Bayesian Optimal Active Search on Graphs

Roman Garnett, Yamuna Krishnamurthy,
Donghan Wang, Jeff Schneider
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213
{rgarnett, ykrishna}@andrew.cmu.edu
{dwang, schneide}@cs.cmu.edu

Richard Mann
Uppsala Universitet
Uppsala 751 06
Sweden
rmann@math.uu.se

ABSTRACT

In many classification problems, including numerous examples on modern large-scale graph datasets, a large quantity of unlabeled data are available. The cost of obtaining a label for such data can be very expensive, for example when human intervention is required. Both the semi-supervised and active learning communities have approached such problems. The former focuses on how to aid the classification task by exploiting the distribution of unlabeled data, and the latter addresses the problem of choosing the most useful labels to acquire, that would subsequently minimize the cost incurred in the pursuit of the learning goal. Various algorithms have been proposed by these communities that learn from a large dataset with few labeled data, and graph datasets have in particular received a lot of attention for large-scale applications such as collaborative filtering for recommendation systems and link prediction in social networks.

Here, we focus on a specific active binary-classification problem, where the goal is to find the members of a particular class as quickly as possible. In some situations, such as fraud detection or the investigative analysis of potentially criminal social networks, only the members of the malicious class are sought, whereas obtaining labels for points in the positive class is only useful for the purpose of facilitating that task. We derive the Bayesian optimal policy for this decision problem and test our proposed algorithm on two large-scale graph datasets. The optimal policy can be implemented in parallel, and it is our hope that the described algorithm can scale to even larger graphs.

Categories and Subject Descriptors

I.5.1 [**Computing Methodologies**]: Pattern Recognition-Models

Keywords

machine learning, graph learning, active learning, semi-supervised learning, Bayesian decision theory

MLG '11 San Diego, CA, USA

1. INTRODUCTION

Current real-world machine learning problems are often posed on large graphs, such as web graphs and other communication graphs, including social networks. In this manuscript, we will consider binary-classification on graphs. In many scenarios, it is much easier to collect input data than it is to observe an associated label, which could require a relatively expensive human action. For this reason, considerable research in semi-supervised and active learning has focused on how to construct useful probabilistic models when unlabeled data are available, as well as how to solve the associated decision problem of determining which labels are the most useful to observe for the chosen learning objective.

Active learning research has often focused on the problem of obtaining those labels that will improve overall classification accuracy as much as possible. Here, we consider a different problem, where the members of one particular class are deemed critical and are to be located as quickly as possible. An example application of this sort is locating malicious actors in a social or computer network, where finding the interesting points is more important than secondary measures such as classification accuracy. We call this problem the *generalized Battleship* problem, in reference to the well-known board game, which has gameplay that takes the same form.

We will treat this problem in the context of Bayesian decision theory and derive the Bayesian optimal policy for an appropriate utility function. The decision analysis does not depend on the nature of the underlying classification problem; in particular, the input space and associated probabilistic model are not constrained. In this manuscript, however, we will consider the problem from the point of view of graph learning. We will briefly review and discuss how to construct useful models that can exploit the graph structure of a given dataset. In particular, we discuss the computation and use of similarity measures between the nodes of the graph, based on random walks. We argue that measures based on the graph Laplacian, including commute time, truncated commute time, and in particular the GRANCH algorithm can be useful in this context.

1.1 Paper Outline

The rest of this paper is arranged as follows. In Sections 2 and 3, we formally describe the generalized Battleship problem and related work. In Section 4, we provide the Bayesian optimal policy for this problem and analyze its complexity. In Section 5, we will discuss the construction

of useful probabilistic models for the classification portion of the problem when the input space is in the form of a graph. Finally, we will present the results of evaluating our algorithm on two large real-world graphs.

2. PROBLEM DEFINITION

Suppose we have a finite set of n elements $\mathcal{X} \triangleq \{x_i\}_{i=1}^n$ and a subset of m elements $\mathcal{X} \supset \mathcal{T} \triangleq \{t_i\}_{i=1}^m$, which we will call *targets*. We consider the following problem. Suppose we do not know which members of $\mathcal{X}$ are targets *a priori*, but can successively request the binary $\{0, 1\}$ response to questions of the form "is x in $\mathcal{T}$?" for arbitrarily selected elements x. We wish to proactively select a sequence of queries $\{q_1, q_2, \dots\}$ with the goal of finding the elements of $\mathcal{T}$ as quickly as possible, that is, in the least number of queries.

