Active Search on Graphs

Xuezhi Wang
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA, USA
xuezhiw@cs.cmu.edu

Roman Garnett
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA
rgarnett@andrew.cmu.edu

Jeff Schneider
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA
schneide@cs.cmu.edu

ABSTRACT

Active search is an increasingly important learning problem in which we use a limited budget of label queries to discover as many members of a certain class as possible. Numerous real-world applications may be approached in this manner, including fraud detection, product recommendation, and drug discovery. Active search has model learning and exploration/exploitation features similar to those encountered in active learning and bandit problems, but algorithms for those problems do not fit active search.

Previous work on the active search problem [5] showed that the optimal algorithm requires a lookahead evaluation of expected utility that is exponential in the number of selections to be made and proposed a truncated lookahead heuristic. Inspired by the success of myopic methods for active learning and bandit problems, we propose a myopic method for active search on graphs. We suggest selecting points by maximizing a score considering the potential impact of selecting a node, meant to emulate lookahead while avoiding exponential search. We test the proposed algorithm empirically on real-world graphs and show that it outperforms popular approaches for active learning and bandit problems as well as truncated lookahead of a few steps.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]; H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms

Keywords

Active Learning, Graph Search

1. INTRODUCTION

Many learning applications consider a large amount of unlabeled data for which we would like to obtain labels, but it is too expensive to collect them all. These applications have led to increasing interest in active learning algorithms that choose data points for labeling with the goal of optimizing a criterion based on the accuracy of the model learned from the chosen points. A typical algorithm builds a model from the labels already collected and iteratively uses it to select the next point for labeling that is expected to most improve the model.

In this paper, we focus instead on the *active search* problem [5], where we seek points belonging to a certain positive class. Although we will still build a predictive model from the selected points, and may choose points to improve our model's accuracy, we will ultimately be evaluated only by how many positives we find among our queried points. Many real-world applications are active search problems, including drug discovery (where "effective drugs" are the sought class) and product recommendation (where "purchased products" are the sought class). In these examples, an accurate model is only useful if we can use it to locate more members of the desired class. We get no credit for model accuracy itself or correctly predicted labels themselves.

Although active search applications appear with many different types of data, here we restrict our attention to graphs, where the graph structure is known but labels on nodes are expensive to collect. There are many interesting applications of active search in graphs. In a marketing application, targeting a given individual might be quite expensive, but a social network might be available to infer the tastes of as-of-yet uncontacted users. A company might analyze a network of financial transactions in order to discover fraudsters, but investigating a particular selected entity is expensive. An academic or analyst might like to find papers on a particular subject in a citation graph without having to read too many of them.

One might expect typical active-learning algorithms to be appropriate for active search as well because they can produce a good model that can be used to find positives. However, in active search a good algorithm must trade off the need to exploit (use the current model to collect positives) against the need to explore (develop a better model to more accurately guess the positives in future selections). A traditional active learning method would focus entirely on exploring and only collect positives by accident. Therefore, strategies of a different nature are required. The exploration/exploitation feature of the problem might lead one to consider bandit algorithms for this task. This seems promising except that in active search an algorithm can not repeatedly make the same choice to collect more reward. Once it finds a positive, it must move on and look somewhere else for another one.

As is typical with active learning and bandit-style problems, the optimal active learning solution, in general, requires an intractable lookahead search over an exponential number of possible future queries and label outcomes. An algorithm based on evaluating the expected utility over a truncated lookahead has been proposed, and good empirical results have been obtained by using a smart pruning strategy that, in some cases, reduces the cost by orders of magnitude and makes longer lookaheads possible [5]. In the same work, it was proven that arbitrarily better performance can occur with even one further step of lookahead. In empirical examples, it seems that much better performance is available from looking ahead much further than is possible even with smart pruning.

Many successful algorithms for active learning and bandit problems do a myopic or 1-step evaluation of a well-crafted surrogate objective rather than directly optimizing expected utility. Inspired by their successes, we propose such a method for active search in graphs. We use a soft-label model for graphs, which attaches a "pseudonode" to each original node that holds the observed labels. For our surrogate objective, we propose the probability of a positive (the exploitation) plus a measure of impact based on the number of additional positives likely to be identified (the exploration). Both the model and the impact factor can be efficiently computed using incremental updates to the model matrices. We compare our method to uncertainty sampling, a modified UCB algorithm, and a previously proposed model for graphs. On three real-world graph datasets, our method outperforms all the others.

2. RELATED WORK

There has been much research in the area of semi-supervised learning, where the setting is the learning algorithm receives both a labeled training set and a set of unlabeled test points, and the objective is to predict the labels of the test points. Semi-supervised learning algorithms leverage the structure of unlabeled data during training to improve learning performance. Most of these works have been focused on achieving good classification with partially labeled data. In [10], the authors propose a Markov random walk based algorithm to classify unlabeled points using the information of labeled ones as well as the graph structure. The authors adopt two techniques, maximum likelihood with EM and maximum margin subject to constraints, to estimate the unknown parameters that indicate the distribution of each data point over the class labels. In [11], the authors propose a semi-supervised label learning method which is based on the Gaussian random field model. The mean field is characterized by a harmonic function, and can be efficiently obtained by matrix computation or belief propagation. In [6], the authors adopt a relational active learning model to improve both model estimation and prediction after acquiring a node's label. They propose a model which combines a network-based certainty score with semi-supervised ensemble learning, as well as relational resampling to utilize both the local relational dependency and sufficient global variance. In [2], the authors analyze the stability of several transductive regression algorithms, where the problem setting is similar to that in semi-supervised learning. There also has been some work on efficient semi-supervised learning, such as [3]. In this work the authors try to apply semi-supervised learning on 80 million images gathered from the Internet, with "clean labels" manually obtained on a small fraction. The authors have been able to obtain highly efficient approximations for semi-supervised learning that are linear in the number of images, compared to traditional methods that scale polynomially with the number of images.

