A Class-Based Search System in Unstructured P2P Networks

Juncheng Huang, Xiuqi Li, and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

***Abstract*— Efficient searching is one of the important design issues in peer-to-peer (P2P) networks. Among various searching techniques, semantic-based searching has drawn significant attention recently. Gnutella-like efficient searching system (GES) [18] is such a system. GES derives a *node vector*, a semantic summary of all of the documents on a node, based on vector space model (VSM). The topology adaptation algorithm and search protocol are then designed according to the similarity between node vectors of different nodes. However, although GES is suitable when the distribution of documents in each node is uniform, it may not be efficient when the distribution is diverse. When there are many categories of documents at each node, the node vector representation may be inaccurate. We extend the idea of GES and present a class-based semantic searching system (CSS). It makes use of a data clustering algorithm, online spherical k-means clustering (OSKM) [16], to cluster all documents on a node into several classes. Each class can be viewed as a virtual node. Virtual nodes are connected through virtual links. As a result, class vector replaces node vector and plays an important role in the class-based topology adaptation and search process, which makes CSS very efficient. Our simulation using the IR benchmark TREC collection demonstrates that CSS outperforms GES in terms of higher recall, higher precision and lower search cost.**

Keywords: Class-based search, GES, semantic clustering, topology adaptation, P2P networks.

I. Introduction

There has been a growing interest in peer-to-peer (P2P) networks since applications like Napster and Gnutella became successful in the past few years. P2P networks can be classified into two categories according to the control over data location and network topology: *structured and unstructured.* Structured P2P systems such as CAN [10], Pastry [11], and Chord [13] can guarantee to find existing data and provide bounded data lookup efficiency. However, it suffers from high overhead to handle node churn, which is a frequent occurrence of node joining/leaving. In addition, these systems support exact-match lookups well but are not suitable for full-text content search. Unstructured P2P systems such as Gnutella are more flexible in that there is no need to maintain special network structure, and they can easily support complex queries like keyword/full text search. The drawback is that their routing efficiency is low because a large number of peer nodes have to be visited during the search process.

This work was supported in part by NSF grants, ANI 0073736, EIA 0130806, CCR 0329741, CNS 0422762, CNS 0434533, CNS 0531410, and CNS 0626240. Email: {jhuang4@, xli@cse., jie@cse.}fau.edu

A number of search techniques have been proposed for unstructured P2P networks. They are either blind searches like random walk [7], or informed searches like routing indices [5]. These techniques are designed to eliminate high-cost flooding. Moreover, some extensions have been made to strengthen these techniques, such as k-walker random walk [7], iterative deepening [15], directed BFS [15], and so on. However, none of the above utilize VSM (vector space modeling) or LSI (latent semantic indexing), so they are not suitable for semantic searching. Some searching techniques introduce a semantic-based search mechanism which supports topic (keyword or full-text) searching, such as [12], [17]. Some of them utilize VSM, others do not. Some are hierarchically centralized, others are decentralized.

The topology of an unstructured P2P network is a random graph. Recently, some schemes [6], [9], [14] have been proposed to build a semantic overlay on top of the P2P overlay. In the semantic overlay, semantically related nodes are connected to each other and form a semantic group. Gnutella-like efficient searching system (GES) [18] proposed by Zhu et al. summarizes all the documents at each node into an average term vector (named node vector) based on VSM. Its features include, (1) semantic clusters (nodes are organized into clusters according to their node vectors), (2) semantic or random link (physical link which connects two nodes whose node vectors are semantically relevant or irrelevant, respectively), (3) node-based topology adaptation algorithm, and (4) node-based search protocol which combines biased walk and flooding. The drawback of GES is that if there are different categories of documents on a node, the node vector representation may be inaccurate. For example, files on a node may fall under music class, book class, and sports class.

In this paper, we present a distributed, dynamic, class-based semantic searching system (CSS) in unstructured P2P networks. CSS extends GES by clustering all documents on a node into different classes. The goal of CSS is to make searching more efficient, which means achieving higher recall with lower search cost. CSS builds virtual semantic groups and uses an informed search. CSS uses class vectors to represent document classes on a single node. The virtual short and long links replace physical semantic and random links in GES. With the above variation, we develop a new class-based topology adaptation algorithm and a class-based search protocol which directs a query to the most relevant virtual semantic group.

The summary of our contributions in this paper

21st International Conference on Advanced Networking and Applications(AINA'07)
0-7695-2846-5/07 $20.00

follows:

- We develop a new distributed, dynamic, and class-based IR search system. The new concepts of class vector and virtual link are introduced. The class-based topology adaptation algorithm has virtual nodes (namely classes on each node) organized into virtual semantic groups through virtual links between classes. The class-based search protocol for CSS is effective in retrieving relevant documents.
- We take advantage of state-of-the-art document clustering algorithms, in particular online spherical k-means clustering (OSKM) [16], to cluster documents at each node efficiently and precisely.
- We use the IR benchmark TREC collection to evaluate our search system. The experimental results show that our system is more efficient than GES in all cases.