We will approach this task using Bayesian decision theory. This will require selecting a classification model that provides the posterior probability of a point x belonging to $\mathcal{T}$ conditioned on previously observed data $\mathcal{D}$,

$$p(x \in \mathcal{T} \mid x, \mathcal{D}).$$

It will be mathematically convenient for us to define this problem equivalently in terms of a binary class label y, which we will define to be the value of the characteristic function $\chi(x \in \mathcal{T})$. In the applications we consider, there will often be a notion that members of $\mathcal{T}$ should cluster together under some measure of similarity on $\mathcal{X}$, which will influence the choice of this probabilistic model. We will discuss how we can effectively address this intuition for problems where $\mathcal{X}$ is a graph later.

An atypical real-world problem that could be treated under this framework is the Milton Bradley board game "Battleship," where two players alternate firing "shots" to determine the presence of a "ship" in an identified point of a 10×10 lattice. Before the game begins, each player places five ships on their board, hidden from the other player, each of which occupies two-to-five contiguous points on the lattice. In this case, $\mathcal{E} = [10] \times [10]$, $n = 100$, $m = 17$ (the total number of squares occupied by ships), $\mathcal{T}$ identifies the *a priori* unknown locations of the opponent's ships, and keen players evaluate their beliefs about ship placement before deciding the locations of their shots.

3. RELATED WORK

Active learning is a mature field with a large associated body of literature [15]. The basic premise of active learning is that if the cost of obtaining a label is very high, then we can hope to achieve our learning objective with less overall cost by taking control of the labeling process. Typically, in the active binary-classification problem, the chosen learning objective is related to properties of the associated probabilistic model. Two examples include generalization error [22] and optimality criteria related to the Fisher information, for example, A-optimality [14]. One of the simplest active learning techniques for binary classification is *uncertainty sampling* [8], where at any given point, the algorithm requests the label for the point x^* with the greatest posterior uncertainty:

$$x^* \triangleq \arg\min_x \left| p(y = 1 \mid x, \mathcal{D}) - 1/2 \right|.$$

For binary classification, this is equivalent to choosing the point that maximizes information gain.

An intimately related problem to active learning is semi-supervised learning [20], which studies how to use the distribution of unlabeled data to improve model accuracy. This is often a critical problem in active learning due to the high cost of labeling and consequent abundance of unlabeled data.

The game of Battleship itself has been subject to some research as well. In [10], integer programming models were described to solve static logic puzzles based on the game. It has been also shown that a particular decision problem[1] related to the game is NP-complete [16].

The objective considered in this paper, locating members of an identified class, is unusual as far as the authors know. A similar problem that has been considered is the active discovery of examples of previously unseen classes [6]. Additionally, it has been noted that standard active learning techniques can help uncover relatively balanced training sets, even with significant imbalance between class proportion in the data [3]. This result suggests that there is promise in active approaches to the problem we consider, because previously described techniques can already help to uncover more members of a designated class (targets) than would be found by random search. However, our experimental results show that the formal treatment of the exact problem defined here performs considerably better than uncertainty sampling.

Here, we will eschew any consideration of classification or model accuracy completely and instead measure the success of our choice of queries based purely on the overall number of targets uncovered by our search. From a decision-theoretic perspective, the generalization error or variance of our model has no impact on our outlined goal and therefore should not play a role in our choice of policy.

4. THE OPTIMAL BAYESIAN POLICY

As stated above, we will analyze the generalized Battleship problem from the perspective of Bayesian decision theory and will derive the optimal policy. We begin by defining an appropriate utility function that captures the ultimate goal of the problem.

Define the utility of a set of observations $\mathcal{D} \triangleq \big\{(x_i, y_i)\big\}$ to be the number of inputs x_i seen with associated class label $y_i \in \{0, 1\}$ equal to 1 (equivalently, the number of targets found, $\{x_i\} \cap \mathcal{T}$):

$$u(x_1, x_2, \dots, y_1, y_2, \dots) \triangleq \sum_i y_i.$$

This simple expression naturally captures the spirit of the problem as defined above.

Without loss of generality, we will assume that at the onset we will only be allowed a fixed number of queries t; note that by setting $t \triangleq n$, we recover the most-general problem. In applications where the cost of obtaining a label is high, it is

[1]Here "decision problem" is used in the context of computational complexity.

the total cost of the queries that limits their number, rather than the quantity of available unlabeled points.

We must now derive the correct policy for successively deciding the locations of our queries. We begin by considering the case when we are allowed to make exactly one more query and will then address the general case.

