On Node Classification in Dynamic Content-based Networks

Charu C. Aggarwal* Nan Li†

Abstract

In recent years, a large amount of information has become available online in the form of web documents, social networks, blogs, or other kinds of social entities. Such networks are large, heterogeneous, and often contain a huge number of links. This linkage structure encodes rich structural information about the underlying topical behavior of the network. Such networks are often *dynamic* and evolve rapidly over time. Much of the work in the literature has focussed either on the problem of classification with purely text behavior, or on the problem of classification with purely the linkage behavior of the underlying graph. Furthermore, the work in the literature is mostly designed for the problem of static networks. However, a given network may be quite diverse, and the use of either content or structure could be more or less effective in different parts of the network. In this paper, we examine the problem of node classification in *dynamic* information networks with both text content and links. Our techniques use a random walk approach in conjunction with the content of the network in order to facilitate an effective classification process. This results in an effective approach which is more robust to variations in content and linkage structure. Our approach is dynamic, and can be applied to networks which are updated incrementally. Our results suggest that an approach which is based on a combination of content and links is extremely robust and effective. We present experimental results illustrating the effectiveness and efficiency of our approach.

Keywords: Graph Classification, Structural Classification, Label Propagation

1 Introduction

In recent years, there has been an explosion of text content on the web in a variety of forms. In addition to the standard forms of content such as web pages, an enormous amount of content may be found in the form of blogs, wikis, and other forms of social media. Such networks are often a challenge for mining algorithms because they often contain both structure and content. Some examples of such networks include author citation networks, co-authorship networks, product databases with large amounts of text content, and so on. Such networks are *highly dynamic* and may be frequently updated over time. For example, new nodes may constantly be created in the network, as new postings are created in a blog network; similarly old nodes may be deleted, as old postings are deleted. As a result the structure of the network may be quite dynamic, and may vary over time. In the most general case, we model our problem as a graph of nodes, each of which may contain text content.

A key problem which often arises in these domains is that of node classification [1, 2]. The classification problem arises in the context of many network scenarios in which the underlying nodes are associated with content. In the node classification problem, it is assumed that a subset of the nodes in the network may be labeled. It is desirable to use these labeled nodes in conjunction with the structure and content for the classification of nodes which are not currently labeled. For example, many blogs or other network documents may naturally belong to specific topics on the basis of their content and linkage patterns. However, most such documents may not be formally associated with labels in social networking scenarios because of a lack of resources available for a human-centered labeling process. In this paper, we will address the classification problem, in which it is desirable to determine the categories of the unlabeled nodes in an automated way with the use of both structure and content of the network. The presence of labels on a subset of the nodes provides the implicit training data which can be leveraged for learning purposes.

The node classification problem is particularly challenging in the context of very large, dynamic, and evolving social and information networks. In particular, a number of natural desiderata are applicable in the design of classification algorithms in this scenario. These desiderata are as follows:

- Social and information networks are very large, as a result of which link classification algorithms need to be *efficient*. This can be particularly challenging, if we intend to use both text and links during

*IBM Thomas J. Watson Research Center, Email: charu@us.ibm.com.

†University of California at Santa Barbara, Email: nanli@cs.ucsb.edu.

the classification process. The addition of text content to the linkage information is responsible for a considerable increase in the size of the underlying network representation.

- Many such networks are dynamic, and are frequently updated over time. Since the structure of the network may constantly change over time, the underlying classification model may also change. Therefore, the model needs to be efficiently updatable in *real time* in order to account for such changes. Such a dynamic model also needs to be easy to use, so that the end-process of classification can be achieved without too much overhead.
- Such networks are often noisy, as many of the links and content features may not be relevant to the classification process. In addition, different portions of the network may be better suited to different kinds of classification models. For example, some portions of the network may be better classified with structure, whereas other portions may be better classified with content. We need to design a classifier, which can make such decisions in a seamless way, so that the appropriate parts of the network may be used most effectively for the classification process.

The problem of classification is widely studied in the data mining community [7]. The problem has been studied in the context of both structural [3, 4, 10] and content-based [9, 12, 13, 16] analysis. Two natural choices can be used for classification of content-rich networks:

- The most straightforward approach is to directly use text classifiers in order to perform the classification. A variety of text classifiers are available for this purpose. A detailed evaluation of techniques for text categorization may be found in [13, 16]. However, such an approach ignores the rich structural information which is often available in the context of a network.
- A second way to perform the classification is by using the information which is latent in the underlying link structure. For example, effective methods have been proposed in [10] in order to perform link-based classification. Similar techniques have been used in [3] in order to label blogs for classification. However, these methods fail to leverage the information which is available in the underlying content for classification purposes.

It is clear that both text and links encode important information about the underlying network. Furthermore, these provide different views of the underlying information. For example, the text provides ideas about content, whereas the linkage behavior provides information about interconnectedness between different kinds of nodes, some of which may be used for classification. The latter is especially the case, when the content in a given node is limited and the linkage information provides an idea of the relationships of the test node with other labeled nodes. On the other hand, the content can be used to glean better classification insights when the linkage structure is either sparse or not informative enough to provide information about the classification behavior of the underlying node. Therefore, it makes sense to examine whether it is possible to *combine* text and linkage behavior in order to perform robust classification, which works more effectively in a generic scenario. Furthermore, such an integration must be seamless, in that it should be able to automatically use the most effective strategy in a given scenario. This paper will propose a random walk approach, which combines text and linkage behavior, and show that it can be used in a seamless way in order to perform more robust classification. We will refer to this algorithm as *DYCOS*, which corresponds to the fact that it is a DYnamic Classification algorithm with cOntent and Structure. Furthermore, the approach is *dynamic*, and *scalable to large networks*, as it can be applied to large, and rapidly updatable networks.

This paper is organized as follows. We will discuss related work in the remainder of this section. In Section 2, we will introduce a dynamic random-walk model for classification with text and links. Section 3 shows how the *DYCOS* algorithm leverages this model for the classification process. The experimental results are presented in Section 4. Section 5 presents the conclusions and summary.

