PREDICTING LINKS IN LARGE COAUTHORSHIP NETWORKS

Kevin Miller, Vivian Lin, and Rui Zhang
Group ID: 5

1. INTRODUCTION

The problem we are trying to solve is predicting future links or recovering missing links of large-scale coauthorship networks by supervised random walk method[1]. Although [1] addresses the problem with reasonable accuracy, issues of efficiency and scalability as well as more detailed problems of the algorithm still remain to be solved or at least mitigated. To achieve this goal, we apply the methods of error back-propagation [2] and of exploring overlapping communities[3] to the classical supervised random walk algorithm.

2. PRIOR WORK

[1] proposes a supervised learning method for predicting future friendship links in coauthorship networks. To predict whether a source nodes will be friends with a candidate node v in the future, the algorithm computes the probability that a random walker starting at (and randomly returning to) *s* would land on *v* in any given timestep and compares the probability to a threshold. Blackstrom and Leskovec use an SVM-like algorithm to learn a model for the transition probabilities in a network, based on known ground-truth friendships and non-friendships. The optimization problem involved in this learning phase penalizes all instances in which a ground-truth non-friend node is assigned a higher random walk probability than a ground-truth friend node.

One problem with the classical random walk method is that we don't know how the runtime scales with network size, or how likely the algorithm is to converge to a poor local minimum given certain starting points. There are several factors contributing to the algorithm's runtime. First, there is the amount of time we have to wait for probability and gradient computation steps (both iterative algorithms) to converge each time we compute the cost and gradient for a new w. Second, there is the fact that probabilities and gradients have to be computed for each source node, although they are independent from each other and therefore can be computed in parallel. Third, even when the algorithm does converge to a locally optimal w, we do not know how far from globally optimal it is, since the optimization problem is non-convex. We can use several different starting points to increase our odds of finding a good local minimum, but this adds to the runtime and still does not guarantee any starting point will result in a good local minimum.

To solve the first problem, we consider calculating our gradients using error back-propagation described in [2], in hopes of reducing the amount of computation needed to find an acceptable approximation to the gradient. Like [1], [2] presents an algorithm that learns a model for computing random walk probabilities. However, in [2], these probabilities correspond to the likelihood of a random walker without random restarts visiting a given node. Also, the random walker in [2] has the option of randomly jumping to a node that is not necessarily connected by an edge to the current node. The most important difference between [2] and [1] however, is the fact that the task in [2] is not to predict links between nodes, but simply to predict some scoring of a node (which might, for example, be used to rank web pages). In particular, the optimization problem in [2], unlike the one in [2], penalizes the square difference of the estimated random walk probabilities from some target value. As in [1], a gradient-based algorithm (in this case simple gradient descent) is used to solve the optimization problem and learn a model for transition probabilities, based on pairwise node features for each edge.

To solve the second problem, we use the clique propagation algorithm to find overlapping communities in our network, so that we can treat each overlapping community as a separate network and thus solve many small, independent problems in parallel to a get an approximate solution to our larger problem. This approximation should be fairly accurate, given that the empirically observed likelihood of link formation decreases rapidly with node distance, as illustrated in [1] (in fact, the random restart term in [1] ensures that the supervised random walks algorithm will model this relationship as an exponential drop-off). All in all, it seems reasonable to assume that most of the information about link formation will be very local with respect to the source node.

Using overlapping communities also provides us with an interesting approach to the third problem. If we average together the w's obtained for each overlapping communities, we very well might end up with a w that is a good starting point (both in terms of convergence time and convergence optimality) for training on the entire network.

3. ALGORITHM

The main contribution of our work is to integrate the method of error back propagation and exploring overlapping communities into the original supervised random work method. We will give a brief introduction of the classical random work and more detailed description of error back-propagation and overlapping community algorithms.