Suppose that we have already made $t-1$ observations $\mathcal{D}_{t-1}$. To select the location of our final observation x_t, we calculate the expected utility of a candidate point x_t, marginalizing out the unknown value of y_t:

$$\begin{aligned}\mathbb{E}\big[u(x_t, y_t, \mathcal{D}_{t-1}) \mid x_t, \mathcal{D}_{t-1}\big] \\ &= \int u(x_t, y_t, \mathcal{D}_{t-1}) p(y_t \mid x_t, \mathcal{D}_{t-1}) \,\mathrm{d}y_t \\ &= u(\mathcal{D}_{t-1}) + p(y_t = 1 \mid x_t, \mathcal{D}_{t-1});\end{aligned}$$

the optimal decision x_t^* is therefore simply the point with the highest posterior probability of being a target:

$$x_t^* \triangleq \arg\max_{x \notin \mathcal{D}_{t-1}} p(y = 1 \mid x, \mathcal{D}_{t-1}). \tag{1}$$

This fact makes intuitive sense: when we have only one query remaining, there is no point in considering any potential future strategic value in its location; rather, we are compelled to greedily seek one final target.

Given the optimal policy for selecting x_t, we now consider the problem of choosing the location of the second-to-last point x_{t-1}. While making our decision in this case (as well as with any other x_i with $i < t$), the problem becomes more difficult because we must now contemplate the possible consequences of our choices and how they will impact our future decisions. The mechanical manifestation of this remark is that during the calculation of the expected utility for the two-step-lookahead case, we must integrate out the unknown location of the final observation x_t, as well as its label:

$$\begin{aligned}&\mathbb{E}\big[u(x_{t-1}, y_{t-1}, x_t, y_t, \mathcal{D}_{t-2}) \mid x_{t-1}, \mathcal{D}_{t-2}\big] \\ &\quad= u(\mathcal{D}_{t-2}) + \\ &\qquad+ \int u(x_{t-1}, y_{t-1}, x_t, y_t) p(y_{t-1} \mid x_{t-1}, \mathcal{D}_{t-2}) \times \\ &\qquad\qquad\times p(x_t \mid \mathcal{D}_{t-1}) p(y_t \mid x_t, \mathcal{D}_{t-1}) \,\mathrm{d}y_{t-1} \,\mathrm{d}x_t \,\mathrm{d}y_t \\ &\quad= u(\mathcal{D}_{t-2}) + \\ &\qquad+ 2p(y_t^* = 1 \mid x_t^*, y_{t-1} = 1, \mathcal{D}_{t-1}) \times \\ &\qquad\qquad\times p(y_{t-1} = 1 \mid x_{t-1}, \mathcal{D}_{t-2}) \\ &\qquad+ \;\; p(y_t^* = 0 \mid x_t^*, y_{t-1} = 1, \mathcal{D}_{t-1}) \times \\ &\qquad\qquad\times p(y_{t-1} = 1 \mid x_{t-1}, \mathcal{D}_{t-2}) \\ &\qquad+ \;\; p(y_t^* = 1 \mid x_t^*, y_{t-1} = 0, \mathcal{D}_{t-1}) \times \\ &\qquad\qquad\times p(y_{t-1} = 0 \mid x_{t-1}, \mathcal{D}_{t-2}) \\ &\quad= u(\mathcal{D}_{t-2}) + \\ &\qquad+ \Big(\mathbb{E}\big[u(x_t^*, y_t^*, \mathcal{D}_{t-1}) \mid y_{t-1} = 1, \mathcal{D}_{t-1}\big] + 1\Big) \times \\ &\qquad\qquad\times p(y_{t-1} = 1 \mid x_{t-1}, \mathcal{D}_{t-2}) + \\ &\qquad+ \;\; \mathbb{E}\big[u(x_t^*, y_t^*, \mathcal{D}_{t-1}) \mid y_{t-1} = 0, \mathcal{D}_{t-1}\big] \times \\ &\qquad\qquad\times p(y_{t-1} = 0 \mid x_{t-1}, \mathcal{D}_{t-2}), \end{aligned} \tag{2}$$

where the integral over x_t was evaluated trivially because $p(x_t \mid \mathcal{D}_{t-1})$ is simply $\delta(x_t - x_t^*)$,[2] where δ is the Dirac delta function.

To evaluate the two-step expected utility at a point x_{t-1}, we therefore sample over the unknown value $y_{t-1} \in \{0, 1\}$; for each possible value of y_{t-1}, we find the optimal last observation x_t^* given that fictitious observation as described above. There are four possibilities for the possible realizations of (y_{t-1}, y_t^*); two of these increase our current utility by one, and one increases the utility by two, giving rise to (2). Note that sampling over y_t^* is not required, because the expected utility in the one-step case has already been evaluated. Of course, to select x_{t-1}, we again choose the currently unlabeled point with the highest expected utility.

To handle x_i for $i < (t-1)$, we simply repeat the procedure described above recursively. Pseudocode for the full ℓ-step lookahead procedure, with any number of remaining queries, is shown in Algorithm 1.