1.1 Related Work The problem of text classification [9, 12, 13, 16]. has been studied widely in the information retrieval literature. Detailed surveys may be found in [13, 16]. In the context of the web and social networks, text classification poses a significant challenge, because the text is often drawn from heterogeneous and noisy sources which are often hard to model with a standardized lexicon. Some of the earliest work on the use of linkage techniques to enhance classification may be found in [5]. This work uses the *text content* in adjacent web pages in order to model the classification behavior of a web page. However, it is not focussed on the problem of node-classification in a partially labeled graph of documents.

The problem of node classification has also been studied in the graph mining literature, and especially relational data in the context of *label or belief propagation* [14, 18, 19]. Such propagation techniques are also

used as a tool for semi-supervised learning with both labeled and unlabeled examples [21]. A technique has been proposed in [10], which uses link-based similarity for node-classification. Recently, this technique has also been used in the context of blogs [3]. However, all of these techniques use link-based methods only. Some recent work has been done on the clustering problem with content and links [20]. Another work [4] discusses the problem of label acquisition in the context of collective classification. Label acquisition is an important problem, because it is required in order to provide the base data necessary for classification purposes. A method to perform *collective classification* of email speech acts has been proposed in [6]. It has been shown that the analysis of relational aspects of emails (such as emails in a particular thread) significantly improves the classification accuracy. It has also been shown in [5, 17] that the use of graph structures during categorization improves the classification accuracy of web pages. While these methods provide a limited application of structural information, they are not designed to work for massive and dynamic networks which may constantly evolve over time. This paper provides a first approach to the problem of *efficient and dynamic* node classification in a *massive labeled* network, where both text and node labels are available for classification purposes. Much of the work proposed recently is not applicable to the case of *massive information networks in a dynamic scenario* because of scalability issues. We will use carefully designed summary structures which can efficiently perform such a classification. We will show that the use of both sources in the classification process provides an effective classification technique in such massive networks. Furthermore, we design our technique to work effective for *massive and dynamic networks* which may constantly evolve over time. We will show that our method will show considerable improvements over other existing methods.

2 Node Classification Model with Text and Links

We will first introduce some notations and definitions which are relevant to the node classification problem. We assume that we have a large network containing a set of nodes $\mathcal{N}_t$ at time t. Since, the approach is dynamic, we use a time-subscripted notation $\mathcal{N}_t$ in order to denote the changing nodes in the network. A node in $\mathcal{N}_t$ may correspond to a blog post, a social network profile page, or a web page. We also assume that a subset $\mathcal{T}_t$ of these nodes $\mathcal{N}_t$ may be labeled. These nodes form the training nodes, and they contribute both linkage and text information for classification purposes. We assume that the nodes in $\mathcal{T}_t$ are labeled from a total of k classes, which are drawn from the set $\{1 \ldots k\}$. As in the case of the node set $\mathcal{N}_t$, the set $\mathcal{T}_t$ is not static, but may dynamically change over time, as new labeled nodes may be added to the network. For example, either a new labeled node may be added to both $\mathcal{N}_t$ and $\mathcal{T}_t$, or an existing node in $\mathcal{N}_t$ may be initially unlabeled (and therefore not a part of the training data), but may eventually be labeled, when new training information is received. In the latter case, we add that node to $\mathcal{T}_t$. Similarly, the set of edges at time t is denoted by $\mathcal{A}_t$. Furthermore, new labels may be acquired for different nodes over time, as a result of which the set $\mathcal{T}_t$ may change as well. Clearly, this dynamic setting is extremely challenging, because it implies that the training model may change rapidly. The entire network is denoted by $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{A}_t, \mathcal{T}_t)$ at a given time t.

In order to achieve our goal, the *DYCOS* approach will construct a summary representation which is based on both text and link structure. In order to perform the classification, we will create a text-augmented representation of the network, which is leveraged for classification purposes. We will show how to implement this summary representation efficiently, so that it is possible to use it effectively in a network. Our broad approach is to construct an intuitive random walk based approach on the network, in which both text and links are used during the walk process for classification. The level of importance of text and links can either be controlled by a user, or it can be inferred in an automated way, as discussed below. Since we intend to design classification techniques which use the underlying content, it is useful to first determine the words which are most discriminative for classification purposes. The ability to select out a compact classification vocabulary is also useful in reducing the complexity and size of the model at a later stage. The discriminative quantification of a given word from the corpus is performed with the use of a well known measure known as the *gini-index*. We dynamically maintain a *sample reservoir* S_t of *labeled documents* in the collection, and use them for the purposes of computing the gini-index. For this purpose, we can use the reservoir sampling algorithm discussed in [15]. From time to time, we compute the gini-indices in order to compute the discriminative power of the different words. The frequency of updating the gini-indices can be either equivalent to or less than the frequency the network is dynamically updated. For a given word w, let $p_1(w) \ldots p_k(w)$, be the relative *fractional presence* of the word w in the k different classes. In other words, if $n_1(w) \ldots n_k(w)$ be the number of pages in the sample S which contain the word w, then we estimate $p_i(w)$ as

follows:

$$p_i(w) = n_i(w) / \sum_{j=1}^{k} n_j(w) \tag{2.1}$$

Then, the gini-index $G(w)$ for the word w is computed as follows:

$$G(w) = \sum_{j=1}^{k} p_j(w)^2 \tag{2.2}$$

The value of $G(w)$ always lies in the range $(0, 1)$. If the word is evenly distributed across the different classes, then the value of $G(w)$ is closer to 0. On the other hand, if the word w has a preponderance in one of the classes, then the value of $G(w)$ is closer to 1. Thus, words which have a higher value of $G(w)$ are more discriminative for classification purposes. As a first step, we pick a set $\mathcal{M}_t$ of the top m words which have the highest value of $G(w)$ and use them in order to construct our structural node classification model. The set $\mathcal{M}_t$ represents the *active vocabulary* which is useful for classification purposes. In our current implementation (Section 4), $\mathcal{M}_t$ is updated at the same pace as the dynamic network is updated. Nonetheless, we note that $\mathcal{M}_t$ does not need to be updated at each time instant t. Rather, it can be updated in batch at specific instants in time, with a much less frequency compared to that the network is updated. The discriminatory indices of the words are analyzed periodically, and the most discriminatory words are used for classification purposes. These discriminative words are used in order to create a new semi-bipartite representation of the network which is useful for classification purposes.