Supervised random walk:

The key algorithm of random walk is a well defined optimization problem:

$$\min_{w} F(w) = \|w\|^2 + \lambda \sum_{d \in D, l \in L} h(p_l - p_d)$$

where w is the weights of features extracted from the network, and h is a predefined loss function. Here, p is the stable distribution of the probability score assigned by the ranking function. To compute p, a Markov network with p representing the final stable distribution of the network is defined. Let Q denote the probability transition matrix, where each element is the probability that one particular node forms a link with another node. Let p to be the final score distribution, then we have

$$p^T = p^T Q$$

Because the equation cannot be solved directly, we use an iterative way to approximate the solution (Note that in the defined Markov network, we never know whether the converged score is correct or not, but we can verify the result by examining its prediction accuracy) by iterating the following formulas until they converge :

$$p_u^{(t)} = \sum_j p_j^{(t-1)} Q_{ju}$$

$$\frac{\partial p_u}{\partial w_k}^{(t)} = \sum_j Q_{ju} \frac{\partial p_j}{\partial w_k}^{(t-1)} + p_j^{t-1} \frac{\partial Q_{ju}}{\partial w_k}$$

Once we obtain p and its gradient, Q is recomputed as follows:

$$Q'_{uv} = \begin{cases} \dfrac{a_{uv}}{\sum_w a_{uw}} & if\ (u,v) \in E, \\ 0 & otherwise \end{cases}$$

Then we get the gradient of the objective function by:

$$\frac{\partial F(w)}{\partial w} = 2w + \sum_{l,d} \frac{\partial h(p_l - p_d)}{\partial w}$$

where dp/dw can be computed as follows:

$$\frac{\partial p_u}{\partial w} = \sum_j Q_{ju} \frac{\partial p_j}{\partial w} + p_j \frac{\partial Q_{ju}}{\partial w}$$

Then we can update w by:

$$w := w - \alpha \frac{\partial F(w)}{\partial w}$$

Once w is updated, we'll return to the beginning of the algorithm and update p and its gradients and Q matrix. We follow this process until w converges.

Error back-propagation:

The main idea of back-propagation is to treat the coauthorship network as a neural network. For a particular source node s in layer l, the author traces back (according to the edge direction) from s and arrive at its neighbor nodes in layer l-1 pointing s (parent nodes of s). Then the author traces back from these nodes at layer l-1 and arrive at nodes in layer l-2. Continuing this process, we can transform the original network into a specialized (by the selected source node) neural network. One problem is how to define the first layer in this neural network. According to [2], the first layer is composed by nodes that are visited in the last. In above, each of the source node's parents in the coauthorship network has a particular layer to define its position in the neural network. Figure 1 is a good example to illustrate this transformation process:

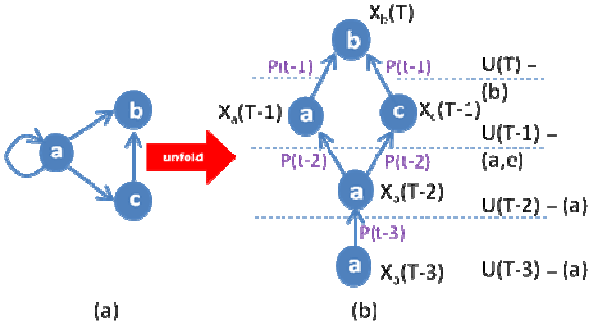


Figure 1. (b) is the transformed neural network from original network (a)

Let Q denote the probability transition matrix, just as the one defined in random walk algorithm. The forward probability computation can be followed by:

$$p_k^{(t)} = \sum_i p_i^{(t-1)} Q_{ik}^{(t)}$$

By matrix representation, we have:

$$p^{(t)} = Q^{(t)T} \cdot p^{(t-1)}$$

To compute the transition probability matrix, one needs to back propagate the error from the source node layer. Instead of using the square difference of source node score returned by the ranking function as our error measure, we use the supervised random walk objective function as our error measure. The reason is that 1) the function is more powerful to characterize different error measure by choosing different h functions; 2) The function's gradient is already computed in our implementation of supervised random walk, so we don't need to recompute it for error-back propagation, and also it can be a good comparison with random walk. The error is characterized by:

$$E^p = F(w) = \|w\|^2 + \lambda \sum_{s \in S} \sum_{d \in D, l \in L} h(p_l - p_d)$$

To minimize the objective function, gradient descent by a generic parameter w is applied:

$$w' = w - \eta \frac{\partial E^p}{\partial w} = w - \eta \frac{\partial F(w)}{\partial w}$$

To compute dF(w)/dw, the chain rule is applied:

$$\frac{\partial F(w)}{\partial w_k} = \sum_{t=-\infty}^{T-1} \left(\sum_i \sum_j \frac{\partial F(w)}{\partial Q_{ij}^{(t)}} \cdot \frac{\partial Q_{ij}^{(t)}}{\partial w_k} \right)$$

where Q is the transition matrix, and Qij represents the probability of node random walker follows a link from node i to node j.