Active graph search involves an exploration and exploitation dilemma, where the Upper Confidence Bound (UCB) algorithm [1, 9, 4] is a popular method of addressing this issue. The basic idea of UCB in multiarmed bandit problems is to sum the current estimate about the reward of each arm and the uncertainty about that arm. Choosing arms with a high expectation corresponds to exploitation and choosing those with high uncertainty corresponds to exploration. UCB is appealing because it comes with regret bounds but the setting is too confined to be used in the active search problem. UCB intends to repeatedly select good arms while the active graph search problem does not allow repeated selections. In [8], the authors propose contextual bandits with similarity information. We could use the graph structure to provide such information. However, this would not change the fundamental problem with bandit approaches for active search, which is that we will never select the same node more than once.

There is a more subtle issue as well. Ideally, the exploration component of an algorithm would optimize some measure of information gained from a label. In a traditional independent-arm bandit problem, this is easily replaced by the uncertainty for a particular arm because the information gain is confined to that arm. When a Gaussian process model is used in a bandit problem [9], information is spread throughout the model. Because of the symmetric and homogenous properties of typical kernels, the information gain for sampling at a point can again be substituted with the current model uncertainty at that point. Typical graph models offer no such easy way out. The potential information to be gained by choosing a hub can be much larger than that of choosing a disconnected singleton even if the latter is much more uncertain. This property motivates the impact factor in our proposed method.

3. APPROACH

3.1 Problem Description

Here we formally define a binary graph active search problem. We are given a finite set of n nodes, indexed $\{1, ..., n\}$, which have an unknown set of labels $Y = \{y_1, ..., y_n\}$ where $y_i \in \{0, 1\}$ and we want to identify the nodes for which $y_i = 1$. We are given a corresponding weight matrix $W = [w_{ij}]$, where w_{ij} indicates the strength of the relationship between y_i and y_j. Initially all nodes belong to the unlabeled set, U. At each iteration we choose a node i, find out y_i, and move node i to the labeled set, L. Our performance after k iterations is the sum of the y_i in L.

3.2 Models

We begin by considering models for predicting the unknown values of Y. In [11], the authors propose a harmonic function f to represent their predictions, which minimizes the energy function $E(f) = \frac{1}{2}\sum_{i,j} w_{ij}(f(i) - f(j))^2$. By setting the derivative to zero we can get $f = D^{-1}Wf$, where

D is the diagonal matrix with entry D_{ii} representing the degree of node i, i.e., $D_{ii} = \sum_j w_{ij}$. Separating the labeled points f_l and the unlabeled values f_u, and also the corresponding W matrix and D matrix, we get a more explicit form of f for unlabeled points:

$$f_u = (D_{uu} - W_{uu})^{-1} W_{ul} f_l$$

In practice we do not need to do the expensive matrix inversion. We can approximate f_u by iteratively multiplying an initial value f by the matrix $D^{-1}W$ and update only those entries in f_u until convergence.

Each entry f_i in the harmonic function is an indicator of the probability that a random walk starting from node i will hit a label 1 before it hits a label 0. However, this model has some problems, especially for active search. Suppose we first discover the hub node i of a star structure with label $y_i = 0$, which is connected to many nodes with label $y_j = 1$ in its immediate neighborhood N_i (node $j \in N_i$ if $w_{ij} > 0$). Discovering any number of nodes with label $y_j = 1$ in N_i will never increase any remaining element of f_u from N_i since a random walk will always stop at the 0 label of node i. Figure 1 shows an example of this hub-blocking problem, where target nodes (with $y = 1$) are shaded solid.

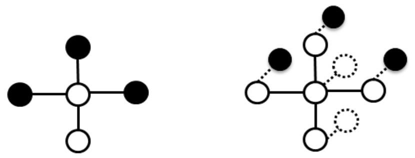

Figure 1: An example showing an original graph (Left) and the soft-label graph (Right)

3.2.1 *The Soft-Label Model*

In the original formulation of the harmonic function model [11], the resulting $f_i = P(y_i = 1|L)$ indicates the probability that a random walk starting from node i will hit a label 1 before it hits a label 0. Hence hitting a label 0 will effectively end the random walk and assign a label 0 to the starting node i, which is the main cause of the hub-blocking problem mentioned above. We can resolve this issue by changing the stopping criteria to be indeterministic, i.e., we add a probability η to the random walk, such that when it hits a labeled node, it stops with probability η, and with probability $1 - \eta$ it ignores the label and continues the random walk. More specifically, we attach a pseudo node to each labeled original node i to hold its label, and use the edge weight between the pseudo node and the original node to adjust η.