2.1 The Semi-Bipartite Content-Structure Transformation One of the goals of the *DYCOS* algorithm is to create a model which can deal with the content and links in a *seamless* way for the transformation process. For this purpose, both the content and the original links are transformed into a structural representation, which is referred to as the *semi-bipartite* content-link transformation. The set $\mathcal{M}_t$ provides a more compact vocabulary which is used in order to create a *semi-bipartite* content-link transformation. The semi-bipartite representation is a graph in which one partition of nodes is allowed to have edges either within the set, or to nodes in the other partition. The other partition is only allowed to have edges to the first, but it does not have any edges within the set. Therefore, it is referred to as *semi-bipartite*, as only one of the two node sets satisfies the bipartite property. The semi-bipartite content-link

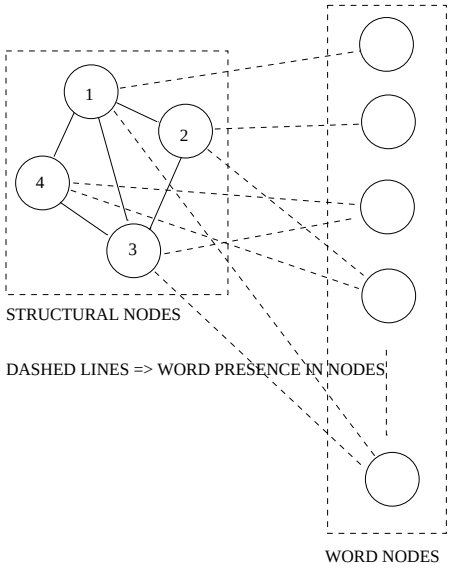


Figure 1: The Semi-bipartite Transformation

transformation defines defines two kinds of nodes: (i) The first kind are the *structural nodes* which are the same as the original node set $\mathcal{N}_t$. This set inherits edges from the original network. (ii) The second kind of nodes are the *word nodes* which are the same as the discriminative vocabulary $\mathcal{M}_t$.

Then, we construct the semi-bipartite graph $F_t = (\mathcal{N}_t \cup \mathcal{M}_t, A_t \cup A'_t)$, in which $\mathcal{N}_t$ and $\mathcal{M}_t$ form the two sides of the bipartite partition. The set A_t is inherited from the original network, whereas A'_t is constructed on the basis of word-presence in the text in different network nodes. Specifically, an undirected edge exists between the information network node $i \in \mathcal{N}_t$, and the word node $j \in \mathcal{M}_t$, if the corresponding word is contained in the information node i. Thus, the edges in A_t are within a partition, whereas the edges in A'_t are across the partition. An example of this transformation is illustrated in Figure 1. The node set which corresponds to the structural nodes has edges which are indicated by solid lines, whereas the connections between structure and content nodes are illustrated by dashed lines. Thus, a walk from one node to another may use either solid or dashed lines. This provides a way to measure proximity both in terms of link and content. The ability to utilize such proximity in the context of a classification process helps us combine links and content in a seamless way for classification in terms of the structural proximity in the new transformed network.

In addition, a number of data structures are required in order to allow efficient traversal of the text and linkage structure in our random-walk approach. These

data structures are as follows: (1) For each of the word nodes $w \in \mathcal{M}_t$, we maintain an inverted list containing the set of node identifiers which contain the word corresponding to w. We assume that the set of nodes pointed to by word i is denoted by P_i. (2) For each of the original set of nodes $\mathcal{N}_t$ in the network structure, we maintain an inverted list of words contained in the corresponding document. The set of words pointed to by node i is denoted by Q_i. (3) For each node identifier, we maintain information about its class label, if the node is labeled. Otherwise, we simply maintain the meta-information that the node is not labeled.

We note that total size of all the inverted lists P_i for different values of i is at most equal to the text size of the collection, if it is represented in terms of only the discriminative words. Similarly, the total size of all the inverted lists Q_i for different values of i is at most equal to the discriminative text collection size. These inverted lists can be updated easily during addition or deletion of nodes to the collection. During addition or deletion of nodes, we need to either add to or delete from inverted lists P_i, such that word i is contained in the added or deleted node. We also need to add (delete) an inverted list Q_r corresponding to the newly added (removed) node r. We note that this incremental update process is extremely efficient, and can be dynamically performed for a data stream. From time to time, we may also want to adjust the word nodes, depending upon the change in discriminatory behavior. In such cases, we need to add or delete corresponding word nodes. The process of updating the inverted lists is similar to the previous case. The update process can also be efficiently applied to a node, when the content within a node changes. In this cases, the corresponding links between the structure and content nodes need to updated.

3 Classification with Text and Link-based Random Walks

In this section, we will describe the classification approach of the *DYCOS* algorithm. The use of both content and links during the random walk process is critical in creating a system which provides effective classification. Since random walks can be used to define proximity in a variety of ways [8], a natural approach is to construct proximity-based classifiers which use the majority labels of random walk nodes for the propagation process. Since the text is included within the node structure of the semi-bipartite graph, it follows that a random walk on this graph would implicitly use both text and structural links during the classification process. The *starting node* in this random walk is the unlabeled node in $\mathcal{N}_t$ which needs to be classified. Of course, we would also like to have a way to control the relative impact of text and structural nodes during the classification process. We note that a straightforward use of a random walk over the semi-bipartite graph F_t may not be very effective, because the walk can get lost by the use of individual word nodes in the random walk. In order to be able to control this relative importance, we will define the walk *only over the structural nodes with implicit hops over word nodes*. Specifically, a step in the random walk can be one of two types:
(1) The step can be a *structural hop* from one node in $\mathcal{N}_t$ to another node in $\mathcal{N}_t$. This is a straightforward step from one node to the next with the use of a link in the original graph. If such a link does not exist, then the structural hop teleports to the starting node.
(2) The step can be a *content-based multi-hop* from a node in $\mathcal{N}_t$ to another node in $\mathcal{N}_t$. This step uses the linkage structure between the structural and word nodes during the hop. Thus, each hop really uses an *aggregate analysis of* the *word-based* linkages between one structural node in $\mathcal{N}_t$ and another structural node in $\mathcal{N}_t$. The reason for this aggregate analytical multi-hop approach is to reduce the noise which naturally arises as a result of the use of straightforward walks over *individual word nodes* in order to move from one structural node to the other. This is because many of the words in a given document may not be directly related to the relevant class. Thus, a walk from one structural node to the other with the use of a single word node could diffuse the random walk to less relevant topics.