Combined with

$$p_j^{(t)} = \sum_i p_i^{(t-1)} Q_{ij}^{(t)}$$

we have

$$\frac{\partial F(w)}{\partial Q_{ij}^{(t)}} = \frac{\partial F(w)}{\partial p_j^{(t)}} \cdot \frac{\partial p_j^{(t)}}{\partial Q_{ij}^{(t)}} = \frac{\partial F(w)}{\partial p_j^{(t)}} \cdot p_i^{(t-1)}$$

By matrix representation of dQij/dw and dF(w)/dQij, we have the following representation that can be implemented directly:

$$\frac{\partial F(w)}{\partial w_k} = \sum_{t=-\infty}^{T-1} (p^{(t-1)})^T \cdot \frac{\partial Q^{(t)}}{\partial w_k} \cdot \frac{\partial F(w)}{\partial p^{(t)}}$$

Because we have computed dQ/dw in our implementation of random walk algorithm, what we only need to do now is to compute dF(w)/dp. The method of computing it is in a recursive way. Let

$$\delta^{(t)} = \frac{\partial F(w)}{\partial p^{(t)}}$$

then we have

$$\delta_k^{(t-1)} = \sum_i Q_{ki}^{(t)} \delta_i^{(t)}$$

By matrix representation, we have

$$\delta^{(t-1)} = Q \cdot \delta^{(t)}$$

Here we can see the contribution to the output error of each layer which is computed iteratively in a backward direction.

One thing we should really be careful about is delta of the output layer, because it is the initialization of the whole back propagation. Using F(w) as our objective function, we have

$$\delta^{(T)} = \frac{\partial F(w)}{\partial p^{(T)}} = \lambda \sum_{d \in D, l \in L} \frac{\partial h(\Delta)}{\partial \Delta} \cdot \frac{\partial \Delta}{\partial p^{(T)}}$$

$$= \begin{cases} \lambda \sum_{d \in D, l \in L} \Delta(1-\Delta) & if\ p^{(T)} is\ negative\ sample \\ -\lambda \sum_{d \in D, l \in L} \Delta(1-\Delta) & if\ p^{(T)} is\ positive\ sample \end{cases}$$

where h function is the predefined loss function and

$$\Delta = p_l - p_d .$$

Overlapping communities:

Basically the algorithm of computing overlapping communities is pretty straightforward. It doesn't have much math involved in it. [3] provides a good method to detect overlapping communities of a real network. The main idea is to find all complete graphs first, and then look for general k-clique connected subsets by studying the overlap between them. According to [3], a k-clique-community is defined as connected clique components where the neighboring cliques are linked to each other by at least k-1 common nodes. The clique-clique matrix is used to characterize the relationships between cliques in the network. The advantage of this matrix is that it captures almost all necessary information that will be used to find cliques that participate in a k-clique-community, for any value of k; in particular, there is no need to recompute this matrix for different k values.

The following method is used to locate cliques. The starting value of clique size k is the maximum degree over all nodes in the network. Given a k value, the method repeatedly selects a node and extracts all k-cliques that contain that node, and then removes the node and its edges. Once there are no more nodes to select from, the algorithm decrements k and goes through the clique-finding procedure again, until k = 1.

4. EXPERIMENTS AND RESULTS

4.1 Data statistics

In this project, we first create a set of scale-free graphs using Copying model to test our program. Each graph starts with a 3-node complete graph, then each new node connects to three old nodes in two ways. First, with probability alpha, it picks a node uniformly at random. Second, with probability 1-alpha, it picks a node based on the probability proportional to the degree of each nodes. Two independent Gaussian features are assigned to each edge. A random walk generated by a fixed model w is used to obtain ground truth link labels, so that one can attempt to recover this w from (possibly a noisy version of) the features and the ground truth training link labels.