Algorithm 1: Function FIND-OPTIMAL-ACTION

input : data $\mathcal{D} \triangleq (\mathbf{x}, \mathbf{y})$, allowed number of shots ℓ, model $p(y = 1 \mid x, \mathcal{D})$
output: the Bayesian optimal action $x^* \in \mathcal{X} \setminus \mathbf{x}$

```
if ℓ = 1 then
    x* ← max_{x ∈ X \ D} p(y = 1 | x, D)
    return x*
else
    for x ∈ X \ x do
        utility(x) ← 0
        for y ∈ {0, 1} do
            utility(x) ← utility(x) + y·
            FIND-OPTIMAL-ACTION( D ∪ {x, y}, ℓ − 1)
        end
    end
end
x* ← max_{x ∈ X \ D} utility(x)
return x*
```

4.1 Running Time

The running time of Algorithm 1 may be calculated directly. Suppose there are n points to choose from at time $t - \ell$. Without any heuristics, performing exact ℓ-step lookahead requires considering all of the

$$\prod_{i=0}^{\ell-1} (n - i)$$

possible choices for the remaining ℓ points; for each of these choices we must also sample $2^{\ell-1}$ possible observations of $y_{t-\ell+1}, \ldots y_{t-1}$, as well as incrementally condition the model based on $\ell - 1$ fictitious observations. If that conditioning

[2]Notice that the value of x_t^* depends on the data $\mathcal{D}_{t-1}$, including the unknown value of y_{t-1}; it might differ as a function of that conditioning.

takes time c, then the total work required is

$$\frac{2^{\ell-1}(\ell-1)cn!}{(n-\ell)!} = \mathcal{O}\big(c\ell(2n)^{\ell}\big).$$

For lookahead more than a few steps into the future, this procedure can become daunting due to the sampling required. This is a common issue in fully optimal sequential Bayesian decision problems.

One possible and common mechanism for addressing this problem is instead approximating exact inference by shortening our lookahead horizon [7, 11]. For timestep $t-m$ with $m > \ell$, we may myopically pretend that there are in fact only ℓ observations remaining and choose x_{t-m} by maximizing the ℓ-step-lookahead expected utility. In nontrivial applications, we can often expect that our uncertainty about the future will increase dramatically with the number of steps we look ahead. In this case, such a myopic approach should be a reasonable approximation to the optimal policy, because this large uncertainty results in little gain from marginalizing distant future decisions. In Bayesian Gaussian process global optimization, a problem with a similar form, a one-step approximation is often used with great success [7]. In [11], the authors later described the optimal ℓ-step-lookahead procedure and showed that two-step lookahead can outperform this approach.

An alternative or potentially complementary approach is to restrict the search space as we look ahead. For example, suppose that our model giving $p(y = 1 \mid x, \mathcal{D})$ only considers those observations in $\mathcal{D}$ that are within the k nearest neighbors of x under some chosen similarity measure. In this case, the work required to perform ℓ-step lookahead reduces significantly. For example, we may perform two-step lookahead by precomputing the expected utility for one-step lookahead at every point. Now for a chosen candidate x_{t-1}, when sampling over y_{t-1}, we only need to condition the models for those points x that contain x_{t-1} as one of their nearest neighbors. finding x_t^* given y_{t-1} requires much less work, because for $k \ll n$, most of the available choices will already have the required one-step lookahead utilities precomputed. The average-case running time in this scenario now reduces to approximately $\mathcal{O}\big(c\ell(2k)^{\ell}\big)$, which is much more manageable and with hope allow for increasing ℓ beyond what was feasible before. Restricting to k-NN models also allows the algorithm to be easily parallelized and operate on very large datasets—the calculation of expected utilities is embarrassingly parallel and the restriction to local models makes each individual computation reletively fast.

5. GRAPH APPLICATIONS

The decision theoretic policy given above does not depend on the nature of the underlying classification problem. However, we will spend some time discussing how the algorithm can be applied to large-scale graph datasets.

As mentioned previously, a natural assumption to make when constructing the classifier $p(y = 1 \mid x, \mathcal{D})$ is that the target class clusters together under some metric or similarity measure. Here we agree with the conclusions of many previous research efforts[3] and promote the use of similarity measures based on random walks, which have performed well both in our experiments and numerous other graph-learning problems such as collaborative filtering and web clustering [18]. Additionally, it can be computed quickly enough to scale to very large graphs. In particular, we advocate the GRANCH algorithm [12, 13] for calculating approximated truncated commute times in directed graphs. GRANCH was effective in our experiments, and its efficiency and trivial parallelizibility should enable it to scale to graphs with millions of nodes.

The GRANCH algorithm approximates the *truncated commute time* between two nodes in a graph, which is defined in terms of simple Markovian random walks on the graph. We discuss briefly commute time and related notions below. Note that a great deal of existing research has promoted and evaluated the use of these and other related similarity measures [18].