We will discuss more details about how this content-based multi-hop is computed slightly later. We use a statistical analysis of the nodes encountered during the random walk in order to perform the classification. A key aspect here is to be able to control the importance of structure and content during the hops. For this purpose, we use a *structure parameter* p_s. This parameter defines the probability that a particular hop is a structural hop rather than a content hop. When the values of p_s is set at 1, then it means that content is completely ignored during the classification process. On the other hand, when the value of p_s is set at 0, then it means that only content is used for classification. We will discuss more details about the classification process below.

3.1 Classification Process The process of classification uses repeated random walks of length h starting at the source node. The random walk proceeds as follows. In each iteration, we assume that the probability of a structural hop is p_s. Otherwise, a content multi-hop is performed with probability $(1 - p_s)$. By varying the value of p_s, it is possible to control the relative importance of link and content in the classification process. While defining the length of a walk, a content-hop is

defined as a single hop in the same way as a structural hop, even though a content walk is really performed using analysis of intermediate word nodes. A total of l such random walks are performed. Thus, a total of $l \cdot h$ nodes are visited in the random walk process. These nodes may either belong to a particular class, or they may not be labeled at all. The most frequently encountered class among these $l \cdot h$ nodes is reported as the class label. If no labeled node is encountered through all random walks (which is a very rare situation), *DYCOS* simply reports the most frequent label of all nodes currently in the network. This is specific to the current time stamp and does not depend on the particular source node. A high-level pseudo-code sketch of the classification algorithm is presented in Algorithm 1.

Data: Network $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{A}_t, \mathcal{T}_t)$, number of random walks, l, walk length, h, structural hop probability, p_s
Result: Classification of $\mathcal{T}_t$, accuracy, θ

```
for Each node v in T_t do
    for i from 1 to l do
        Perform an h-hop random walk from v, with structural hop probability, p_s;
    Classify v with the class label most frequently encountered;
θ ← the percentage of nodes correctly classified;
Return classification labels and θ;
```

Algorithm 1: *DYCOS* Classification Process

Next, we will discuss the efficient implementation of structural and content hops. This is done with the use of the inverted indices which are available at the different nodes of the graphs. At each node in the random walk process, we flip a coin with probability p_s. In the event of a success, we perform a structural hop; otherwise we perform a content hop. Structural hops are straightforward, because we only need to look up the adjacency list for that node, and perform the corresponding hop.

For the case of the content-based hops, a two-step approach is required. First, we need to determine the nodes with the top-q most frequent 2-hop paths from a node in $\mathcal{N}_t$ to another node in $\mathcal{N}_t$ with the use of an intermediate word node. The first step is to determine all the nodes which are reachable in paths of length 2. Let the relative frequency of the number of 2-hop paths which lead to these q nodes be denoted by $r_1 \ldots r_q$. Then, we sample the ith among these nodes with probability r_i. By truncating the random walk process to only the top-q nodes, we ensure that the random walk is not lost because of non-topical words in the documents. In order to actually perform the walk, we need to use the inverted lists at the nodes in $\mathcal{N}_t$ and $\mathcal{M}_t$. For each node in $\mathcal{N}_t$, we can determine the word nodes contained in it. Then, for each word node, we can determine the structural nodes which contain that word. This can again be achieved by using the inverted lists at the word nodes. The union of these lists is the set of nodes which can be reached in a content-walk of length 2. The top-q most frequent nodes among these are sampled for the purposes of determining the next node in the walk. We note that content hops are less efficient to perform than structural hops, but the use of inverted lists greatly speeds up the process.

3.2 Analysis An important point to note is that we are essentially using Monte-carlo sampling of the paths from different nodes. The use of such a sample can result in some loss of accuracy, but the advantage is that it is much more efficient than the use of an exact computation of node probabilities. This is because the exact computation of node probabilities can require expensive matrix operations based on the structure of the graph adjacency matrix. This is not very helpful for a large network in which many such computations need to be performed. The sampling approach is also critical in being able to utilize the approach effectively in a dynamic scenario in which repeated re-computation of node probabilities is required. Therefore, an efficient sampling approach such as the one discussed in the paper is critical. In this section, we will study the loss of accuracy which arises from the use of such samples. The aim is to show that the use of Monte-Carlo samples retains practically the same effectiveness as an approach which can determine the hop probabilities exactly. As mentioned earlier, the class which is visited the maximum number of times during the entire random walk process is reported as the relevant class. As in the previous discussion, we ignore word nodes in the analysis of hops, since they are only used as intermediate nodes during the content hops. Therefore, all hops are considered to be either structural hops from one node in $\mathcal{N}_t$ to another node in $\mathcal{N}_t$, or content hops from one node in $\mathcal{N}_t$ to another node in $\mathcal{N}_t$ with the use of an intermediate word node. The main focus is to show that the *ordering* of different classes in terms of the number of visits does not change significantly because of the sampling process. For this purpose, we will use the Hoeffding inequality. First, we will consider the case of two classes. Then, we will generalize our results to an arbitrary number of classes. Let us consider two classes 1 and 2, for which the *expected fraction of visits* for a particular test node are f_1 and f_2 respectively, so that $b = (f_1 - f_2) > 0$. In this case, class 1 is a

more appropriate label for the test node as compared to class 2, because it is the majority class. We further note that the sum of f_1 and f_2 may not necessarily be 1, because many of the intermediate nodes in the hop may be unlabeled. We would like to determine the probability that the Monte-Carlo sampling process results in the *undesirable outcome* of the ordering of the classes 1 and 2 being reversed during the random sampling process. This directly provides us with the classification error probability.