Next, we evaluate the algorithm on the coauthorship dataset from DBLP Computer Science Bibliography(http://dblp.uni-trier.de/xml), where nodes represent authors and edges represent the coauthorship. Note that for the relationship involving more than 2 authors, a complete sub-graph is formed to represent their co-authorship, and that's why we introduce the idea of exploring overlapping communities, which capture the sub-complete graph (k cliques) structure of our coauthorship network.

In addition, we examine the properties of the synthetic and the co-authorship dataset. While one synthetic data is examined; we create four graphs from the co-authorship data. The four graphs are built by extracting all the edges created before 1970/1/1, 1980/1/1, 1990/1/1, and 2000/1/1. Notice that the graph of 1980 contains similar among of nodes (~15,000) as synthetic data. With these five graphs, the statistics are listed as follows:

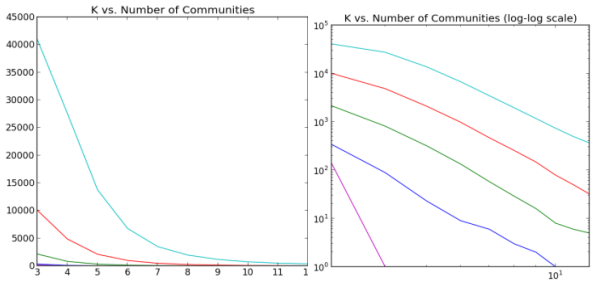


Figure 2. Both graph shows the clique size (k) versus the number of communities containing at least one clique of size k. The right graph is the log-log scale of the left graph, which shows a power law of the relationship

As shown above, the first statistics is the clique size (k) versus the number of communities containing at least one clique of size k. We examine various time limits (1970, 1980, 1990, and 2000) to explore a general trend of this relationship. Every clique of size k is a complete graph with k nodes. For different time limits, we calculate the number of communities before that time limit. It is interesting to note that the relationship follows a power law, which is clearly shown in the right graph.

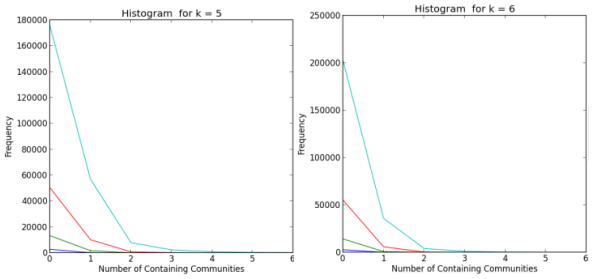


Figure 3. The number of communities versus the number of nodes belonging to that number of communities for k = 5 or 6.

The second statistic is the number of communities versus the number of nodes belonging to that number of communities. We have plotted four graphs to explore the influence of k on this relationship. It is clearly shown that as k increases, the relationship tends to follow an exponential decrease rule. This makes sense, because naturally the nodes that can be shared by a large number of communities should be really small. Such nodes with high value of community number are really hot nodes. They can form many links across many different communities potentially, and they also act as a bridge to form links between communities (or form super links if we see communities as super nodes).

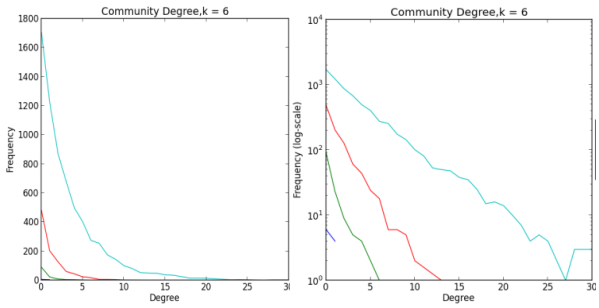


Figure 4. Degree of communities versus the number of communities having that degree. Right graph is the log scale of the left graph(x axis unchanged), which shows a exponential decreasing relationship

The third statistic is the degree of communities versus the number of communities having that degree. Basically, the degree of a community is the number of other communities overlapping with the selected community. If we view the community as a super-node, the graph would be the same as (super) node degree distribution. It is clearly shown that the community degree distribution follows a law of exponential decrease. When taking log of the y axis (see the right plot), we see the relationship becomes linear, which verifies our result.