The harmonic function and the soft-label model can both take advantage of the structure information in the graph, but they utilize labeled information in different ways. Given a certain query node, the harmonic function is only able to use the labeled information from those nodes that have a path to this query node, and this path is not allowed to have any other labeled nodes on it (otherwise the labeled nodes would already have blocked the path). While the soft-label model can effectively utilize the labeled information from all the nodes in the graph, with the advantage that the closer labeled nodes have higher influence on the query node than the labeled nodes farther away.

Leaving f as the estimate of the original nodes associated with a label 1 and letting x_l represent the labeled pseudo nodes (with entry 0 for unlabeled nodes), we get:

$$f = D_*^{-1} \begin{bmatrix} W & D_l \end{bmatrix} \begin{bmatrix} f \\ x_l \end{bmatrix}$$

where W is the original weight matrix, and D_l is an $n \times n$ diagonal matrix with

$$D_{l(ii)} = \begin{cases} \frac{\eta}{1-\eta} \sum_j w_{ij} & i \in L \\ 0 & i \in U \end{cases}$$

which indicates that there is a transition probability η from a labeled node i to its labeled pseudo node. D_* is also a diagonal matrix with $D_{*(ii)} = \sum_j w_{ij} + D_{l(ii)} = \frac{1}{1-\eta} \sum_j w_{ij}$ for $i \in L$, acting as a row normalizing factor.

3.2.2 *Incorporating Prior Information*

It is often useful to include prior information on labels and we can attach a pseudo node to the unlabeled original nodes for this purpose. We set the pseudo node label to be the value of the prior. The weight of the attached edge represents the strength of the prior. We set the weight of node i to be $\omega_0 D_{ii}$, where ω_0 is the strength, and D_{ii} is the degree of node i. By a similar derivation as above and absorbing the row normalizing factor we get:

$$f = \begin{bmatrix} A & D' \end{bmatrix} \begin{bmatrix} f \\ x \end{bmatrix} \quad \Rightarrow \quad f = (I - A)^{-1} D' x \qquad (1)$$

where

$$A_{ij} = \begin{cases} (1-\eta)(D^{-1}W)_{ij} & i \in L \\ \frac{1}{1+\omega_0}(D^{-1}W)_{ij} & i \in U \end{cases}$$

$$D'_{ii} = \begin{cases} \eta & i \in L \\ \frac{\omega_0}{1+\omega_0} & i \in U \end{cases}$$

Here x is a predetermined vector with labels in the entries corresponding to labeled nodes, and a value π for the prior in the entries corresponding to unlabeled ones. f will be a vector we want to compute, with f_i indicating $P(y_i = 1|L)$ for all the nodes, but we only care about those entries i with $i \in U$.

A similar model would be the Cortes model [2], which is a generalization of the Gaussian Mean Fields model [11]. This model also has a kind of "softening" and we could plug it into our method, but we prefer the way the priors in our model give a smooth transition of values going away from labeled nodes.

Adding a prior in the model has several advantages. First it enables the model to distinguish between nodes not connected to any labeled nodes and nodes that are connected to 0-label nodes. Second, it localizes the active search, which means we would rather first search the closest neighborhood of a node with label 1. Imagine a large connected component in the graph that has only one labeled node which is positive. Using either the harmonic function or the soft-label model will result in the same f score for every node left in this component. After adding a prior we can get relatively higher scores in the neighborhood of this positive node, and the scores gradually decrease for the nodes farther away from it.

3.3 Selection Criterion

We propose a selection criterion with the following form:

$$\text{score}_i^{(t)} = f_i^{(t)} + \alpha \times \text{IM}_i^{(t)} \qquad (2)$$

where $f_i^{(t)}$ indicates the model's prediction for node i after seeing t labels, $\text{IM}_i^{(t)}$ is the expected impact on future positives found by choosing node i now, and α is a parameter trading off exploration and exploitation. At each iteration, we evaluate score$_i$ for all unlabeled nodes i, and choose the node with the highest score.

There are many possibilities for defining IM. The entropy in f_i would be an obvious choice, however that does a poor job of capturing how much effect node i has on the rest of the graph and especially how much it will increase the number of positives we find in the future after observing y_i. We can consider $\sum_{i \in U} f_i$ as an indication of the number of positives we will find in the future. Therefore, we propose to explicitly condition on the expected value of y_i and measure its potential to increase values of f in the unlabeled part of the graph. We propose:

$$\text{IM}_i^{(t)} = P(y_i^{(t)} = 1 | L^{(t)}) \delta(P(y))$$

where

$$\begin{aligned} \delta(P(y)) &= \sum_{j \in \{U^{(t)} \setminus i\}} [P(y_j^{(t+1)} = 1 | y_i^{(t)} = 1, L^{(t)}) \\ &\qquad - P(y_j^{(t)} = 1 | L^{(t)})] \end{aligned}$$

Using f vector as before with each entry f_i representing $P(y_i = 1|L)$, we have an equivalent form:

$$\text{IM}_i^{(t)} = f_i \sum_{j \in \{U \setminus i\}} (f'_j - f_j) \tag{3}$$

where f is the original prediction for each node and f' is the prediction conditioned on adding node i to the training set with label $y_i = 1$.