LEMMA 3.1. *Let us consider two classes with expected visit probabilities of f_1 and f_2 respectively, such that $f_1 - f_2 > 0$. Then, the probability that the class which is visited the most during the sampled random hop process is reversed to class 2, is given by at most $e^{-l \cdot b^2/2}$.*

Proof. Let X_i be the random variable which represents the fraction of nodes of class 1, which are visited during the ith random walk, and let Y_i be the random variable which represents the fraction of nodes of class 2, which are visited during the ith random walk. Then we define the random variable defining the *differential hop fraction* as $Z_i = X_i - Y_i$. It is clear that Z_i is a random variable which lies in the range $[-1, 1]$, and has an expected value of b. Then, we define the random variable S as the sum of the different values of Z_i over the l different random walks. Therefore, we have:

$$S = \sum_{i=1}^{l} Z_i \tag{3.3}$$

Since $E[Z_i] = b$, it follows that $E[S] = l \cdot b$. In order for the majority class to be class 2, we need $S < 0$. Therefore, we would like to determine the probability $P(S < 0)$. For this purpose, we will make use of the Hoeffding inequality, because S is expressed as a sum of *bounded* random variables in the range $[-1, 1]$. By using $E[S] = l \cdot b$, we get:

$$P(S < 0) = P(S - E[S] < -l \cdot b)$$

Since S is the sum of l independent random variables, which lie in the range $[-1, 1]$, we can use the Hoeffding inequality to bound the probability of error, a proxy for which is the expression $P(< 0)$:

$$P(S < 0) = e^{-l \cdot b^2/2}$$

The result above shows that the probability of error because of sampling reduces exponentially with the number of paths that are sampled. For example, consider the case, when we sample 100 different paths, and $b = 0.1$. In that case, the probability of error is given by at most $e^{-1000 \cdot 0.01/2} = e^{-5} < 1\%$. This suggests that the *additional error of approach because of the sampling process* will be less than 1%. In general, the exponential rate of error decrease with sample size is critical in ensuring that the approach can be used efficiently with Monte-carlo sampling. Next, we will generalize the result to the case of k classes. First, we need to define the concept of *b-accuracy* of a sampling process in a case with k classes.

DEFINITION 1. *Let us consider the node classification problem with a total of k classes. We define the sampling process to be b-accurate, if none of the classes whose expected visit probability is less than b of the class with the largest expected visit probability turns out have the largest sampled visit probability.*

We note that the above definition is simply a generalization of the case for two classes. The main purpose of defining the concept of b-accurate is to ensure that none of the classes which are too far off from the optimum value are picked as a result of the sampling process. We can directly generalize the results of Lemma 3.1 in order to prove that the sampling process is b-accurate.

THEOREM 3.1. *The probability that the sampling process results in a b-accurate reported majority class is given by at least $1 - (k-1) \cdot e^{-l \cdot b^2/2}$.*

Proof. The results of Lemma show that the probability of pairwise error is at most $e^{-l \cdot b^2/2}$ because of the sampling process. Therefore, the probability of pairwise error for any of the (at most) $(k-1)$ other classes which have expected visit probability at most b of the optimum is given by at most $(k-1) \cdot e^{-l \cdot b^2/2}$. The result follows.

Theorem 3.1 presents the theoretical analysis of the accuracy of the proposed random walk-based approach. It states that even with random walks, the proposed classification method is able to report the correct majority class for a node with at least a certain probability. This suggests that the proposed *DYCOS* framework is a good solution to overcome the potential noise encountered during random walks.

4 Experimental Results

In this section, we validate the effectiveness and efficiency of *DYCOS* with experiments on real data sets. The effectiveness is measured by classification accuracy, which is the proportion of correctly classified nodes as to the total number of test nodes which are classified. The efficiency is measured by the execution time of classification. For this purpose, we report the wall-clock time. In order to establish a comparative study, we compare the

performance of *DYCOS* to that of NetKit-SRL toolkit[1], which is an open-source network learning toolkit for statistical relational learning [11]. The results obtained in a multi-class classification environment demonstrate that *DYCOS* is able to improve the average accuracy over NetKit-SRL by 7.18% to 17.44%, while reducing the average runtime to only 14.60% to 18.95% of that of NetKit-SRL. The specific setup of NetKit-SRL chosen in our experiments will be described later. It is worth mentioning that NetKit-SRL package is a generic toolkit without particular optimization for our problem definition.

In order to illustrate the efficiency of *DYCOS* in a dynamic environment, the model update time in the presence of incrementally arriving data is also reported. This confirms that the underlying classification model can be dynamically maintained in an efficient way. This can be very important in the context of large scale applications in which the change occurs continuously over time.

4.1 Experiment Setup Two real data sets are used in our experimental evaluation, which are CORA data and DBLP data, as shown in Table 1.

Table 1: Data Set Description

Name	Nodes	Edges	Classes	Labeled Nodes
CORA	19,396	75,021	5	14,814
DBLP	806,635	4,414,135	5	18,999

In Table 1, the labeled node number is the number of nodes whose class labels are known and the class number is the number of distinct classes the nodes belong to.

CORA data. The CORA graph is downloaded from `http://www.cs.umass.edu/~mccallum/code-data.html`. This data set contains a set of research papers and the citation relations among them. There are 19,396 distinct papers and 75,021 citation relations among them. Each node is a paper and each edge is a citation relation. A total of 12,313 English words are extracted from the titles of those papers to associate each paper with keywords. The CORA data set is well-suited for our experiments because the papers are classified into a topic hierarchy tree with 73 leaves. Each leaf represents a specific research area in computer science. We reconfigure the hierarchy to achieve a more coarse-grained classification. We extract 5 classes out of the 74 leaves, and 14,814 of the papers belong to these 5 classes. Since the CORA graph does not include temporal information, we segment the data into 10 sub-graphs, representing 10 synthetic time periods. The classification is performed dynamically when the graph increasingly evolves from containing only the first sub-graph to containing all sub-graphs.