5.1 Random Walk Similarity Measures

Let us begin by defining some notation. Let $G \triangleq (V, E)$ be a graph, where $V \triangleq \{v_i\}_{i=1}^n$ is a finite set of n vertices, and E is an antireflexive relation on $V \times V$ defining the edges of the graph. In the undirected case, E is a symmetric relation. Additionally, let $w \colon E \to \mathbb{R}$ be a nonnegative weight function on the edges of G; for unweighted graphs this function can be defined trivially by setting $w = 1$ when an edge is present and 0 otherwise. We will assume here that G is connected in the undirected case and strongly connected in the directed case. This assumption is for simplicity only; disconnected graphs can be handled easily if desired by considering the components separately.

Let v be an arbitrary node in G. Consider the following simple Markovian random walk on the edges of E. At time $t = 0$, we begin at v. We choose an edge e incident to v randomly with probability proportional to the total weight of all edges incident to v and "walk" to the other vertex incident with e. At time $t = 1$, we take another step of the walk by repeating this procedure from the new vertex and repeat *ad infinitum*. It is clear that the edge weights induce a probability distribution on the set of infinite random walks from v, as well as those walks beginning from any other vertex. Given a second node $v' \in V$, define the *hitting time* from v to v', $h(v, v')$ to be the expected time for a random walk from v to enter v' for the first time. Given this, we may define a symmetrized measure called *commute time*, $c(v, v')$, which is the expected number of steps a random walk from v takes to hit v' and then return. It is clear that $c(v, v') = h(v, v') + h(v', v)$. Commute time is a metric on V, as is its square root [4].

For any graph G we may calculate the commute time between any two vertices $v, v' \in V$ using a simple recurrence relation. Let $p(v, v')$ be the transition probability for walking from v to v' under the probability distribution induced by w. Then we have the following expression for hitting time, from which we can build commute times:

$$h(v, v') = \begin{cases} 0 & v = v' \\ 1 + \sum_{v'' \in V \setminus \{v\}} p(v, v'') h(v'', v') & \text{otherwise.} \end{cases} \tag{3}$$

[3]Dozens of references may be found in the very useful and impressively complete review in [18].

This relation is cumbersome and not terribly convenient for real applications. In the case of undirected graphs, however, we can calculate hitting and commute times exactly using the graph Laplacian.

5.1.1 Undirected Graphs

It has been shown [4, 1] that hitting and commute times can be calculated exactly for undirected graphs using the pseudoinverse of the graph Laplacian. Define the adjacency matrix $\mathbf{A}$ by setting $A_{ij} \triangleq w(v_i, v_j)$, and define the diagonal degree matrix $\mathbf{D}$ by $D_{ii} \triangleq \sum_j A_{ij}$.

The graph Laplacian is defined by $\mathbf{L} \triangleq \mathbf{D} - \mathbf{A}$.[4] $\mathbf{L}$ is a symmetric positive semi-definite (SPSD) matrix; it has an eigenvalue of 0 with multiplicity equal to the number of connected components of G. The Moore–Penrose pseudoinverse of $\mathbf{L}$, $\mathbf{L}^+$, is also a SPSD matrix, with properties intimately connected to the random walks described above. In particular, the entries of $\mathbf{L}^+$ can be understood as inner products of vectors in $\mathbb{R}^{n-1}$ that correspond to the vertices of G. Remarkably, the Euclidean distances between the vectors in that latent space are proportional to the square root of the exact commute times between the vertices [4].

Unfortunately, this relation is of limited practical value. To begin, it is only true for undirected graphs. More seriously, computing $\mathbf{L}^+$ is infeasible for n greater than a couple of tens of thousands of nodes in G, although both [4] and [1] propose computational schemes to aid the situation. Fouss, *et al.* [4] describe how to calculate a column of $\mathbf{L}^+$ "on demand" via a sparse Cholesky factorization of $\mathbf{L}$, and [1] proposes an iterative algorithm for finding commute times.

Still another alternative is to perform a sparse singular value decomposition of $\mathbf{L}$, retaining only singular vectors corresponding to the the smallest $m \ll n$ nonzero singular values to form a low-rank approximation to $\mathbf{L}^+$. This is in fact equivalent to performing principal component analysis (PCA) on the latent Euclidean vectors described above and projecting onto their first m principal components [4].

5.1.2 Directed Graphs

In the case of directed graphs (where $\mathbf{L}$ will not in general be SPSD), there is no simple analytic expression for hitting or commute times. Commute times can still be calculated in this case via (3); however, this expression is somewhat unwieldy and computationally burdensome. As a result, research involving these quantities tends to focus on undirected graphs, symmetrizing the adjacency matrix of directed graphs if need be. In many applications, the directed nature of the edges in a graph can be very informative; for example, the direction of hyperlinks in a web graph is an often highly asymmetric relation.