Note that we do not condition on seeing $y_i = 0$. Intuitively you might want to set up the impact criterion to marginalize over the unknown outcome. However, doing that would correspond to estimating the change in expected number of positives in this neighborhood under the assumption that your policy will continue choosing nodes in this neighborhood even if it sees a negative outcome. Of course this is not the policy we will follow. If a negative is observed, the policy will move to some other part of the graph. By doing it the way we propose we are representing both the unknown outcome and the decision that will follow (i.e. to continue choosing nodes in this neighborhood or not).

This impact factor is clearly heuristic and computing the true future expected increase in positives chosen is just as computationally intractable as implementing the full optimal policy. However, this definition of IM is able to tractably imitate a full look ahead by computing the full impact over all the nodes in the graph through the model. An example is enlightening. Imagine a graph of many separate components, each of which is a clique of widely varying size. A smart exploration algorithm would take samples from the cliques in descending order of their sizes. Observe that a truncated lookahead of k steps is only able to distinguish the value between cliques of size less than k. All cliques of size k or greater will look equal to the truncated look ahead algorithm. Such an algorithm will explore somewhat randomly until there are only cliques of size smaller than k left and suffer poor performance as a result. Our proposed IM, however, will exactly give all the nodes scores in proportion to their clique's size and it will make good exploration choices from the beginning.

3.4 Computational Issues

Evaluation of the selection criterion requires repeated conditioning on single new label observations, which would require $O(n^3)$ time if we apply eq. 1 naively. Here we show two methods to make this computationally more efficient.

3.4.1 *Efficient matrix inverse updates*

We can reduce the computation by following the efficient update procedure suggested in [12]. When we add only one label to the graph, only one row of matrix A and only one entry in the diagonal matrix D' will be affected. Denote the original matrix inverse as $\Delta^{-1} = (I - A)^{-1}$, and the new inverse after one row is changed as $(\Delta')^{-1} = (I - A')^{-1}$.

According to the matrix inversion lemma, the new inversion $(\Delta')^{-1}$ is given by:

$$\begin{aligned} (\Delta')^{-1} &= (\Delta + (1-r)ee^\top A)^{-1} \\ &= \Delta^{-1} - \frac{\Delta^{-1}[(1-r)ee^\top A]\Delta^{-1}}{1 + (1-r)e^\top A \Delta^{-1} e} \\ &= \Delta^{-1} - \frac{(1-r)\Delta^{-1}_{(:,i)} A_{(i,:)} \Delta^{-1}}{1 + (1-r) A_{(i,:)} \Delta^{-1}_{(:,i)}} \end{aligned} \tag{4}$$

where e is a column vector with all entries 0 except the ith entry set to 1. Here we use $(i,:)$ to represent the ith row of the matrix, $(:,i)$ to represent the ith column, and r denotes $(1+\omega_0)(1-\eta)$. If we precompute Δ^{-1}, each time we add a label in the graph, it takes $O(n^2)$ to get the new inversion $(\Delta')^{-1}$, where n is the number of nodes in the graph. We can also efficiently update f after querying node i, when we get its label $I(y_i = 1)$. To update f, we first have the following equations (denote $s = \eta I(y_i = 1) - \frac{\omega_0 \pi}{1+\omega_0}$):

$$\left\{ \begin{array}{l} f = (I - A)^{-1} D' x \\ f' = (I - A')^{-1}(se + D'x) \end{array} \right.$$

Hence we can update f by:

$$\begin{aligned} f' &= (I - A')^{-1}(se + D'x) \\ &= (\Delta^{-1} - \frac{(1-r)\Delta^{-1}_{(:,i)} A_{(i,:)} \Delta^{-1}}{1 + (1-r) A_{(i,:)} \Delta^{-1}_{(:,i)}})(se + D'x) \\ &= f + \frac{s - (1-r)(f_i - \frac{\omega_0 \pi}{1+\omega_0})}{1 + (1-r) A_{(i,:)} \Delta^{-1}_{(:,i)}} \Delta^{-1}_{(:,i)} \end{aligned} \tag{5}$$

Using facts like $f = \Delta^{-1} D' x$, $f_i = A_{(i,:)} f + \frac{\omega_0 \pi}{1+\omega_0}$, and $A_{(i,:)} \Delta^{-1}_{(:,i)}$ is a 1×1 scalar so we can change the order of multiplications. Note here the denominator only multiplies the ith row of matrix A and the ith column of matrix Δ^{-1}, which only takes $O(n)$. Then the equation only involves a column vector $\Delta^{-1}_{(:,i)}$ multiplied by a constant and addition of column vectors, which also takes $O(n)$. Similarly, we can compute the impact factor efficiently after assigning a target label to each node we want to query, where the overall cost equals to querying all $O(n)$ unlabeled examples, which is still $O(n^2)$. The complete algorithm is shown in Algorithm 1. In order to make the computation less expensive, [3] may offer an even faster alternative, but it is only an approximation to the Cortes model [2].