The rest of the paper is organized as follows. We survey related work in Section II. The preliminaries about VSM and GES are introduced in Section III. We describe the design of each component of CSS in Section IV. Simulation results are presented in Section V. The paper is concluded and future work is discussed in Section VI.

II. Related Work

An informed search such as routing indices [5], directed BFS [15], and local indices [15] achieves more efficiency than a blind search by having each node forward a query to a subset of neighbors based on previous query results or the summaries of documents stored in neighbors. Similarly, in CSS, nodes replicate the information about the class vectors of each neighbor and use directed walks. However, they differ in the information kept for neighbors.

Recently, the semantic-based search mechanism has become widely used in literature. Zhou et al. [17] extended Gnutella, a well-known P2P system, by adding a content-based relevance mechanism. The idea is to estimate the relevance of peers locally when receiving query messages. Only those peers deemed as relevant will receive the forwarded query. The similarity between the query model and the document collection model is then measured. Our search protocol is inspired from this method but takes a further step by directly calculating the similarity between the query vector and the class vector, which is more accurate. Shen et al. [12] proposed a semantic-based document search system which is based on a hierarchical summary structure. This system supports semantic-based content searching by utilizing VSM and LSI, as in CSS. However, it requires the underlying P2P architecture to be constructed with superpeers. It is a hierarchically centralized system in which a centralized index is maintained at a server in superpeers, and all queries are directed to it first. In contrast, our system CSS is, in essence, totally decentralized.

Search techniques based on semantic groups (clusters) are also very useful. A semantic overlay is built on top of the P2P network before the search process. In [6], the semantic overlay network (SON) is built as follows. First, the documents at each node are classified and a document hierarchy is spread throughout the network. Second, the SON (clusters) is built according to that document hierarchy. Finally, each new node joins the SON in a Gnutella fashion (flooding) by finding a proper cluster. In [14], a small world structure is built and maintained by using a gossiping mechanism. Each peer node periodically sends out a query containing its own profile. When it receives an answer to its query, it will analyze the answer to decide whether to add the candidate node to its neighborhood or not. In some cases, other neighbors may have to be replaced due to size constraint. The topology adaptation algorithm in CSS is inspired partly from the above articles. However, it is different in that it uses random walk, not flooding, to build semantic groups and its relevance calculation formula differs from that in [14]. In [9], Ng et al. proposed a query routing model called firework query model on top of the clustered P2P network. It clusters P2P networks based on the characteristics of peer nodes and introduces the notion of attractive and random links, which resembles CSS. However, geographical information is used to determine the similarity between peer nodes.

The topology adaptation algorithm in CSS bears similarity to the following papers. In [2], SETS tries to reorganize the network topology so that topic-related peers are close to each other. However, in SETS, a single designated node is responsible for clustering nodes into topic segments, which is actually not distributed. In [4], Gia uses a topology adaptation algorithm to balance the capacity of the network. We borrow the idea of the three-way handshake protocol for node join/leave from [4] and simplify it by removing the satisfaction level. The topology adaptation algorithm in CSS can be viewed as the extension to that in GES [18]. However, they differ from each other as follows. First, CSS uses class-based virtual links to assist topology adaptation algorithm while GES is node-based. Second, CSS reduces two host caches (semantic and random caches) in GES into one since there is no corresponding concept of semantic and random links in CSS. Finally, CSS uses a novel formula to calculate the relevance score between nodes. The score is crucial to our topology adaptation algorithm.

There are many document clustering algorithms. They can be classified as k-means, fuzzy c-means, hierarchical clustering and mixture of Gaussian. K-means is an exclusive clustering algorithm. Online spherical k-means clustering (OSKM) [16] is an online version of the spherical k-means algorithm based on the well-known winner-take-all competitive learning. We choose it to cluster documents at each node because it can achieve much better clustering results than many others and its implementation is not too complex to handle.

III. Preliminaries

A. VSM

The vector space model (VSM) [3] is a way of representing documents through the words (terms) that they contain. It is a standard technique in information retrieval. In VSM, each document or query is represented using a term frequency vector. In each vector, the terms are stemmed with stop words (functional words like “is”, “for”, “the”) removed. Suppose a collection includes two documents. The content of d

Indexed Term	all	back	dog	fox	jump	lazy
Document 1	0	1	1	1	1	1
Document 2	1	0	0	0	0	0
Indexed Term	men	now	over	quick	run	time
Document 1	0	0	1	1	0	0
Document 2	1	1	0	1	1	1

TABLE I
AN EXAMPLE SHOWING VSM REPRESENTATION.

one is "The quick fox jumped over the lazy dog's back". The content of document two is "Now is the time for all men to run quickly". Table I shows the VSM representation of the collection. In general, a collection of n documents $(D_1, D_2, \cdots, D_n)$ with t distinct terms $(T_1, T_2, \cdots, T_t)$ can be represented by a (sparse) matrix, in which w_{ij} means the weight of term i in document j.

$$\begin{bmatrix} & T_1 & T_2 & \cdots & T_t \\ D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{bmatrix}$$

In addition, each term is assigned a weight that reflects its importance in describing the document content. Among many term weighting schemes, the "dampened" tf scheme weighs each term in the form of $1+\log(tf)$ (tf means term frequency). It does not require global information, which fits semantic search in P2P networks.