4.2 Results and Finding

In this project, we implemented supervised random walk algorithm[1] and tried to improve its performance by integrating back-propagation and overlapping community algorithms. Because the data size is big, to improve efficiency, most of our big operations are in a vectorized form, and we take advantage of sparse representations whenever they are warranted. Also, our code is parallelized over source nodes. We will first show the synthetic data results, and then talk about our coauthorship data results.

Synthetic data:

First, as a sanity check, we have roughly replicated the synthetic data test in [1], in which ground-truth link labels are generated by doing a random walk with a fixed w. Due to time constraints, we limited the size of our networks to 5000 nodes and randomly selected only 20% of the negative and positive nodes to include in our objective function. To ensure convergence of the pagerank update, we included a uniform-at-random jump to any node, neighbor or otherwise, with probability 0.15.

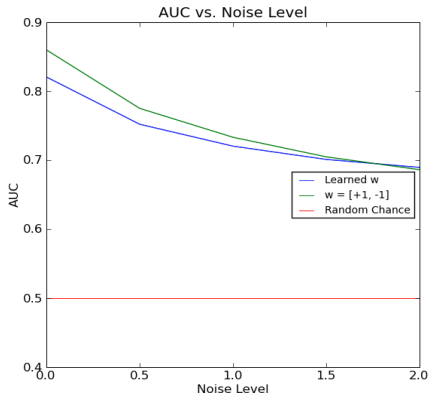


Figure 5. Experiments on synthetic data.

Our results for this experiment are not as good as those in [1], but they do show a similar trend. In particular, note that the AUC from the learned w eventually overtakes the AUC from w = [+1, --1] as the noise level is increased. Also, note that the magnitude of w decreases as the noise level increases, which makes sense, since w is less likely to overfit to very noisy features that don't correlate well with the ground truth in the training set. We suspect that the two most likely reasons that we don't do as well as [1] on synthetic data are (a) We don't provide as much support for the objective in each graph, since we downsample our ground truth link set by 20%, and (b) We use a pretty high jump probability term (0.15), which is almost as high as our alpha value, which might undermine the source-sensitivity of our random walk (indeed, we note that in most cases, there is little to no difference between the lowest ground-truth positive probability and the highest ground-truth negative probability). It is also possible that we are overregularizing (we're currently using lambda = 1.0, as suggested in [1]).

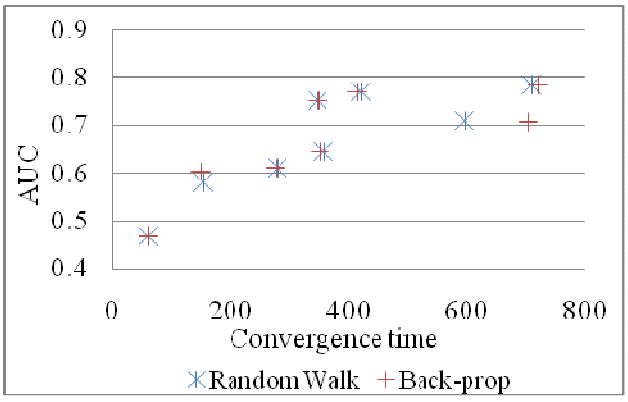


Figure 6. Convergence time vs. AUC

Second, we explore the relationship of convergence time and AUC (Figure 6). There should be a trade-off between these two factors, as we might expect it take more time to find a good ending point. From the graph, we can see that as the convergence time increases, the auc values also increase, albeit with diminishing returns.