One reason for the focus on the undirected case is that by virtue of their being SPSD, $\mathbf{L}$ and $\mathbf{L}^+$ are valid kernel matrices, as are related quantities derived by manipulating their spectra [21]. This property naturally makes these quantities attractive due to the incredible amount of research devoted to kernel methods.

[4]Sometimes a normalized Laplacian is used instead, defined by $\mathbf{L} \triangleq \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

Considerably less research has investigated similar techniques for the case of directed graphs. Chung [2] studied the Laplacian of directed graphs in the context of random walks, and the ideas in that paper have been applied to machine learning problems [19].

Here we will use the GRANCH algorithm [12, 13], which provides a very fast mechanism for calculating *truncated commute times*, that is, commute times between nodes when the length of random walks is constrained to be of a given finite length. Sarkar and Moore argue that not only does the truncation of random walks allow for computational feasibility, but also that it helps overcome occasional issues with commute time [12]. For example, when using standard full commute times, nodes with very large degree can be "close" to a node of interest, even if all paths to that node are relatively long. This phenomenon has been noted recently and described in a theoretical context for random geometric graphs [17]. Truncating the walks helps to discourage this phenomenon.

GRANCH is designed to quickly find the nearest neighbors to a node in a graph under truncated commute time. It can scale to graphs with millions of nodes on a single machine,[5] and works correctly on directed graphs without modification. For these reasons, we selected it for the basis of our experiments.

6. RESULTS

To evaluate the policy described in Section 4, we created two binary classification problems on real-world graphs. In the applications we have envisioned, the target class will often be a somewhat small fraction of the overall dataset, and we kept this in mind during the creation of these experiments.

6.1 Implementation and Setup

We implemented Algorithm 1 recursively in MATLAB. For the probabilistic model $p(y = 1 \mid x, \mathcal{D})$, we chose to use a simple k-NN classifier, where $k = 200$. The 200 nearest neighbors for each node in each graph were obtained by querying GRANCH on the directed graph of interest.

We chose to use the simple k-NN classifier for several reasons. First, our goal in these experiments was to evaluate the decision algorithm presented above, and we did not want that evaluation to be confounded by any unrelated details of the classification mechanism. Secondly, our desire is that the methods in this paper can scale to very large graphs, and k-NN models allow for easy parallelism, as well as cost savings as described in Section 4.1. We also note that on some similar tasks, such as collaborative filtering, k-NN classifiers have achieved better success than more complex SVM techniques in some experiments [5].

6.2 CiteSeer^x Data

For our first experiment, we created a graph from a subset of the CiteSeer^x citation network. Papers in the database were grouped based on their venue of publication (after extensive data cleaning), and papers from the 48 venues with the most

[5]For a graph with approximately 1.7×10^5 nodes and 2.8×10^6 edges, GRANCH was able to complete a query for the 200 nearest neighbors to a node in the graph in 0.37 s on a machine with a 2.3 GHz AMD Opteron processor.

associated publications were retained. The graph was defined by having these papers as its nodes (42 133 in total) and directed citation relations as its edges.

For this experiment, we designated all papers appearing in NIPS proceedings (2 198 in total, 5.2% of the dataset) as targets. Locating these papers is a difficult task—among the top 10 most-frequent venues in the dataset are the highly related venues AAAI, IJCAI, and ICML, which together comprise 15.2% of the dataset, and there are more venues related to machine learning in the remaining 38 as well.

We ran 1 000 steps of both the one-step- and two-step-lookahead versions of the algorithm presented above. To facilitate computation, not every page was considered in the two-step algorithm; rather, we only calculated the two-step expected likelihood for a subset of the unlabeled pages. This subset was chosen to include the unlabeled pages among the 200 nearest neighbors of the last 10 targets found, the pages with the last 10 targets found as nearest neighbors, and additionally a randomly selected subset of the remaining nodes (comprising 1% of the remainder). This heuristically chosen subset was designed to attempt to enable both exploitation and exploration. Although the Bayesian optimal action might not be found among these points, we hope that at least one point will be "close enough" to the optimal action in expected utility that the overall performance will not be dramatically impacted.

We also compared our derived policy with a more typical active binary-classification algorithm (uncertainty sampling), where at each step, the point with the maximum uncertainty was queried.

Each algorithm was initialized with the label of a single randomly selected NIPS paper; all other papers were left unlabeled at the start of the experiment.

At completion, the one-step algorithm had found 226 NIPS papers, the two-step algorithm had found 251, and the uncertainty sampling algorithm had found 206. Random search would find an expected number of only 52 papers given the same number of shots. Figure 1 shows the cumulative number of targets found by each of the methods throughout the run of the experiment.