There are many different ways to measure how similar two documents are, or how similar a document is to a query. The cosine measure is a very common similarity measure. Given a normalized document vector D and a normalized query vector Q, the relevance score is calculated as:

$$REL(D,Q) = \sum_{t \in D,Q} f_{t,D} \times f_{t,Q} \quad (1)$$

In the above formula, t is a common term occurring in both D and Q. $f_{t,D}$ is the weight of term t in D and $f_{t,Q}$ is the weight of term t in Q.

B. GES Overview

GES [18] is a distributed, content-based IR system proposed by Zhu et al. There are four components in GES that are also in CSS. However, they are different in that CSS introduces the new concept of class vector and makes all these components class-based. In addition, a query is routed through virtual links in CSS rather than through physical links in GES. The following is a brief summary of the GES version of these components.

Node Vector and Physical Link: A node vector is a compact outline of all documents on a node, which is indeed an average normalized term vector derived from the term vectors of all documents on a node. To judge whether two nodes or a node to a query are relevant, the cosine similarity measure between node vectors and query vector is used. A physical link is a link that connects two nodes in the P2P network. Physical links can be further classified as semantic (relevant) or random (non-relevant) links based on the relevance score of the node vectors of the two adjacent nodes.

Topology Adaptation Algorithm: The goal is to organize relevant nodes into semantic groups through semantic links. The algorithm is implemented in a distributed manner. Each node periodically issues two random walk query messages that contain its node vector. One is for nodes whose node vectors are sufficiently relevant to the node vector of the query source, the other is for non-relevant nodes. These two kinds of candidate nodes selected by the query are put into the query source's semantic and random host caches, respectively. After that, each node periodically checks these two caches for semantic or random neighbor addition/replacement based on the relevance score calculated from node vectors. When the number of neighbors reaches the maximum limit, an existing neighbor has to be dropped before adding a new neighbor.

Selective One-Hop Node Vector Replication: Each node maintains the node vectors of its random neighbors to assist the informed search process. The node vectors of semantic neighbors are not replicated.

Search Protocol: GES uses a biased walk rather than a random walk to forward a query through random links. Each node looks up its local documents satisfying the query. If at least one relevant document is found on a node, this node is called *semantic group target node*. This target node terminates the biased walk and starts flooding the query along its semantic links. If no relevant document is found, the node forwards the query to the random neighbor whose node vector is most relevant to the query vector. This two-stage search protocol makes a query walk into a proper sematic group and then retrieves many useful responses within it. Besides, the book-keeping technique is also used in GES to sidestep redundant paths.

IV. SYSTEM DESIGN

A. Overview

In CSS, each node contains several classes of documents and has corresponding class vectors as a summary of the semantic content on each node. Each class may have two types of virtual links, short and long links that connect with similar and dissimilar classes on its neighbor nodes respectively. The topology adaptation algorithm reorganizes the network according to the similarity of the contents on different nodes. The class-based search protocol routes a query through long links to the most relevant virtual semantic class group and then floods the query within that group to retrieve relevant documents.

B. Class Vector and Virtual Link

A class vector is a centroid vector of all documents in a class. We calculate class vectors on a node based on VSM as follows. First, a term vector is derived to represent a document, in which each term's weight is assigned by its frequency in that document. Second, we re-weigh each term using "dampened" tf scheme in the form of $1 + \log(tf)$. Third, we n

the weighed term vector to unit length. Fourth, we feed all processed term vectors (corresponding to all documents on a node) to the OSKM [16] clustering algorithm. Finally, given the number of classes you want to cluster (e.g., 6), the algorithm outputs the given number of normalized class vectors and a list showing which document belongs to which class.

Given two classes of documents (class X and Y), their relevance score is the cosine similarity of their normalized class vectors listed below:

$$REL(X,Y) = \sum_{t \in X,Y} w_{t,X} \times w_{t,Y} \quad (2)$$

In this formula, t is a common term occurring in both class vector X and class vector Y. $w_{t,X}$ is the weight of term t in X, and $w_{t,Y}$ is the weight of term t in Y. If the relevance score is no less than a certain threshold, these two classes are considered relevant, otherwise not.