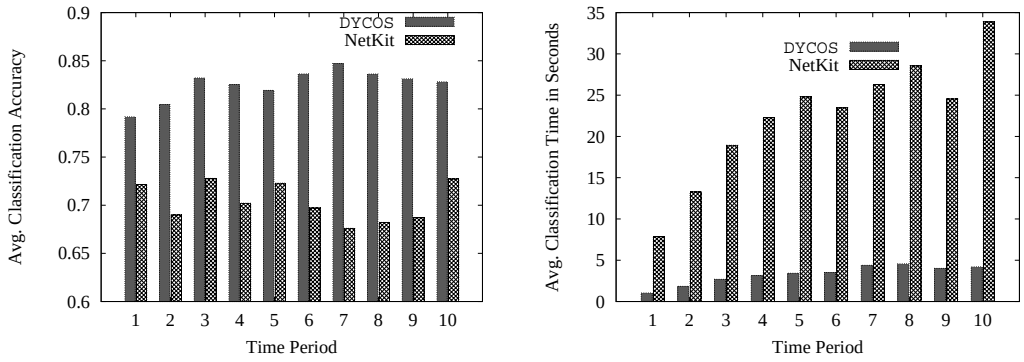
DBLP data. The DBLP graph is downloaded from `http://www.informatik.uni-trier.de/~ley/db/` and updated until March 27, 2010. It contains 806,635 distinct authors and 4,414,135 collaboration edges among them. Each node is an author and each edge represents a co-author relationship. A total of 194 English words in the domain of computer science are manually collected to associate authors with keyword information. The number of occurrences of each word is calculated based on the titles of publications associated with each author. We use 5 class labels, which denote 5 computer science domains: computer architecture, data mining, artificial intelligence, networking and security. We associate some of the authors with ground-truth class labels using information provided by ArnetMiner[2], which offers a set of comprehensive search and mining services for academic community. In total we have collected class labels for 18,999 authors. We segment the whole DBLP graph into 36 annual graphs from year 1975 to year 2010. The classification is performed dynamically as the graph evolves over time.

NetKit-SRL toolkit. The well-known NetKit-SRL toolkit was used for comparative study. NetKit-SRL, or NetKit for short, is a toolkit for learning from and classifying networked data. It is open-source and publicly available. It is aimed at estimating the class membership probability of unlabeled nodes in a partially labeled network [11]. NetKit contains three key modules, local classifier, relational classifier and collective inferencing. For more details on NetKit, we refer the readers to [11]. In our experiments, we use domain-specific class-prior as the local classifier, network-only multinomial Bayes classifier as the relational classifier and relaxation labeling for collective inferencing.

Experiment parameters. Several parameter settings were tested in order to examine *DYCOS*'s performance under various conditions. They are: 1) The number of most discriminative words, denoted by m (see Section 2). 2) The size constraint of the inverted list for word i (see Section 2.1), denoted by a, i.e. $|P_i| <= a$. This is required to control the extent to which each word should be expanded to form 2-hop paths from the current source node. Such a constraint is necessary to increase the efficiency of the random-walk process. 3) Number of top 2-hop paths, q (see Section 3.1). 4) The structure parameter p_s (see Section 3). 5) Number of

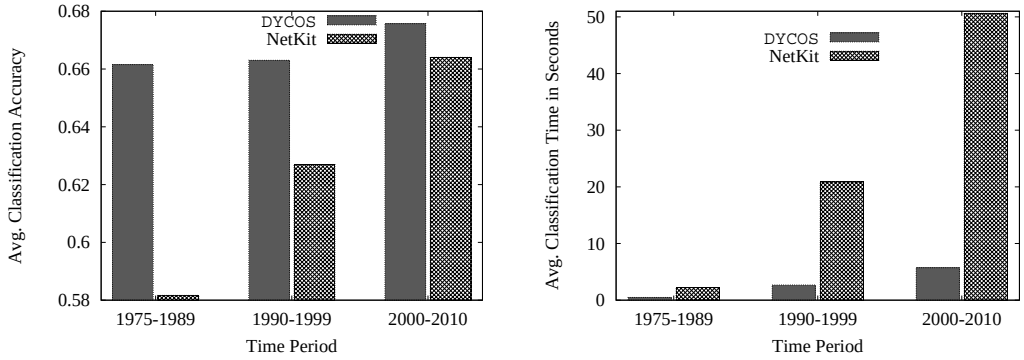
[1] `http://www.research.rutgers.edu/~sofmac/NetKit.html`.

[2] `http://www.arnetminer.org/`.



(a) Avg. Classification Accuracy Comparison (b) Avg. Classification Time Comparison

Figure 2: Comparative Study on CORA



(a) Avg. Classification Accuracy Comparison (b) Avg. Classification Time Comparison

Figure 3: Comparative Study on DBLP

random walks for each source node, l (see Section 3.1). 6) Number of hops in each random walk, h (see Section 3.1). All experiments are run using one core on an Intel Xeon 2.5GHz and 32G RAM server with Fedora 8.

In the following sections, the classification performance of *DYCOS* is first compared to that of NetKit. Then, we examine how well *DYCOS* performs in a dynamic network environment. Finally, we demonstrate the model sensitivity to two important parameters, the number of most discriminative words, m, and the size constraint of inverted lists for words, a.

4.2 Classification Performance The classification performance is measured by both accuracy and efficiency. The accuracy is the fraction of correctly classified nodes. The efficiency is defined by the wall-clock execution time in seconds. The parameter setting for *DYCOS* is: $m = 5$, $a = 30$, $p_s = 0.7$, $l = 3$, and $h = 10$. We will show later *DYCOS* is not significantly sensitive to these parameters. The setting for NetKit is as aforementioned. We utilize the "-test" option in NetKit to ensure a consistent set of testing samples, on which the accuracy is computed, between NetKit and *DYCOS*.

4.2.1 Comparative Study on CORA Data Set In this section, we compare the *DYCOS* and Netkit algorithms on the CORA data set. Figure 2(a) shows the average classification accuracy of both the *DYCOS* and Netkit algorithms for each synthetic time period. Clearly, the *DYCOS* classification model enables a performance gain ranging from 9.75% (time period 1) to 22.60% (time period 7). The average accuracy increment induced by *DYCOS* is 17.44% on the CORA data set.

The comparison of running time for each time period is shown in Figure 2(b). As illustrated, *DYCOS* is much more efficient in terms of running time. The running time of *DYCOS* is only a portion of that of NetKit, and it ranges from 12.45% (time period 10) to 16.70% (time period 7). The average running time of *DYCOS* is 14.60% that of NetKit on the CORA data set.

4.2.2 Comparative Study on DBLP Next, we present the comparative results on DBLP data. In order to establish a dynamic view, we divide the entire 36-year course of time into three periods, 1975-1989, 1990-1999 and 2000-2010. Figure 3(a) presents the average accuracy of both *DYCOS* and Netkit, for each time period. *DYCOS* achieves a performance gain ranging from 1.75% (time period 2000-2010) to 13.73% (time period 1975-1989). The average accuracy increment induced by *DYCOS* is 7.18% on DBLP.

