DBLP. The data is structured as follows. Nodes represent authors, and edges represent scientific collaboration relationships between pairs of authors. Each author is associated with an attribute indicating his/her main research interest among three different areas of Computer Science: Databases, Machine Learning, and Software Engineering. The purpose of the attribute prediction program is to figure out the research interests of authors who do not have them, and the purpose of the entity resolution program is to figure out whether similar names in the dataset actually refer to the same author.

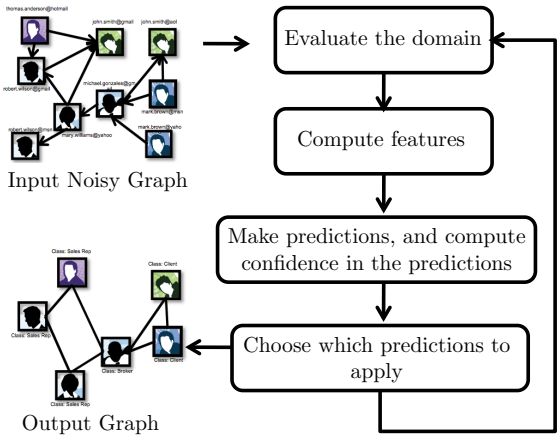

Figure 1: Workflow of the unifying framework

3.1 Defining Prediction Domains and Features

In order to define our graph inference tasks, we need to specify the prediction domains and features. Prediction domains are used to specify the set of nodes (in case of attribute prediction) or the set of pairs of nodes (in the case of link prediction or entity resolution) that are considered for prediction. Features are used to define different metrics and signals that are given to the prediction function in order to come up with the decision.

Features: We can divide the features broadly into three categories based on their complexity: local, local structural, and global structural. Local features are based on the attributes of the prediction elements. They can be defined for nodes (e.g., age, income, etc.) or pairs of nodes (e.g., similarity based on an attribute). Local Structural features require exploration of a small fixed neighborhood around the prediction element. Examples of such features can be a node's degree, the number of common neighbors between pairs of nodes, etc. Global Structural features are not tied to a specific neighborhood and can encompass the entire graph (e.g., Katz coefficient or betweenness centrality).

The features can be specified using Datalog in a straightforward manner. In Figure 2, we define `DB-Coauthors` to express the number of coauthors of author `X` whose research interests are in Databases. Another example is `Intersection` which counts the number of common coauthors between a pair of authors.

Domains: While features may be defined for all prediction ele-

```
DOMAIN AP-Domain(#X):- Node(X,'?') {
  DBCoauthors(#X,Count<Y>):- Edge(X,Y),Node(Y,'DB')
  ...
  AP-Features(#X,DB,ML,SE):- DBCoauthors(X,DB),
    MLCoauthors(X,ML),SECoauthors(X,SE)
  AP-Predictions(#X,predict-AP(DB,ML,SE),
    confidence-AP(DB,ML,SE)):- AP-Features(X,DB,ML,SE)
}
ITERATE(*) {
  UPDATE Node(X,A):- AP-Predictions(X,A,C) IN TOP(C,10)
}
```

(a)

```
DOMAIN AllNodes(#X) :- Node(X,_)
        {  Degree(#X,Count<Y>):- Edge(X,Y)    }
DOMAIN ER-DOMAIN(#X,#Y):- Edge(X,Z),Edge(Y,Z) {
  Sim(#X,#Y,strsim(X,Y)):- Node(X,_),Node(Y,_)
  Intersection(#X,#Y,Count<Z>):- Edge(X,Z),Edge(Y,Z),
    X!=Y
  Union(#X,#Y,DX+DY-I):- Degree(X,DX),Degree(Y,DY),
    Intersection(X,Y,I)
  Jaccard(#X,#Y,I/U):- Intersection(X,Y,I),Union(X,Y,U)
  ER-Features(#X,#Y,S,J):= Sim(X,Y,S),Jaccard(X,Y,J)
  ER-Predictions(#X,#Y,confidence-ER(S,J)):-
    ER-Features(X,Y,S,J), predict-ER(S,J)=TRUE
}
ITERATE(*) {
  Merge(X,Y) :- ER-Predictions(X,Y,C) IN TOP(C,10)
}
DEFINE Merge(X,Y) {
 INSERT Edge(X,Z) :- Edge(Y,Z)
 DELETE Edge(Y,Z)
 DELETE Node(Y,_)
}
```

(b)

Figure 2: (a) Attribute prediction and (b) Entity resolution program fragments

ments, often we want to restrict our attention to only a subset of the elements to make analysis tractable. We refer to such a subset of elements as the *prediction domain*. Prediction domain constructs are used to enumerate the elements for which predictions are made and feature values need to be computed. For attribute prediction, the prediction is over attribute values of the nodes and we can use the `DOMAIN` construct to restrict our attention to a subset of the nodes. This allows us, for example, to predict attribute values only for nodes with missing attribute values, or to predict attribute values only for nodes that have some percentage of neighboring values observed (not missing). Judicious use of prediction domains is especially important for tasks such as link prediction and entity resolution, where the prediction takes place for pairs of nodes. For a reasonably-sized network, it is infeasible to check every possible prediction element, and we must be able to limit the possible node pairs that are considered.