Sometimes we need to define the relevance between a normalized class X and a normalized query Q. The following formula applies:

$$REL(X,Q) = \sum_{t \in X,Q} w_{t,X} \times w_{t,Q} \quad (3)$$

In CSS, we build virtual links on top of physical links. Physical links are the P2P overlay links that connect peers. Virtual links connect two classes on different nodes. Formally, let E be the set of physical links and $E^{'}$ be the set of virtual links. Thus, CSS makes a many-to-one mapping from $E^{'}$ to E. It means that many virtual links can be mapped to one underlying physical link.

The goal of conceptual virtual links is to connect classes of documents on different nodes virtually. If the relevance score between two classes is no less than $short\_rel\_thres$, we build a virtual link between them and call it *short link*. If the relevance score between two classes is no more than $long\_rel\_thres$, we build a virtual link between them and call it *long link*. Note that it is necessary for each document class on a node to have at least one long link to each of its neighbors because otherwise the directed walk in the search protocol would not work. So if the relevance scores of all virtual links coming from one class are higher than $long\_rel\_thres$, we just pick the link with the lowest score as the long link.

Therefore, we do not classify physical links like in GES. Instead, we classify virtual links as short and long links in CSS. The short and long links can be considered as the extension of semantic and random links in GES. Note that we use two thresholds to classify virtual links instead of the $node\_rel\_threshold$ in GES. The reason is that we do not want to build a virtual link between classes with relevance scores not high or low enough (e.g., 0.5). Using two thresholds yields better classification.

Fig. 1 shows an example of a physical link and three virtual links. The relevance score between class 1 on node X whose content is about baseball and class 1 on node Y whose content is about football is higher than $short\_rel\_thres$, so a short link is built since these two classes both belong to sports. On the contrary, the relevance score between class 1 on node X and class 2 on node Y whose content is about cooking is lower than $long\_rel\_thres$. So a long link is built between them because the two classes are not relevant. There is no virtual link built between class 1 on node X and class 3 on node Y because they have medium relevance.

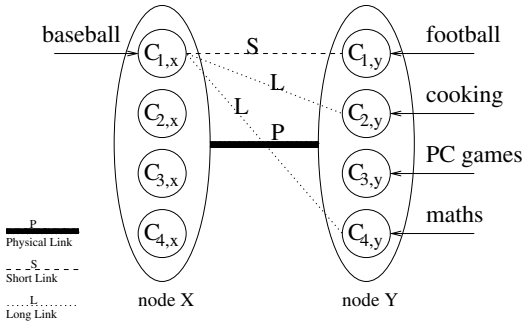


Fig. 1. Class vector and virtual link ($C_{1,x}$ means class 1 on node X).

C. Topology Adaptation Algorithm

The topology adaptation algorithm is an important part of our search system. It aims not only to maintain node connectivity but also to achieve a refined network topology for better search performance. Simply speaking, it aims to find "good" neighbors for each node in a distributed manner. The main goal of the topology adaptation in GES [18] is to ensure that relevant nodes are organized into semantic groups which may be relevant to the same queries. Our topology adaptation algorithm has to consider more factors since class vectors on a node play an important role. Our goal is to ensure that (1) relevant classes on different nodes are connected through virtual similar links (namely short links), and that (2) each class should have enough virtual dissimilar links (namely long links) in order to discover proper virtual semantic groups. Short links and long links are both valuable because during the search process a query is flooded through short links and directly routed through long links. The simulation result shows that they both affect the performance greatly. So we need a criteria to judge whether a node is a good candidate to be a neighbor or not. A formula is designed to calculate the $overall\_score$ between two nodes. We name it $overall\_score$ since it considers both relevance factor (short links) and non-relevance factor (long links). Given two nodes, the $overall\_score$ is defined as follows:

1) Find all short and long links by calculating the relevance scores between all class vectors of the two nodes using formula (2). The relevance scores of a short link and a long link are denoted by rel_score_short and rel_score_long, respectively.
2) Define a medium value:

$$rel\_med = (short\_rel\_thres + long\_rel\_thres)/2$$

3) Sum up the differences between rel_score_short and rel_med for all short links. Similarly, the differences between rel_med and rel_score_long for all lo

are summed up:

$$sum\_diff\_short = \sum_{all\ short\ links} (rel\_score\_short - rel\_med)$$

$$sum\_diff\_long = \sum_{all\ long\ links} (rel\_med - rel\_score\_long)$$

4) Add a weight factor w:

$$overall\_score = sum\_diff\_long + w \cdot sum\_diff\_short$$

When a node joins the network, it first randomly connects to other nodes using a bootstrapping mechanism as in Gnutella. At this time, it is not aware of any content (e.g., class vectors) of other nodes and its classes do not belong to any virtual semantic group. This node then periodically issues a topology query message for such information. The query is routed throughout the network using random walk bounded by a TTL (time to live) until sufficient responses are obtained. This kind of query is different from the search query. So we name it *probe query*. A probe query message contains all class vectors of the node, the maximum number of responses and TTL. The query returns some qualified nodes ranked in decreasing order of $overall\_score$ which will be added to the query initiator's host cache. Each entry in the host cache consists of a node's IP address, node degree, class vectors and $overall\_score$. The cache is continuously updated during the lifetime of a node.