The comparison of classification running time for each time period is shown in Figure 3(b). As shown, *DYCOS* again decreases running time significantly. The running time of *DYCOS* is only a portion of that of NetKit. This time ranges from 11.30% (time period 2000-2010) to 21.30% (time period 1975-1989). The average running time of *DYCOS* is 18.95% that of NetKit on DBLP data.

4.3 Dynamic Update Efficiency In this section, we investigate the efficiency of *DYCOS* in the presence of dynamically arriving data. As aforementioned, *DYCOS* handles temporally-evolving graphs with efficient update mechanisms. Table 2 presents the average model update time (in seconds) of *DYCOS* when new data from the next synthetic time period arrives, on CORA data. The average model update time over all 10 time periods is 0.015 seconds.

Table 2: Avg. Dynamic Updating Time in Each Time Period on CORA

Time Period	**1**	**2**	**3**	**4**	**5**
Update Time (Sec.)	0.019	0.013	0.015	0.013	0.023
Time Period	**6**	**7**	**8**	**9**	**10**
Update Time (Sec.)	0.015	0.014	0.014	0.013	0.011

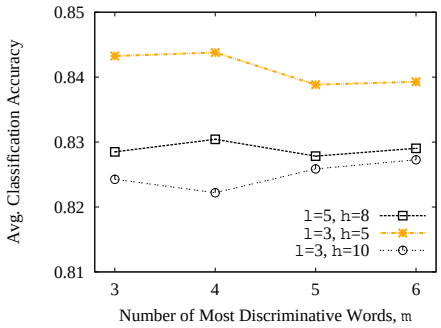


Figure 4: Avg. Classification Accuracy vs. m on CORA

Table 3 presents the average annual model update time (in seconds) of *DYCOS* over various time periods, on DBLP data. The average annual model update time over all 36 years is 0.38 seconds.

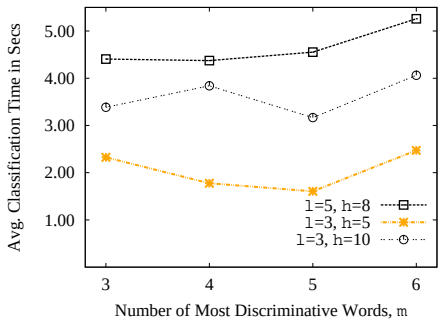


Figure 5: Avg. Classification Time vs. m on CORA

Table 3: Avg. Dynamic Updating Time in Each Time Period on DBLP

Time Period	**1975-1989**	**1990-1999**	**2000-2010**
Update Time (Sec.)	0.03107	0.22671	1.00154

The results demonstrate the efficiency of maintaining the *DYCOS* model under a dynamically-evolving network environment. This is a unique advantage of *DYCOS* in terms of its ability to handle dynamically updated graphs.

4.4 Parameter Sensitivity The purpose of the parameter sensitivity study is to examine the sensitivity of *DYCOS* to various parameters, and the nature of their impact on performance. Due to limited space, we demonstrate the sensitivity study on two particularly important parameters, the number of most discriminative words, m, and the size constraint of inverted lists for words, a.

4.4.1 Parameter Sensitivity on CORA For CORA data set, three different scenarios are created based on the values of the remaining parameters, which are (i) $l = 5$, $h = 8$, (ii) $l = 3$, $h = 5$, and (iii) $l = 3$, $h = 10$, with q set to 10 for all three. Figures 4 and 5 demonstrate the variation in classification performance over different m in accuracy and running time respectively. It is evident that there is no significant correlation between m and the classification performance, though the running time increases slightly with m. This is expected because a higher value of m does not necessarily imply better quality of results, since less discriminative words might be included when m increases.

The accuracy and efficiency variations with the pa-

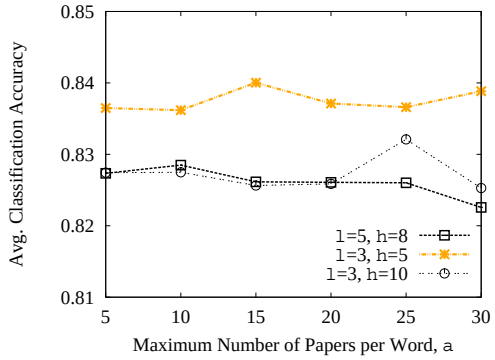


Figure 6: Avg. Classification Accuracy vs. a on CORA

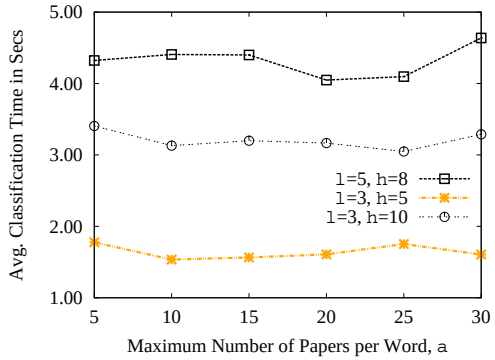


Figure 7: Avg. Classification Time vs. a on CORA

rameter a are presented in Figures 6 and 7. The result is reasonable, considering that while larger inverted lists for words give opportunity for more 2-hop paths to be considered, they might meanwhile include less relevant 2-hop paths.

The robustness of *DYCOS* is therefore illustrated in that it does not heavily rely on parameter setting, and more efficient choices can achieve equally effective results.