3.4.2 *Efficient updates using label propagation*

When the graph is very large, even the efficient updates of sec. 3.4.1 may not help because computing the original

Algorithm 1 Active Search on Graphs

```
Input: $\omega_0, \eta, \pi$, precomputed $\Delta^{-1}$, budget.
Initialize the graph with one target and set its index to $bestInd$, initialize all entries in $f^{(0)}$ with $\pi$. Update labeled set $L^{(0)} = \{bestInd\}$ and unlabeled set $U^{(0)} = \{1, \cdots, n\} \setminus L^{(0)}$, $t = 1$.
repeat
    Recompute $f^{(t)}$ using Eq. 5, where $i = bestInd$;
    Recompute the new inversion $(\Delta^{(t)})^{-1}$ using Eq. 4, where $i = bestInd$;
    Compute $f'$ for each index $i \in U^{(t)}$ using Eq. 5 with $(\Delta^{(t)})^{-1}$;
    Compute the impact factor using Eq. 3 with $f'$;
    Select $j$ with the highest score using Eq. 2 with $f^{(t)}$ and the impact factor;
    Query $j$ and set $bestInd = j$, update $L^{(t+1)} = L^{(t)} \cup j$, $U^{(t+1)} = U^{(t)} \setminus j$, $t \leftarrow t + 1$;
until number of query equals to the budget
```

inverse and/or storing the updated inverses is not realistic. For those cases, we propose label propagation techniques to compute the proposed methods in an accurate manner.

First under the soft-label model with prior information, the result is achieved by: $f = [A\ D'][f\ x]^\top = P[f\ x]^\top$, which gives $f = (I - A)^{-1}D'x$. Instead of doing matrix inversion, we can initialize vector f with all zeros and multiply the matrix P iteratively to vector $[f\ x]^\top$, and only update the entries in f until convergence.

To compute the impact, we need some approximation, and there are two ways of achieving that. Suppose node i is the node that we attach a pseudo label to, and we focus on the tth iteration.

The first approach is to recompute f' after adding a node with label 1. As before we have: $f' = P'[f'\ x']^\top$, where x' is obtained by changing x's ith entry from π to 1, P' is obtained by changing one row of matrix $P = [A\ D']$, D'_{ii} is changed from $\frac{\omega_0}{1+\omega_0}$ to η, and the ith row of A is changed from $\frac{1}{1+\omega_0}D_{ii}^{-1}W_{(i,:)}$ to $(1-\eta)D_{ii}^{-1}W_{(i,:)}$. Since only one row in P is changed, if we initialize f' with the existing value of $f^{(t)}$, f' will converge to the correct value after a small number of iterations of re-multiplying $[f^{(t)}\ x']^\top$ by P'.

The second approach is cheaper but less accurate. The idea is to compute the impact on its immediately connected neighbors after assigning an unlabeled node with label 1. In the first step, given the change of one row in P, we have:

$$\begin{cases} f_i^{(t)} = \dfrac{1}{1+\omega_0}D_{ii}^{-1}W_{(i,:)}f^{(t)} + \dfrac{\omega_0\pi}{1+\omega_0} \\ f'_i = (1-\eta)D_{ii}^{-1}W_{(i,:)}f^{(t)} + \eta \end{cases}$$

which results in

$$f'_i = (f_i^{(t)} - \frac{\omega_0\pi}{1+\omega_0})(1+\omega_0)(1-\eta) + \eta$$

In the second step, this change will propagate to node i's immediate neighbors. We can compute this change by:

$$IM_i^{(t)} = \sum_{j \in \{U^{(t)} \setminus i\}} P'_{ji}(f'_i - f_i^{(t)})$$

To be more accurate we may compute the impact propagated not only to the immediate neighbors, but also to neighbors within two hops or even more.

3.5 Algorithm parameters

The jump-to-label probability η depends on how we think each unlabeled node relates to the labeled nodes nearby. If we set $\eta = 1$ then the soft-label model without prior will degenerate to the harmonic function model [11]. Varying η is in some sense similar to varying the parameter k in a KNN model. If we are confident that the label of each unlabeled node should just depend on the nearest labeled node, then it is reasonable to use a larger η. However, if we would like to consider more nearby labels, we use a smaller η. In our experiments we set $\eta = 0.5$ and did not vary it. An alternative would be to use cross validation on a separate graph to find a good value.

In our experiments we found that the values of ω_0 and π do not affect the results much. However, the existence of ω_0 is crucial because without this parameter, matrix $(I - A)$ can be non-invertible. The value of α has a large impact on performance because it controls the exploration/exploitation tradeoff. In our experiments we show results for a wide range of values. In the future work we discuss ideas for setting this parameter automatically.

4. EXPERIMENTAL RESULTS

Data. We demonstrate our approach on three real-world datasets.

The first dataset is a citation network with 14,117 nodes (papers) and 42,019 edges from citeseer, consisting of papers from the top 10 venues in Computer Science. The corresponding weight matrix has entry 1 if there is a citation link between two papers (undirected). The 1844 NIPS papers are labeled as targets.

The second dataset consists of 5271 webpages related to Programming Languages from Wikipedia. The corresponding weight matrix has entry 1 if the two webpages i and j are linked together (also undirected). For each webpage we precompute its topic vector using the software available at [7]. We label webpages with topic "object oriented programming" and related terms "object type class objects types classes method code languages programming", etc. We set the threshold to be 0.4 to get a reasonable number of targets (202).

The third dataset is a graph built from 5000 concepts in the dbpedia[1] ontology marked as "populated places". Each concept is a node in the graph and is backed by a Wikipedia page. We added an undirected edge between two places if one of their corresponding Wikipedia pages links to the other. The dbpedia ontology further divides populated places into "administrative regions", "countries", , "cities", "towns" and "villages"; these five labels serve as class labels. 725 nodes labeled as "administrative regions" are chosen as our targets.

