

---

# A Pipeline Approach to Graph Identification: Supplementary Materials

---

Galileo Mark Namata  
Lise Getoor

NAMATAG@CS.UMD.EDU  
GETOOR@CS.UMD.EDU

Department of Computer Science, University of Maryland, College Park, MD 20742 USA

## 1. Synthetic Data Generator

To evaluate the performance improvements of our optimizations, we created 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 network 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 an input graph from the output graph by adding different types of noise. The algorithm is shown in Algorithm 1.

---

### Algorithm 1 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: Create ambiguous references for nodes from  $\mathcal{G}_{temp}$
- 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}

---

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, including “small-world” and power law degree distribution. For our experiments, we use the Forest Fire generation model

(Leskovec et al., 2007) which models these properties, as well as many others commonly observed in real social networks. We used a *forward burn probability* of 0.4 and a *backward burn probability* of 0.2.

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 (Rattigan et al., 2007) (2 labels, with 1/10 of the graph initially labeled randomly) and create binary attributes based on those labels using the method described in (Bilgic & Getoor, 2008) (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 labels (varied to control link existence ambiguity) generated using the method described in (Rattigan et al., 2007). These attributes were generated for link prediction with the intuition that nodes with similar labels 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 (Bhattacharya & Getoor, 2007) and vary  $p_a$  to control the reference ambiguity. The resulting network is our synthetic output graph.

We create an input graph from our output graph by adding noise to the output graph. We add noise in four ways. First, we add ambiguous references to the graph by adding 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 has the same attributes 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  $n_1$  of the output graph, to an input graph node, corresponding to an output graph node  $n_2$ , if  $n_1$  and  $n_2$  share an edge. Once the reference nodes are generated, we add noise to the attributes of those nodes by removing the labels of all the nodes and randomly changing, as appropriate, the values of some 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 to the current number of edges).

## 2. Evaluation

We evaluate the performance of the ER, LP, and OC 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 (Bhattacharya & Getoor, 2007). 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 nodes will vary based on the entity resolution performance. We address this by evaluating link prediction and object classification by mapping the predictions to and evaluating over 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.

## References

- Bhattacharya, I., & Getoor, L. (2007). Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1, 1–36.
- Bilgic, M., & Getoor, L. (2008). Effective label acquisition for collective classification. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 43–51).
- Leskovec, J., Kleinberg, J., & Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery Data*, 1, 2.
- Rattigan, M. J., Maier, M., & Jensen, D. (2007). *Exploiting network structure for active inference in collective classification* (Technical Report). University of Massachusetts Amherst.