Thirdly, we explore the effect of epsilon (the stopping criterion for gradient computation) over the convergence time of both algorithms (the AUC turns out roughly the same):

Epsilon	1e-12	1e-9	1e-6	1e-4	1e-2
BP	814s	778s	1755s	1450s	1388s
RW	765s	785s	1548s	1565s	1354s

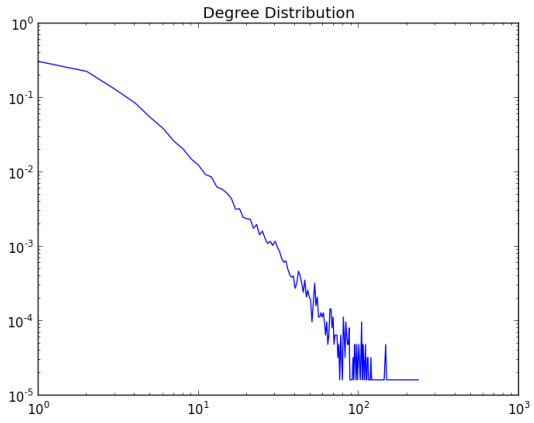
It shows that the algorithms are sensitive to epsilon between 1e-9 to 1e-4. The trend that we see, where the best epsilon is somewhere in the middle of our range of epsilons, is not surprising; for extremely small epsilons, each gradient computation would take very long, and for extremely large epsilons, gradient computations would be faster, but there would be more outer iterations because the gradients would be less accurate.

Coauthorship dataset:

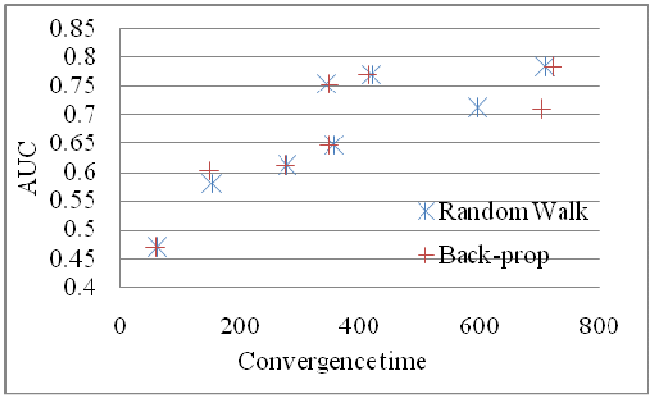
First, like [1], we made a table of the statistics of the coauthorship dataset, and the degree distribution:

	N	E	S	D	C
1980	15121	18610	3	16.67	37.00
1990	61674	101221	103	22.19	145.75

Table: Coauthorship dataset statistics. N and E is the number of nodes and edges respectively; S is the number of source nodes; D is the average number of destination nodes; C is the average number of candidates per source.

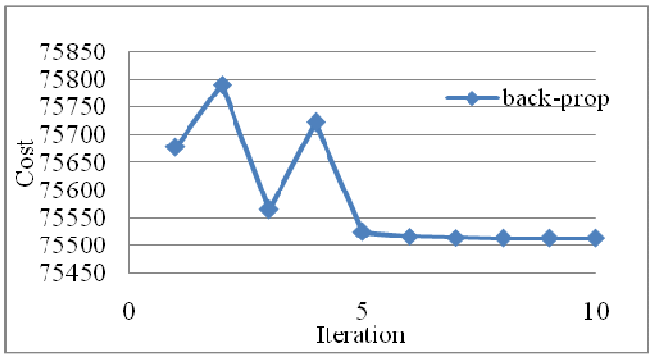


Secondly, we explore the relationship between the prediction error and the number of iterations, just as [1] did. From the graph, we can see as the iteration increase, the prediction accuracy increases accordingly. Sometimes the curve may not be monotonously increasing, because the update step may lead w to a worse value than its previous value (for example, the update step is too big, or close to flat area, and so forth). But the overall trend is reasonable.



From the graph above, we can see that the prediction error between back-propagation and rank-walk is very close. This makes sense, because we use the objective function F(w) as our error measure, and we just use a different way to compute the gradients of it, so the w should not be affected seriously (only the time cost should be affected).

Another thing we can see from the graph is that Netwon method really converges quickly. After two iterations, the prediction accuracy remains almost unchanged. Strictly speaking, we cannot say the algorithm converges based on the graph, because the prediction accuracy can be the same for different w, which is still not converged. To better illustrate when the back propagation algorithm converges, we plot the cost function over the number of iterations below:



From the cost graph, we can see the back propagation algorithm converges after 5 iterations, which is really fast.