With candidate nodes stored in host cache, each node periodically performs neighbor addition and replacement in a similar way to Gia [4]. After a neighbor is added, short and long links for that neighbor will be established immediately. Similarly, before a neighbor is dropped, all short and long links associated with that neighbor have to be removed. A node (say X) chooses a candidate node with the highest $overall\_score$ from its host cache and verifies that the candidate is still active and not an existing neighbor. X then uses a three-way handshake protocol to communicate with the selected neighbor candidate, say Y. It is a distributed handshake protocol, which means that each node decides independently whether to accept the other node as a new neighbor or not. These nodes make a decision according to their own max_links (the maximum number of neighbors allowed), current degree and $overall\_score$ of the requesting node. If current degree is less than max_links, the node automatically accepts the requester as a new neighbor. Otherwise, the node has to check whether it can find a suitable existing neighbor to drop and replace it with the requesting node. X makes such a decision as follows. From all of X's neighbors that are not poorly connected and whose $overall\_scores$ are lower than that of Y, X chooses the neighbor Z with the lowest $overall\_score$ to drop and adds Y as its new neighbor. (A poorly connected node is a node whose degree is less than or equal to the minimum number of neighbors required, namely min_links.)

Each node periodically detects whether the content of its neighbors has changed or not and keeps updating the $overall\_score$ of all of its neighbors. If many documents on a node have been added, removed or changed, then the clustering algorithm is run again on that node and the class vectors are updated. Each node detects the changes and recalculates $overall\_scores$ by obtaining updated class vectors from its neighbors. If the $overall\_score$ is too low (e.g., lower than a certain threshold), in order to keep all short and long links intact we do NOT simply drop that neighbor. Instead, we wait for a good candidate node to replace that neighbor during topology adaptation. In summary, our topology adaptation algorithm can fit a dynamic situation well.

D. Selective One-Hop Class Vector Replication

Each node should store information about the class vectors of all its neighbors in order to assist the search process. We only store class vectors of the classes connected via long links. That is why we call it selective replication. During the search process, queries are routed through one of the long links (need to compare and select one) and flooded through all short links (no need to compare and select). So, we need to replicate information about long links for selection purposes. If a neighbor leaves the network, then the information about its class vectors will be deleted. If the documents on a node's neighbors have been changed, it will receive the updated class vectors and then recalculate $overall\_score$. In summary, our replication algorithm can handle dynamic situations such as node join/leave or the change of documents on neighbor nodes.

E. Search Protocol

The topology adaptation algorithm refines the network topology, and selective one-hop class vector replication informs each node of the class vectors of its neighbors. We then discuss our content-based virtual link assisted search protocol. The protocol is totally class-based, which means that queries are routed from one class to another along virtual links.

A query can be in two separate modes during the search process. One is *directed walk* mode (walk with each hop selected intelligently) and the other is *flooding* mode. When a node initiates a query, it first calculates and compares the relevance scores between the query vector and all its class vectors. The query is then routed to the class with the highest relevance score. This class is called the *query source class*.

After the query source class is found, the query is set to be in directed walk mode. When receiving a query in this mode, each class looks up its locally stored documents for those satisfying the query. A relevance score is calculated between the query and each document using formula (1). If it is higher than a certain threshold, this document is identified as a relevant document for the query. If at least one relevant document is found, then this class (say A) of the node (say X) is called a *virtual semantic group target class*. At this time, the query ends directed walk mode and enters flooding mode. If all documents in class A are identified as non-relevant, class A selects a class (say B) whose class vector is most relevant to the query vector according to formula (3) from all other classes in the same node X and classes connected via X's long links. Fig. 2 illustrates how node X intelligently chooses a class to forward the query. The query Q_a begins with the class $C_{2,x}$ of node X and no documents in $C_{2,x}$ are relevant to the query. So class $C_{2,x}$ tries to find the most suitable class to biasedly forward the query. Its choices include all other classe