These results support the effectiveness of Algorithm 1. The two-step-lookahead procedure was able to find 11.4% of the targets after scanning only 2.2% of the data, five times better than would be expected by random search. The two-step-lookahead policy also outperformed the greedy one-step policy by a significant margin.

6.3 Wikipedia Data

We ran an additional experiment on a much larger dataset obtained from Wikipedia. The input space was Wikipedia pages extracted from the Computer Science category, as well as all its subcategories. In total, there were 171 236 documents in this dataset. Pages in the Programming Languages category or its subcategories (5 422 in total, 3.17% of the dataset) were marked as targets.

We repeated the same experiment described above for the

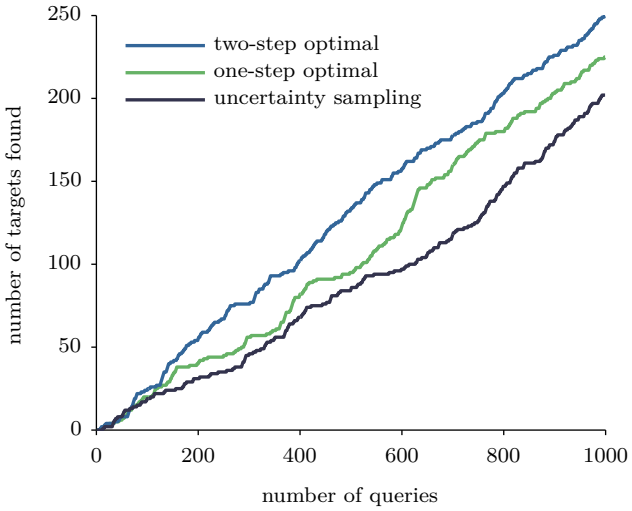

Figure 1: Cumulative number of targets found during 1 000 steps of several active querying schemes on the CiteSeer^x data.

CiteSeer^x data on this dataset, except we doubled the number of shots to 2 000. Out of these, the one-step lookahead procedure located 1 783 targets.

We interpret this result in two ways. The impressive performance of the greedy one-step-lookahead procedure on this task implies that perhaps the chosen classification problem was too easy—programming-language pages might simply cluster together strongly. On the other hand, we also note that GRANCH was highly effective in identifying this cluster, empirically confirming our assertion that it is useful for this purpose.

In light of these observations, we modified the problem setup to make it more difficult, while still providing a means of testing the our search policy. Rather than analyze the link graph of the Wikipedia pages, we used only their textual content for discriminating between classes.

To approach this modified problem, we trained a latent Dirichlet analysis (LDA) topic model from these documents using the MALLET software package [9]. The number of topics was set to 200. The default parameters for MALLET were used, except the hyperparameters of the LDA model were optimized during the learning process. Once the topic vector for each document was found, we measured the cosine similarity between these topic vectors and recorded the 200 most similar documents for each Wikipedia page, as well as the associated cosine similarities. The graph in this case was therefore the k-nearest-neighbor graph on the documents with $k = 200$ and with cosine similarity defining "nearness."

We ran 2 000 steps of the one-step and two-step optimal algorithm, as well as uncertainty sampling given this problem setup. Each algorithm was initialized with the label of a single randomly selected page in the Programming Language category; all other pages were left unlabeled at the start of the experiment.

Among the 2 000 nodes selected for querying by the one-step-lookahead algorithm, 975 were in the Programming Languages category. The two-step-lookahead algorithm found 990, and uncertainty sampling discovered 734. In contrast, random search would be expected to find only 63 targets with the same number of evaluations. The two-step-lookahead algorithm was able to locate 18.2% of the targets after querying only 1.17% of the dataset. These results again confirm the effectiveness of Algorithm 1.

6.4 Discussion

In light of these results, we make an observation on the nature of the three algorithms tested. First, both the relatively good quality of uncertainty sampling, as well as its relative inferiority to the algorithm described above, can be explained by its slightly different goal. Querying the point with the maximum uncertainty causes the classification boundary to be explored. Although this finds many targets (because they lie on the boundary just as often as the others), the different utility function driving the standard algorithm causes it to also query nodes less likely to be targets more often.

In a sense then, uncertainty sampling is a purely explorative approach. The one-step optimal policy, on the other hand, is a greedy and purely exploitative approach, and it is not surprising that it performs better than uncertainty sampling. We believe the increased performance of the two-step-lookahead policy is that by looking two (or more) steps into the future, the algorithm can consider the relative tradeoffs of exploration versus exploitation, which are two goals that must typically be simultaneously be satisfied in optimization procedures.

7. CONCLUSION AND FUTURE WORK

We have introduced the problem we call "generalized Battleship," that creates the challenge of actively seeking out members of an identified class. To approach this problem, we defined a natural utility function and derived the Bayesian optimal policy for the associated decision problem.