We use the keyword `DOMAIN` for defining a domain for features. For example, during entity resolution, we may want to restrict ourselves to pairs of nodes that are sufficiently close to each other in terms of graph distance, or *string similarity distance* [5] between their names. In Figure 2(b), we define `ER-DOMAIN` to include pairs that are within a distance of 2 links from each other. In addition to string similarity and graph neighborhood predicates, other efficient `DOMAIN`-rule-friendly predicates are equality predicates, and locality sensitive hashing. All of these predicates capture different notions of closeness/similarity.

3.2 Iterative Inference and Updating

The next step in the analysis process is to perform the required inferences and updates. For each prediction element, the prediction is made by applying a *user supplied function* over the features computed in the previous step and returning a *prediction* and a *confidence* (or *score*) value. This function can either be a user defined function or a function that is the output of some machine learning system; in the context of GRDB, we treat it as a black box. For attribute prediction, commonly used prediction functions include classifiers like naïve Bayes, logistic regression, and decision trees. Similarly, for link prediction, the problem of deciding whether to add an edge between a pair of nodes is often treated as a *binary* classification problem, and the functions listed above can be used as well. In some cases, especially for entity resolution, a similarity function might be used instead to compute a similarity score for a pair of nodes, and then a thresholding mechanism may be used to decide which nodes to merge or which edges to add.

Depending on the user's specification in the program, the program may just make one pass and commit all of the predictions made. In other cases, the program may choose to commit a subset of the predictions, and iteratively recompute the features and perform inference on the updated graph. The updates include attribute value changes (for attribute prediction), edge insertions/deletions (for link prediction), and node merges (for entity resolution), and we must recompute the values of the features in response to these updates. Before the update acutally takes place, the data analyst can browse the proposed predictions and fix mis-predictions if any. We discuss this interactive aspect of GRDB in Section 5.

In Figure 2(b), `predict-ER` and `confidence-ER` are the prediction and confidence functions respectively. They both take the value of the string similarity and the Jaccard coefficient (defined as the size of the common neighborhood between two nodes divided by the size of the combined neighborhood), and return the prediction and confidence, respectively. In this program, we commit only the top 10 predictions after every iteration (sorted by confidence), and continue iteratively for further predictions. `Merge(X, Y)` indicates that the graph update operation to be performed is a merge (corresponding to entity resolution). Other examples include `INSERT Edge(X, Y)`, indicating edge addition between nodes `X` and `Y` (for link prediction), and `UPDATE Node(X,Att=V)`, indicating that the attribute value of `Att` should be changed to `V` for node `X`, (for classification or attribute prediction).

The update operations corresponding to link prediction and attribute prediction are simple (i.e., a single rule). However, the `Merge` operation can be composite, i.e., defined in terms of other operations. This allows the user specify exactly how to update the attribute values for the new node that is created. In Figure 2 the merge operation for two nodes `X` and `Y` simply copies the edges of `Y` to `X` and then deletes `Y`.

4. SYSTEM ARCHITECHTURE

To implement our framework, we built a deductive database system on top of the Java Edition of the Berkeley DB key/value store. Our implementation of a graph data analysis system involved two key components. First, we implemented a full fledged non-transactional relational database system that has *a query parser*, *a rule-based query optimizer*, *a relational expression converter* for converting Datalog rules to relational expressions, and *a plan executor*. Second, we implemented the necessary special logic to enable our framework, such as incremental maintenance of various types of views. Incremental maintenance is particularly important because of the iterative nature of the framework. After every iteration, not all the features/domains have to be recomputed as a result of the updates/predictions. Therefore, we materialize the result of every Datalog rule in the system, and we treat them as materialized views over the base relations. As the base relations change in response to the predictions made during analysis, we maintain these views accordingly. In our prototype, we use different methods to handle