Algorithm 1 Search

```
1: A query reaches the source node X
2: /* Initialization */
3: The query is located in the identified query source class
4: set query_mode ← directed walk
5: while insufficient responses and TTL bound not reached do
6:    if query_mode = directed walk then
7:       look up relevant documents in the current class
8:       if one relevant document found then
9:          mark current class virtual semantic group target class
10:         set query_mode ← flooding
11:      else
12:         define set C ← {all other classes in the same node X} ∪
            {classes connected via node X's long links}
13:         forward the query to the class in set C which is most
            relevant to the query
14:      end if
15:   end if
16:   if query_mode = flooding then
17:      flood the query along all short links
18:      if no short link found or beyond flood radius then
19:         set query_mode ← directed walk
20:      end if
21:   end if
22: end while
23: return
```

same node X ($C_{1,x}$, $C_{3,x}$, $C_{4,x}$) and classes connected via X's long links ($C_{4,y}$, $C_{2,z}$, $C_{4,z}$). Class $C_{2,x}$ finally chooses $C_{4,y}$ since its class vector is most relevant to the query vector. The directed walk continues similarly until a virtual semantic group target class is found.

The target class then sets the query to be in flooding mode and floods the query along all its short links. Each semantically-related class receiving the query looks up all documents in the class and floods the query along its own short links. Fig. 2 shows that if $C_{2,x}$ is the target class for the query Q_b, it will flood Q_b along its short links to the classes $C_{1,y}$, $C_{3,y}$ and $C_{1,z}$. In addition, the radius of flooding is controlled via TTL. The relevant documents found within the semantic class group are reported to the target class directly. The target class is responsible for aggregating all these files and reporting them directly to the query source. If the number of relevant documents discovered so far is below user expectation specified in the original query, the target class starts another directed walk and the above search process is repeated. The whole algorithm is illustrated in **Algorithm 1**.

We also use TTL to bound the duration of the directed walk and the book-keeping technique to avoid redundant paths. In CSS, each query is assigned a unique GUID by its originator class. Each class remembers the classes to which it has already forwarded queries for a given GUID. If a query with the same GUID comes back to the class, it will be forwarded to a different class with highest relevance score in directed walk mode or it will simply be discarded in flooding mode. However, to guarantee forwarding progress, if a class has already sent the query to all possible classes, it flushes the book-keeping state and starts reusing classes for directed walk.

In summary, the directed walk guides the query towards a target class with similar semantic content. Flooding locates enough desired documents near the target class. Our class-based search protocol makes searching efficient by using a smaller search unit: a class of documents on a node instead of all documents on a node.

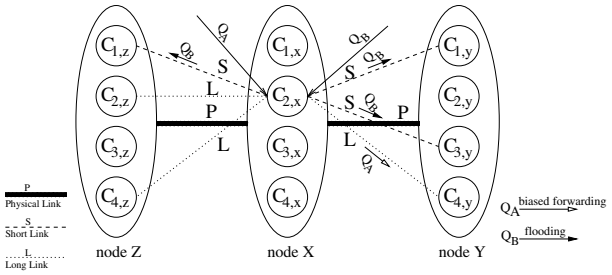


Fig. 2. Query routing graph ($C_{1,x}$ means class 1 on node X).

V. Simulation

In this section, we present the results of our class-based search system (CSS). We first discuss text datasets used in the simulation and then explain performance metrics. After that, simulation settings are described. Finally, CSS is evaluated in different network configurations and document distributions.

A. Text Datasets

We use all documents of AP newswire 1988 along with Topics 151-200 in TREC disks 1&2 to evaluate our search system. They can be found in TREC's ad hoc test collections [1]. The goal of TREC is to provide a benchmark for evaluation in information retrieval from large text collections. We then extract the text field from these documents by removing irrelevant parts. Rainbow [8], a powerful toolkit to preprocess raw text files for further classification and clustering, is then used to calculate the term frequencies for all terms (words) in these documents. Rainbow is also able to perform a stemming process with the aid of Porter stemmer. Finally there are 79,986 files distributed over 1,000 nodes. The queries we use are from TREC topics 151-200. The query vectors are obtained from the text field of these topics using Rainbow. So each query is not a keyword-based search but a text-based search. The query vectors are also stemmed with stop words removed. In addition, the TREC website [1] provides relevance judgment files that can be viewed as "correct answers" for the above 50 queries. These assessment files are obtained by manual identification and are vital to our simulation.

B. Performance Metrics

The following performance metrics are used in the simulation.

1) Recall: It measures the coverage of available relevant results. It is defined as the number of relevant documents retrieved divided by total number of relevant documents in the system.
2) Precision: It is defined as the number of truly relevant documents divided by total number of documents retrieved.

3) Search cost: It is defined as the percentage of classes in the network visited by a query. An efficient search algorithm incurs a low search cost. The less classes visited by a query, the shorter the search response time and the less the computing resource consumption is, especially when there are a huge number of documents in the network.