Finally, we use the coauthorship dataset to test our ideas of overlapping communities. First, we compute the w over every community of the network. Since the size of each community is small, the computation is efficient. Then we initialize w as the average of the weights we computed from all communities, and the result shows that we don't need that much number of iterations before w converges, which is good. We use the 1990 coauthorship dataset as comparison:

	Time	Accuracy	Iteration
BP	7236s	0.904	12
RW	7184s	0.904	12
OP	6714s	0.904	10

Table: BP = back-propagation, RW = random walk, OP=overlapping community

From the table above, we can see that good initialization doesn't affect the final accuracy, but only influences the time cost. It might be possible that F(w) is trapped in some local minimum and produces a worse result, but this doesn't happen in our experiments, given that we haven't tried too many random starts.

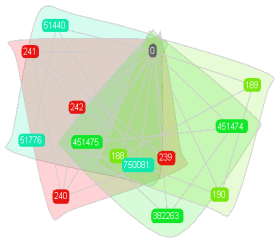


Figure 7. Example of a node that belongs to multiple communities

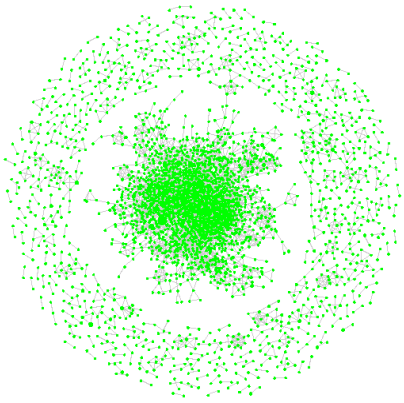


Figure 8. overlapping communities graph(4-clique): Each node represents one community

We chose to use the concept of overlapping communities to split a large problem into many smaller, independent problems, in hopes of making our algorithm more efficient and easier to potentially scale to multicore systems such as GPU's. Even without parallel processing, splitting by overlapping communities could have potential gains even on one processor as long as the big-Oh runtime of the algorithm with respect to network size is superlinear, as that would mean that for sufficiently large networks, the savings that come from decreasing the input size of each subproblem would eventually outweigh the added overhead. In fact, we found that when we split a 60K-node network into 25 overlapping communities, we found that training a w for each one took on average 6.5 seconds, and then training the whole network using the average w from the communities as a starting point took 6714; even if these communities were trained in series rather than parallel, we would still take considerably less time than the 7184 seconds that training on the whole network with a random starting point would take.

5. FUTURE RELATED WORK

There are many other ways of improving the supervised random walk. For instance, we can try to combine back-propagation with stochastic Newton's method or stochastic gradient descent, which might also improve the runtime of the algorithm. Stochastic gradient descent can achieve relatively the same performance as batch gradient descent but only need several percents of batch gradient descent's time. Also, by looking at the probabilities of random walk paths while back-propagating, we can have at least some control over the accuracy of our gradient computation while at the same time limiting the runtime of that computation.

Apart from that, it is also possible to improve the ROC curve by adding good features to the algorithm's feature set. One interesting idea we consider to add is that letting link predictions depend on each other. That is, once we predict that a link exists, we would add it to our network and recompute the random walk probabilities before predicting the existence of more links. In essence, we would be predicting groups of links rather than independently predicting one link at a time. Prediction order would matter here, of course, and it would be interesting to try to find orderings that lead to the most accurate groups of predicted links. Another intriguing idea is that of treating our probability-computation network as

a neural network and changing the activation function to a sigmoid.

REFERENCES

[1] L. Backstrom, J. Leskovec. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In Proc. WSDM, 2011.

[2] M. Diligenti, M. Gori, M. Maggini. Learning Web Page Scores by Error Back-Propagation. Proceedings of the 19th international joint conference on Artificial intelligence, p.684-689, July 30-August 05, 2005, Edinburgh, Scotland

[3] G. Palla, I. Derenyi, I. Farkas, T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society.Nature 435, 814-818, 2005.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Dig. Lib. Tech. Proj., 1998.