Random subsets of the graphs are shown in fig. 2. The three graphs show quite different structures and distributions of positive nodes. The citation graph has many small connected components, and the positive nodes are present in different connected components. The wikipedia graph has large hubs and cliques, and the positive nodes are mainly concentrated in one large component. The populated-place graph, however, is in between these extremes. It has large hubs and cliques, but also many small connected components, and the positives are also present in many different

[1]www.dbpedia.org

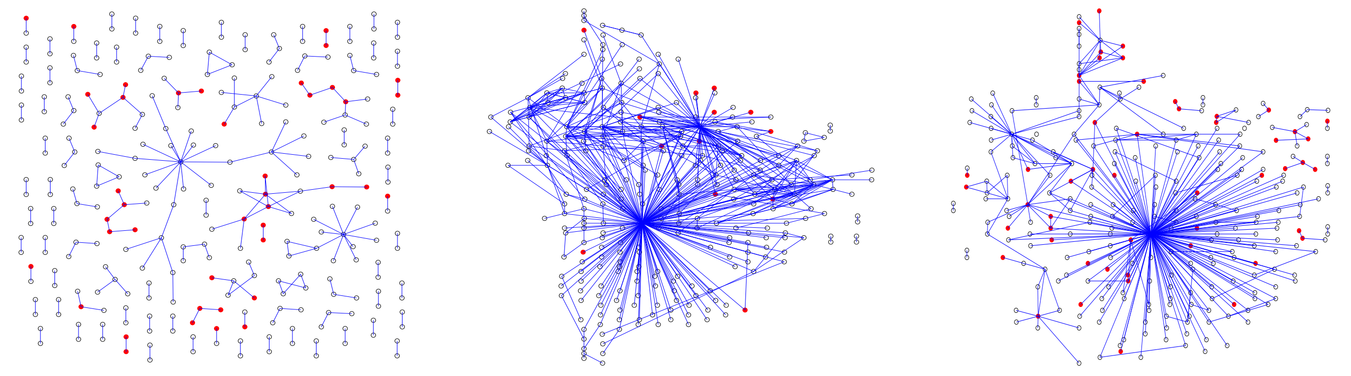

Figure 2: A random subset of the citation network (left), the wikipedia (middle), and the populated-place graph (right) with target nodes shaded solid in red. Each graph demonstrates a different structure and different distribution of positive nodes, which makes the active search task qualitatively different.

connected components. This makes the active search task qualitatively different on the three graphs.

Baselines. We compare our approach with several baselines.

1. Uncertainty Sampling. We use our proposed f function as an indicator of $P(y_i = 1|L)$. Under uncertainty sampling we query the node with f value closest to 0.5.

2. Modified Upper Confidence Bound. UCB is not a natural fit but we modify UCB1 proposed in [1]. We assume that at first each node has been 'pulled' once and the prior is the information we get. We use our proposed f as the current estimation $\overline{x}_j$, and count the number of queried neighbors of node j as n_j.

3. 2-step lookahead from [5]. Longer look aheads are too expensive to carry out.

4. Harmonic Function. We compute the f_u for unlabeled nodes as proposed in [11] and select the highest f_u as our next query.

Experimental Setting. We perform 10 random trials of all methods using a single randomly chosen positive node to initialize each trial. We record the number of positives found as a function of iteration number and average over the 10 trials. We set $\eta = 0.5$ for both datasets, π is set to the true prior proportion of positives, and ω_0 is set to $1/n$, where n is the number of nodes. We test $\alpha = \{0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$.

Results. Figure 3 shows the performance of our proposed model (with its best value of α) compared with the baselines on the three datasets. On all three datasets our proposed method is consistently better than all other baseline methods. The closest competitor is the harmonic function on the citation and wiki dataset, while on the populated-place dataset, the 2-step lookahead method is the second best one. Our experiments show that the difference between our method and the closest competitor is statistically significant ($p < 0.05$ in a paired t-test) after roughly 1300 iterations in the citeseer data, 200 iterations in the wikipedia data, and 10 iterations in the populated-place data.

We also notice uncertainty sampling doing as well as second best in some places. This is because it is built on our soft-label model and the number of positives is very small. Therefore, it imitates our proposed method with $\alpha = 0$ (the score falls below 0.5 after a certain number of iterations, hence picking the node with score closest to 0.5 is equivalent to picking the node with the highest score).

On the citation network, the gain of our proposed algorithm is quite substantial, with only 133 positives missed compared to 328 for the next nearest competitor, a 2.5-fold reduction. We have analyzed individual runs and observed that our method will effectively choose nodes in larger components and more connected portions of the graph first, which corresponds to a larger future gain. We also observe that our method effectively resolves the hub-blocking problem (Sec. 3.2), which results in a better performance compared to the harmonic function.

The gain on the wiki data is smaller, though we again have the best performance. We select this dataset because it is a different type of graph, which consists of a large connected component with large hubs containing most of the positives and some small components (mostly negative). The only opportunity for better performance comes while exploiting the large connected component, which is why we see significantly better performance at iteration 300. After that, all algorithms will complete the large component and be forced to search the small ones at random, hence the curves come together.