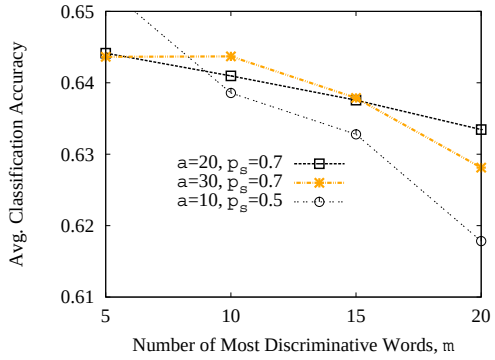


Figure 8: Avg. Classification Accuracy vs. m on DBLP

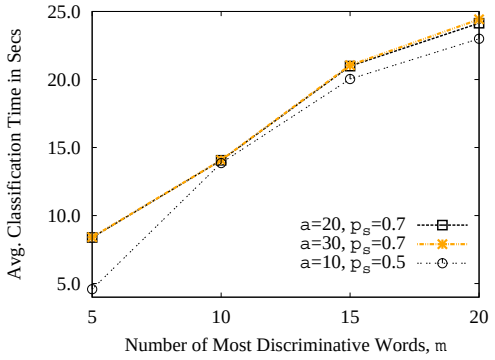


Figure 9: Avg. Classification Time vs. m on DBLP

4.4.2 Parameter Sensitivity on DBLP For DBLP data, Figures 8 and 9 demonstrate the sensitivity to parameter m in terms of both accuracy and running time, under three scenarios: (i) $a = 20$, $p_s = 0.7$, (ii)$a = 30$, $p_s = 0.7$, and (iii)$a = 10$, $p_s = 0.5$, with $q = 10$ for all scenarios. Interestingly, we can observe that, on DBLP data, classification accuracy reduces when m increases. This is expected because a higher value of m implies that less discriminative words are included into computing 2-hop paths during random walks. The sensitivity to a in Figures 10 and 11 shows similar trends in DBLP as in CORA.

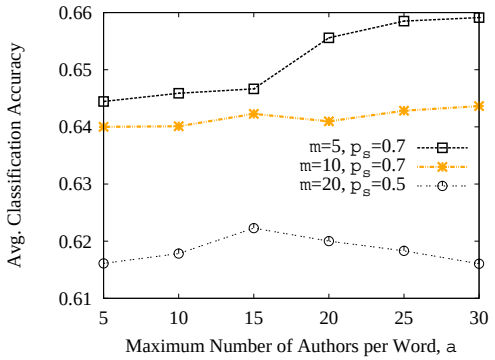


Figure 10: Avg. Classification Accuracy vs. a on DBLP

5 Conclusions and Summary

In this paper, we presented an efficient, dynamic and scalable method for node classification in networks with both structure and content. The classification of content-based networks is challenging, because some parts of the network may be more suited to structural classification, whereas others may be suited to content-based classification. Furthermore, many networks are dynamic, which requires us to maintain an incremental

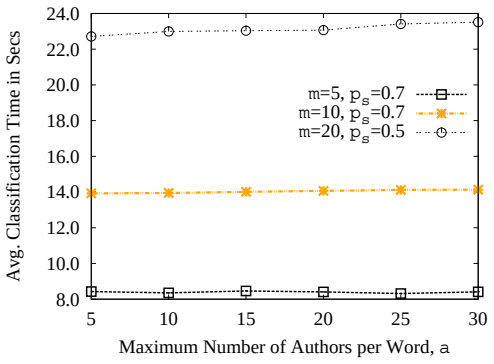


Figure 11: Avg. Classification Time vs. a on DBLP

model over time. Our results show that our algorithms are scalable, and can be be applied to large and dynamic networks. We show the advantages of using a combination of content and linkage structure, which can provide more robust classifications across different parts of a diverse network. We present experimental results on real data sets, and show that our algorithms are much more effective and efficient than competing algorithms in terms of both effectiveness and efficiency.

Acknowledgements

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice hereon.
We would also like to thank Xifeng Yan for his comments and suggestions.

References

[1] C. C. Aggarwal, and H. Wang, *Managing and Mining Graph Data*, Springer, (2010).
[2] C. C. Aggarwal, *Social Network Data Analytics*, Springer, (2011).
[3] S. Bhagat, G. Cormode, and I. Rozenbaum, *Applying link-based classification to label blogs*, WebKDD/SNA-KDD, (2007), pp. 97–117.
[4] M. Bilgic and L. Getoor, *Effective label acquisition for collective classification*, KDD Conference, (2008), pp. 43–51.
[5] S. Chakrabarti, B. Dom, and P. Indyk, *Enhanced hypertext categorization using hyperlinks*, SIGMOD Conference, (1998), pp. 307–318.
[6] V. R. de Carvalho and W. W. Cohen, *On the collective classification of email "speech acts"*, SIGIR Conference, (2005), pp. 345–352.
[7] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, Wiley, (2000).
[8] G. Jeh and J. Widom, *Scaling personalized web search*, WWW Conference, (2003), pp. 271–279.
[9] T. Joachims, *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*, ECML Conference, (1998), pp. 137–142.
[10] Q. Lu and L. Getoor, *Link-based classification*, ICML Conference, (2003), pp. 496–503.
[11] S. A. Macskassy, and F. Provost, *Classification in Networked Data: A Toolkit and a Univariate Case Study*, I Journal of Machine Learning Research, Vol. 8, (2007), pp. 935–983.
[12] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell, *Text classification from labeled and unlabeled documents using EM*, Machine Learning, Vol. 39(2–3), (2000), pp. 103–134.
[13] F. Sebastiani, *Machine learning in automated text categorization*, ACM Computing Surveys, Vol. 34(1), (2002), pp. 1–47.
[14] B. Taskar, P. Abbeel, and D. Koller, *Discriminative probabilistic models for relational data*, UAI, (2002), pp. 485–492.
[15] J. S. Vitter, *Random sampling with a reservoir*, ACM Transactions on Mathematical Software, Vol. 11(1), (1985), pp. 37–57.
[16] Y. Yang, *An evaluation of statistical approaches to text categorization*, Information Retrieval, Vol. 1(1-2), (1999), pp. 69–90.
[17] T. Zhang, A. Popescul, and B. Dom, *Linear prediction models with graph regularization for web-page categorization*, KDD Conference, (2006), pp. 821–826.
[18] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, *Learning with local and global consistency*, Advances in Neural Information Processing Systems, Vol. 16, (2004), pp. 321328.
[19] D. Zhou, J. Huang, and B. Schölkopf, *Learning from labeled and unlabeled data on a directed graph*, ICML Conference, (2005), pp. 1036–1043.
[20] Y. Zhou, H. Cheng, and J. X. Yu, *Graph clustering based on structural/attribute similarities*, PVLDB, Vol. 2(1), (2009), pp. 718–729.
[21] X. Zhu, Z. Ghahramani, and J. D. Lafferty, *Semi-supervised learning using gaussian fields and harmonic functions* ICML Conference, (2003), pp. 912–919.