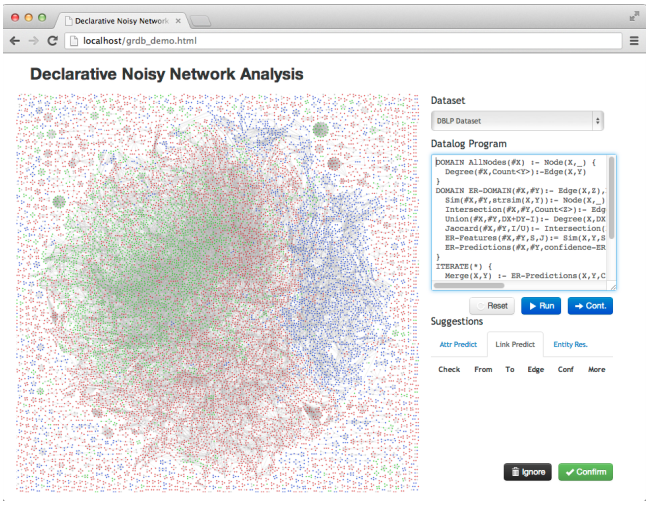

Figure 3: Main interface and visualization of selected dataset

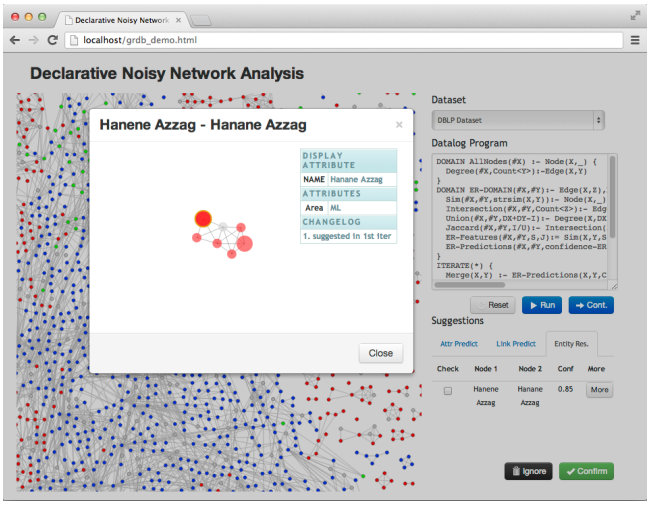

Figure 4: User feedback and local subgraph view

feature views, DOMAIN views, and cascaded views (where the output of one rule is propagated to another rule). We refer the reader to [9] for further details.

5. DEMONSTRATION PLAN

Our demonstration will illustrate how the user can easily perform graph analysis declaratively on several noisy networks, and how she can participate in the analysis process in GRDB.

SIGMOD attendees will be able to explore different datasets, and interact with the visualization of the network through GRDB's web-based console. After choosing a dataset, the corresponding graph will be visualized and the users will be able to zoom in and out to see the local structures; the user can also click a vertex to see its attribute details, by using the main interaction pane on the left side. To perform analysis, the user can edit and run Datalog programs for performing different prediction tasks. Figure 3 shows the main GRDB screen used for analyzing a subset of the DBLP coauthor network described above (having 16k authors and 40k coauthor relationships), using the ER Datalog program shown in Figure 2(b). After specifying a Datalog program, the user clicks the *Run* button to call the GRDB inference engine. After each iteration, the inference engine will return potential predictions that will be listed in the bottom right *suggestions pane*, including the details of possible actions and the confidence score.

GRDB allows the user to make the final decisions about the prediction results, if so desired. A practical middle ground is for the user to review the low-confidence predictions, and for the system to apply the high-confidence predictions automatically. After receiving the prediction results for a set of nodes/pairs of nodes, the user can examine the detailed node/neighborhood information to make a decision of whether to apply or ignore each of the suggested changes. In the suggestions pane of our web console, the user can click *More* button at the end of each suggestion to open a detail subgraph of the node or the pair of nodes under consideration. Figure 4 shows the detail subgraph view for a pair of authors that are predicted to be the same. The two nodes in the pair are highlighted by making them larger than their neighbors. If the user clicks on a node, it will be highlighted and its attribute details will be shown. By examining the details of the suggested node or pair, user can click *Confirm* or *Ignore* on the suggestion. They can continue the execution of the program by clicking *Cont.*, or they could revisit and modify the Datalog program to perform a new analysis.

6. TAKE-AWAY MESSAGE

This demonstration highlights the effectiveness of our system for performing declarative graph analysis over noisy information networks. The key insight behind our approach is to decouple the operations that require traversing the graph structure (typically the computationally expensive step) from the operations that perform modification and update of the extracted network. Our working GrDB prototype enables analysts to write declarative programs (in a Datalog-based language) to specify attribute prediction, link prediction, and entity resolution tasks. It additionally enables the analyst to visualize and control the analysis process through providing feedback about specific inferences being made.

Acknowledgements: This work was supported in part by NSF Grants IIS-0916736 and IIS-0746930, and an IBM Collaborative Research Award.

7. REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.

[2] A. Arasu, C. Re, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, 2009.

[3] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM TKDD*, 1:1–36, 2007.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, 2003.

[5] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. of IJCAI Workshop on Information Integration*, August 2003.

[6] D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification. In *SIGKDD*, 2004.

[7] D. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM SDM*, 2005.

[8] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *CIKM*, 2003.

[9] W. E. Moustafa, G. Namata, A. Deshpande, and L. Getoor. Declarative analysis of noisy information networks. In *ICDE Workshop on Graph Data Management*, 2011.

[10] G. Namata, S. Kok, and L. Getoor. Collective graph identification. In *KDD*, 2011.

[11] M. J. Rattigan and D. Jensen. The case for anomalous link discovery. *SIGKDD Explorations Newsletter*, 7:41–47, 2005.

[12] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 2008.