On the populated-place dataset, the gain of our proposed algorithm is even more substantial. We notice a large gain from the very beginning, and also through all the iterations till the end. When there are many small connected components with positive nodes in them, it is easier to discover those positive nodes first, hence harmonic function tends to exploit this information and repeat the procedure of exhausting the small components at first. However, our algorithm is more efficient at discovering the targets in the large component at the very beginning, thus causing a large initial deviation and higher future gain. Then our algorithm will only turn to other places when it finds enough negatives in this large component. This means our algorithm can effectively explore the graph from the more "positive" parts/clusters, to the less "positive" ones.

The right figures in Figure 3 show the more detailed results of carrying out paired t-tests among some of the competitive methods. We use the harmonic function as the baseline for comparison. The y-axis represents the difference

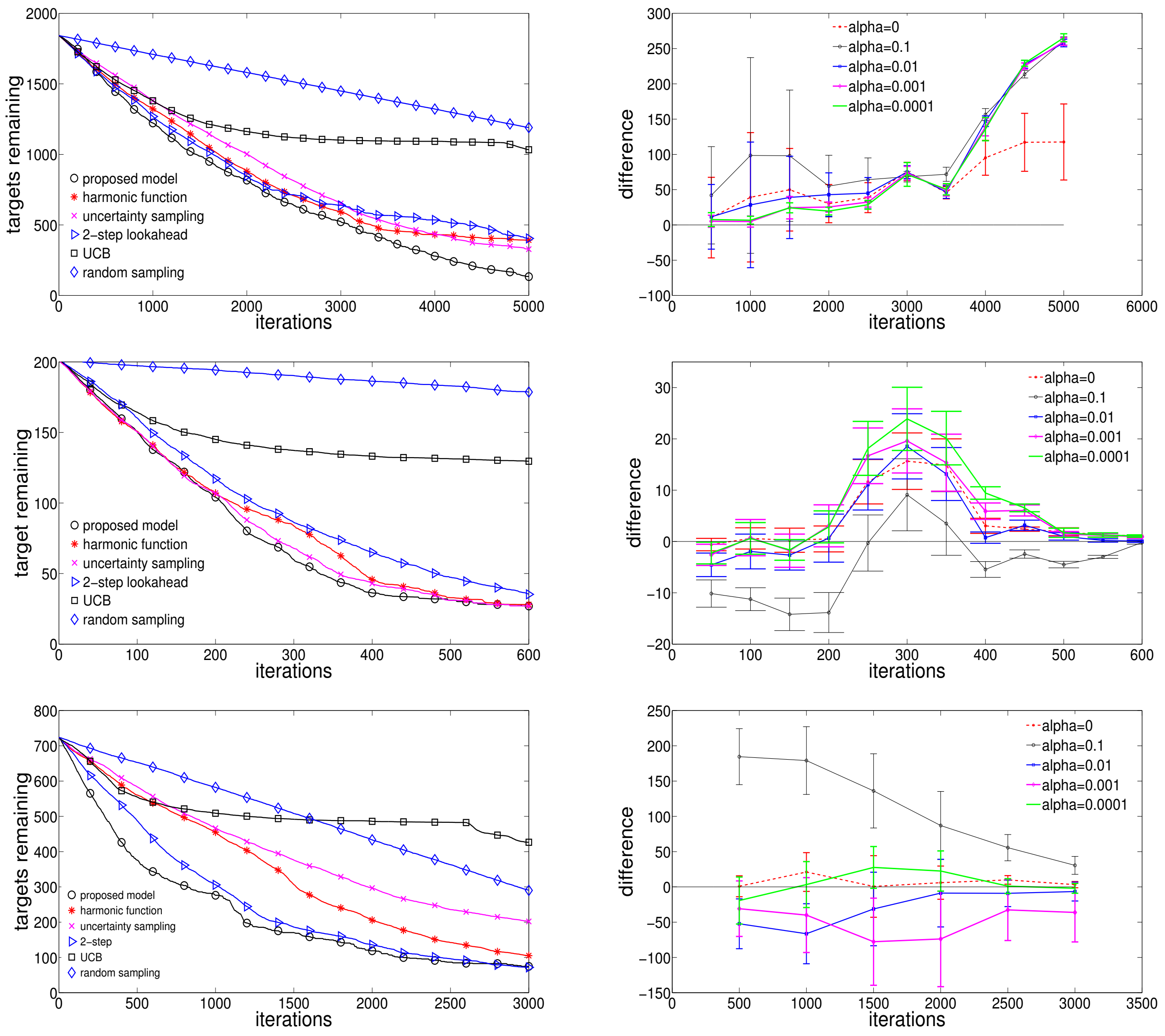

Figure 3: (Left) Total positives remaining by Proposed Model vs. Harmonic Function, Uncertainty Sampling, Upper Confidence Bound, Random Sampling, 2-step lookahead on the citation network (top), wikipedia dataset (middle), populated-place dataset (bottom). (Right) Difference in the number of positives found by (1) Proposed model with different $\alpha >= 0$ vs. (2) Harmonic Function (plotted as the zero line).

Table 1: Positives remaining (percentage) along with the number of iterations by our proposed method, four baselines and random sampling in the citation network, wikipedia dataset, and the populated-place dataset. The results are the averages and standard errors from 10 random trials.