In the opinion of the authors, a common context for such problems will likely involve graph learning. With this in mind, we discussed potential useful models in the context of the generalized Battleship problem using common techniques based on random walks. In particular, we advocate the use of the GRANCH algorithm in this context [12, 13], because it is very efficient, avoids some identified problems in other related quantities, and works without modification on directed graphs. The authors believe this last property is especially important for real-world problems.

The policy presented was tested empirically on two real-world graph datasets and performed well in comparison to a common active learning technique for binary classification, and we believe this difference is due to the different goals of the two methods.

In the future, we hope to investigate different approaches to the underlying classification mechanism for graphs. It would also be interesting to investigate other approaches for increasing the possible number of future evaluations that can be considered by our method; for example, perhaps a branch-and-bound heuristic can be employed with success.

References

[1] M. Brand. A random walks perspective on maximizing satisfaction and profit. In *Proceedings of the 2005 SIAM International Conference on Data mining* (SDM 2005), 2005.

[2] F. Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9:1–15, 2005.

[3] Ş. Ertekin, J. Huang, L. Bottou, and C. L. Giles. Learning on the Border: Active Learning in Imbalanced Data Classification. In *Proceedings of the 16th ACM Conference Information and Knowledge Management* (CIKM 2007), New York, NY, USA, 2007. ACM Press.

[4] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, 2007.

[5] M. Grčar, B. Fortuna, D. Mladenič, and M. Grobelnik. kNN Versus SVM in the Collaborative Filtering Framework. In V. Batagelj, H.-H. Bock, A. Ferligoj, and A. Žiberna, editors, *Data Science and Classification*, Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin, Germany, 2006.

[6] J. He and J. Carbonell. Rare Class Discovery Based on Active Learning. In *Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics* (ISAIM 2008), 2008.

[7] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[8] D. D. Lewis and W. A. Gale. A Sequential Algorithm for Training Text Classifiers. In W. B. Croft and C. J. van Rijsbergen, editors, *Proceedings of the Seventeenth Annual International ACM–SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1994), pages 3–12, London, UK, 1994. Springer–Verlag.

[9] A. K. McCallum. MALLET: A Machine Learning for Language Toolkit, 2002.

[10] W. J. M. Meuffels and D. den Hertog. Solving the *Battleship* Puzzle as an Integer Programming Problem. *INFORMS Transactions on Education*, 10(3):156–162, 2010.

[11] M. A. Osborne, R. Garnett, and S. J. Roberts. Gaussian Processes for Global Optimization. In *3rd International Conference on Learning and Intelligent Optimization* (LION3), 2009.

[12] P. Sarkar and A. W. Moore. A Tractable Approach to Finding Closest Truncated-Commute-Time Neighbors in Large Graphs. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence* (UAI 2007), 2007.

[13] P. Sarkar, A. W. Moore, and A. Prakash. Fast Incremental Proximity Search in Large Graphs. In A. McCallum and S. Roweis, editors, *Proceedings of the 25th International Conference on Machine Learning* (ICML 2008), pages 896–903, Madison, WI, USA, 2008. Omnipress.

[14] A. I. Schein and L. H. Ungar. Active Learning for Logistic Regression: An Evaluation. *Machine Learning*, 68(3):235–265, 2007.

[15] B. Settles. Active Learning Literature Survey. Technical Report 1648, Department of Computer Sciences, University of Wisconsin–Madison, 2010.

[16] M. Sevenster. Battleships as Decision Problem. *ICGA Journal*, 27(3):142–149, 2004.

[17] U. von Luxburg, A. Radl, and M. Hein. Getting lost in space: Large sample analysis of the commute distance. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23* (NIPS 2010), pages 153–160, Red Hook, NY, USA, 2010. Curran Associates, Inc.

[18] L. Yen, F. Fouss, C. Decaestecker, P. Francq, and M. Saerens. Graph nodes clustering with the sigmoid commute-time kernel: A comparitive study. *Data & Knowledge Engineering*, 68:338–361, 2008.

[19] D. Zhou, J. Huang, and B. Schölkopf. Learning from Labeled and Unlabeled Data on a Directed Graph. In *Proceedings of the 22nd International Conference on Machine Learning* (ICML 2005), pages 1041–1048, New York, NY, USA, 2005. ACM Press.

[20] X. Zhu. Semi-Supervised Learning Literature Survey. Technical Report 1530, Department of Computer Sciences, University of Wisconsin–Madison, 2008.

[21] X. Zhu, J. Kandola, J. Lafferty, and Z. Ghahramani. Graph Kernels by Spectral Transforms. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*, pages 277–292. The MIT Press, Cambridge, MA, USA, 2006.

[22] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining Active Learning and Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *Proceedings of the ICML 2003 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 58–65, 2003.