C. Comparison Criteria

As described in the search protocol section, CSS is a class-based search system while GES is node-based. This means that CSS routes a query from one class to another while GES routes a query between nodes. Therefore, how can we compare the performance of CSS and GES using the same criteria? First, we globally cluster all documents in the text datasets into 20 classes by using OSKM [16]. Second, a tunable parameter, cpn is defined to reflect different document distribution. cpn is the abbreviation of $classpernode$ which is the number of classes placed at each node. If cpn equals 5, there are 5 classes of documents per node, which means that we randomly pick up some documents from 5 classes of the above 20 global classes and assign them to nodes. Third, the metric we use is $percentage\ of\ classes\ visited$. There is no problem with class-based CSS. As for GES, we turn to the conversion that the number of classes visited equals the product of the number of nodes visited and cpn. As a result, the same metric can be applicable to both search systems.

D. Simulation Setting

We use a custom simulator to test our search system. A random graph with average degree 6 is generated first as the initial topology and then the topology adaptation algorithm is run for several rounds to reorganize the initial topology. The search process is then performed on the refined topology. In our search system (CSS), there are two fixed parameters (min_links and max_links) and five tunable parameters ($classpernode$ or cpn, $short\_rel\_thres$, $long\_rel\_thres$, w, $good\_neigh\_thres$). We set $min\_links = 3$ and $max\_links = 10$ which are min. and max. limit for node degree. The tunable parameter cpn is set from 1 to 10. $short\_rel\_thres$ and $long\_rel\_thres$ represent two thresholds to define short and long links respectively. We set their values to 0.7 and 0.3, respectively. The weight factor w ($w = 3$) is used to strengthen the effect of short links when calculating $overall\_score$ because the number of short links is much less than that of long links. We use 3.0 as the value of the last tunable parameter $good\_neigh\_thres$, which is the criteria for "good" candidate nodes to be put into the cache in the probe query process.

All values of the above tunable parameters are obtained from experiments so they are heuristic values. However, there are some rules why we choose these particular values. We set the value of $short\_rel\_thres$ larger so that each short link is built between two classes that are very relevant. This makes flooding short links very effective in finding really relevant documents. As for the value of $long\_rel\_thres$, if it is too small, then the number of long links is too few for a query to find a relevant enough class to route itself to. If it is too large, then there will be too many long links for directing a query, which consumes more resources and causes longer delay. Finally, if the host cache is large, we can set $good\_neigh\_thres$ smaller so that more candidate nodes can be considered.

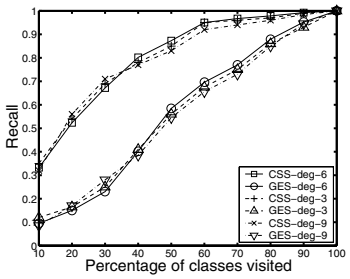


Fig. 4. CSS vs. GES with different node degree (cpn = 5, nodenum = 1000).

E. Simulation Results

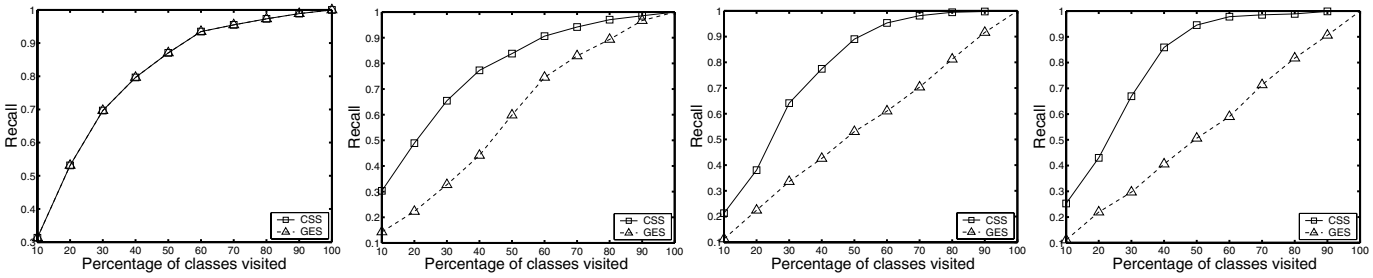
We compare the performance of our search system (CSS) with GES in different network configurations and document distributions.

Fig. 3 shows the performance with different values of $classpernode$ (cpn) ranging from 1 to 10. Four typical graphs with cpn = 1,4,7,10 are illustrated. We can see that (1) CSS performs better than GES when cpn ranges from 1 to 10. If cpn equals 1, then CSS is reduced to GES and they behave in the same way. So the search results are identical for $cpn = 1$ in Fig. 3 (a). However, in Fig. 3 (b)-(d), our search system CSS outperforms GES by achieving higher recall at the same search cost. This can be explained as follows. Since a query is routed through classes instead of all documents on a node, it can locate the target class more precisely and quickly without visiting many irrelevant documents. (2) The trend from Fig. 3 (b)-(d) indicates that CSS performs better than GES when the number of classes at each node increases. This is because the advantage of clustering documents into classes will be more obvious as the number of classes increases. The more classes on a node, the more choices for a query to investigate and the higher precision the search has. We demonstrate that CSS outperforms GES in different document distributions.