Dataset	Citation Network						
Iterations	100	300	500	1000	2000	3000	5000
Proposed	97.5 ± 0.0	90.6 ± 0.0	81.9 ± 0.0	66.3 ± 0.0	44.3 ± 0.0	28.3 ± 0.0	7.2 ± 0.0
Harmonic	96.6 ± 0.2	89.9 ± 0.7	84.1 ± 1.1	71.7 ± 2.3	47.7 ± 0.9	32.1 ± 0.1	21.3 ± 0.1
2-step	96.7 ± 0.2	89.6 ± 0.1	83.2 ± 0.2	68.8 ± 0.1	45.9 ± 0.1	34.7 ± 0.1	21.9 ± 0.1
Uncertainty	96.9 ± 0.2	91.5 ± 0.6	87.0 ± 0.8	74.9 ± 0.9	54.4 ± 0.7	35.4 ± 0.6	17.8 ± 0.2
UCB	96.9 ± 0.2	90.9 ± 0.7	85.3 ± 0.9	74.8 ± 1.1	63.0 ± 0.3	59.7 ± 0.6	56.0 ± 0.5
Random	99.2 ± 0.0	97.8 ± 0.1	96.3 ± 0.1	92.7 ± 0.1	85.7 ± 0.2	78.6 ± 0.2	64.5 ± 0.2

Dataset	Wiki Dataset				Populated Places				
Iterations	100	300	500	1000	100	300	500	1000	3000
Proposed	74.4 ± 1.0	29.6 ± 1.0	15.4 ± 0.0	10.4 ± 0.0	89.0 ± 0.0	68.1 ± 0.0	51.8 ± 0.1	38.1 ± 0.0	10.3 ± 0.0
Harmonic	74.7 ± 1.0	41.4 ± 1.0	16.3 ± 0.0	11.4 ± 0.0	94.3 ± 1.3	85.7 ± 2.1	77.3 ± 2.4	62.8 ± 2.9	14.5 ± 0.8
2-step	79.3 ± 0.9	42.7 ± 0.4	23.4 ± 0.2	12.8 ± 0.0	92.9 ± 0.0	79.1 ± 0.0	67.3 ± 0.0	42.2 ± 0.0	9.9 ± 0.0
Uncertainty	74.4 ± 1.0	33.6 ± 1.0	15.3 ± 0.0	10.3 ± 0.0	94.6 ± 1.1	88.2 ± 1.2	80.5 ± 1.0	64.3 ± 1.3	27.9 ± 2.7
UCB	81.4 ± 1.0	67.9 ± 0.0	65.0 ± 0.0	61.6 ± 0.0	95.8 ± 1.0	84.7 ± 1.6	76.5 ± 1.3	70.2 ± 1.1	58.8 ± 0.7
Random	97.7 ± 0.0	94.2 ± 0.0	90.6 ± 1.0	81.1 ± 1.0	97.8 ± 0.1	93.5 ± 0.3	90.0 ± 0.3	80.3 ± 0.3	40.0 ± 0.4

in the number of targets found by our model with varying $\alpha = \{0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$, compared to the harmonic function.

Table 1 further show the percentage of positives missed by our proposed method and baselines quantitively, with respect to the number of iterations (queries).

5. CONCLUSIONS AND FUTURE WORK DISCUSSION

In this paper, we present a soft-label model for graphs that extends previous random walk style models and give efficient methods of conditioning these models. We propose an impact factor to be used as a criterion for node selection. The impact factor plays the role of encouraging exploration, which is often done in other settings using entropy, uncertainty, or variance. We point out that those concepts are not suitable for active search however, and show empirically that we achieve better performance using our proposed method.

Setting α remains an unresolved issue. An automated method might consider a budget, B, of remaining choices to be made and set it accordingly. A reasonable setting might be $\alpha \sim (B - |L|)/B$, where $|L|$ is the size of labeled set. In this way, as the size of the labeled set increases, α will automatically decrease, corresponding to the natural strategy that in the beginning we want to explore more as we have more budget, and later we want to focus on exploitation.

Alternatively, we might follow the form of the UCB algorithms and determine an adaptive value of α that allows us to derive regret bounds. However, doing so may be challenging given the fact that UCB methods are based on repeated pulls on "best arms", while active search does not allow same node selection.

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. In *Machine Learning*, 2002.

[2] C. Cortes, M. Mohri, D. Pechyony, and A. Rastogi. Stability of Transductive Regression Algorithms. In *ICML*, 2008.

[3] R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised Learning in Gigantic Image Collections. In *NIPS*, 2009.

[4] A. Garivier and O. Cappe. The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond. In *COLT*, 2011.

[5] R. Garnett, Y. Krishnamurthy, X. Xiong, J. Schneider, and R. Mann. Bayesian Optimal Active Search and Surveying. In *ICML*, 2012.

[6] A. Kuwadekar and J. Neville. Relational Active Learning for Joint Collective Classification Models. In *ICML*, 2011.

[7] A. McCallum. Mallet: A machine learning for language toolkit. mallet.cs.umass.edu, 2002.

[8] A. Slivkins. Contextual Bandits with Similartiy Information. In *COLT*, 2011.

[9] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *ICML*, 2010.

[10] M. Szummer and T. Jaakkola. Partial Labeled Classification with Markov Random Walks. In *NIPS*, 2001.

[11] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML*, 2003.

[12] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining Active Learning and Semi-supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.