Fig. 4 compares the performance in different initial node degree (deg = 3,6,9). We can see that CSS still beats GES and the difference between the group of curves of CSS or GES is very small. This is because the topology adaptation algorithm plays a significant role in addition and replacement of neighbors so that initial node degree seems less important.

Fig. 5 compares the performance with different numbers of nodes in the network ($nodenum$ = 500,1000,2000). We can see that CSS still exceeds GES and the difference between the group of curves of CSS or GES is relatively larger than that in Fig. 4. That is because the number of nodes in the network has a direct impact on topology adaptation and search process. CSS proves to be applicable to P2P networks of different sizes.

Fig. 6 presents a graph of precision vs. recall. It is a standard measure in information retrieval. We can observe



(a) Number of classes per node = 1 (b) Number of classes per node = 4 (c) Number of classes per node = 7 (d) Number of classes per node = 10

Fig. 3. CSS vs. GES with different number of classes per node (nodenum = 1000, deg = 6).

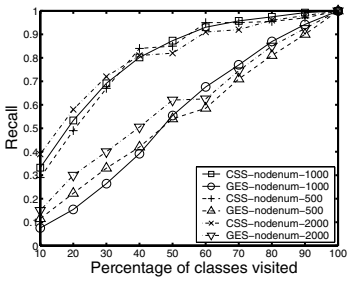


Fig. 5. CSS vs. GES with different node number in the network (cpn = 5, deg = 6).

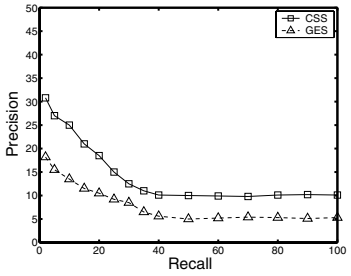


Fig. 6. Precision vs. Recall (cpn = 5, nodenum = 1000, deg = 6).

precision of CSS is higher than that of GES in the same recall, which means CSS is more precise in the search process due to its class-based search protocol. However, the highest precision is around 32%, which seems low. This is because the truly relevant documents for a query in the relevance judgment file are only around one thousandth of the whole document collection (text datasets). Besides, we aim to conduct a full search to find all truly relevant documents since there are not many. This is why we can not set a high threshold for document retrieval.

VI. Conclusion

In this paper, we extend GES and present a class-based search system (CSS) in Gnutella-like unstructured P2P systems. CSS exploits a state-of-the-art data clustering algorithm and makes each component class-based. The simulation shows that CSS is more efficient than GES in all cases. In summary, CSS is more suitable when the documents at each node are diverse while GES is applicable when the documents are uniform. In the future, we will conduct more simulations using larger datasets and different metrics such as precision@10.

References

[1] Text REtrieval Conference (TREC). http://trec.nist.gov.

[2] M. Bawa, G. Manku, and P. Raghavan. Sets: Search enhanced by topic segmentation. In *Proceedings of the 26th Annual International ACM SIGIR Conference*, pages 306–313, Toronto, Canada, 2003.

[3] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, and N. Lanham. Making gnutella-like p2p systems scalable. In *Proceedings of ACM SIGCOMM*, pages 407–418, Karlsruhe, Germany, Aug 2003.

[5] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing (IEEE ICDCS'02)*, 2002.

[6] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, University of Stanford, May 2004.

[7] Q. Lv, P. Cao, and E. Cohen. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th ACM International Conference on Supercomputing (ACM ICS'02)*, pages 84–95, New York, Jun 2002.

[8] A. McCallum. Rainbow toolkit. www.cs.cmu.edu/∼mccallum/bow/.

[9] C. H. Ng and K. C. Sia. Peer clustering and firework query model. In *Proceedings of WWW*, 2002.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, Nov 2001.

[12] H. T. Shen, Y. Shu, and B. Yu. Efficient semantic-based content search in p2p network. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):213–236, 2004.

[13] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[14] H. F. Witschel. Content-oriented topology restructuring for search in p2p networks. Technical report, University of Leipzig, Germany, 2005.

[15] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing (IEEE ICDCS'02)*, 2002.

[16] S. Zhong. Efficient online spherical k-means clustering. In *Proceedings of IEEE Int. Joint Conf. Neural Networks (IJCNN 2005)*, pages 3180–3185, Montreal, Canada, Aug 2005.

[17] Y. Zhou, W. B. Croft, and B. N. Levine. Content-based search in peer-to-peer networks. Technical report, University of Massachusetts, 2004.

[18] Y. Zhu, X. Yang, and Y. Hu. Making search efficient on gnutella-like p2p systems. In *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS'2005)*, Denver, Colorado, Apr 2005.