Information Retrieval and Query Routing in Peer-to-Peer Networks

WONG Wan Yeung

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

© The Chinese University of Hong Kong
June 2005

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or whole of the materials in the thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.

Abstract

Recently, information retrieval based on Peer-to-Peer (P2P) networks is becoming a popular and dynamic research topic. In this work, we study two scenarios with their own problems and we solve them by our proposed *Site-to-Site (S2S) Searching* in a pure P2P network and *GAroute* in a hybrid P2P network respectively.

The first scenario is that Web information retrieval by Centralized Search Engines (CSEs) like Google have three shortcomings. First, CSEs are centralized so that they require expensive resources to handle search requests. Second, the search results are not always up-to-date. Third, website owners have no control over their shared contents such as preventing published contents from being searched. To circumvent the aforementioned shortcomings, we refer to Gnutella to distribute search engines called S2S search engines over websites (peers) in a pure P2P network, which maintain their updated local contents with full control by their owners. Hence, each website becomes an autonomous search engine and they join together to form a S2S Searching network. In this thesis, we show the system architecture, indexing and matching algorithms of S2S search engines. In addition, we explain our query routing algorithm based on *distributed registrars* which prevents the *query flooding problem* existing in Gnutella. Furthermore, we describe our S2S communication protocol

which is based on Common Gateway Interface (CGI). Finally, we present our experimental results which show that S2S Searching is scalable (approximate linear) in some large S2S networks.

The second scenario is that content-based information retrieval in hybrid P2P networks like YouSearch using the *Direct Connection Model (DCM)* has two shortcomings. First, the query initiating peer consumes high bandwidth for its own network transmission. Second, we have poor semi-parallel search if there are too many relevant peers. To circumvent the aforementioned shortcomings, we introduce the *Query Propagation Model (QPM)* into YouSearch and form a new hybrid P2P network. In order to obtain optimal query routing paths in our network, we model our problem as the *Longest Path Problem* which is NP-complete and we propose a Genetic Algorithm (GA) called GAroute to obtain high quality approximate solutions in polynomial time. In this thesis, we describe our network and GAroute algorithm. In addition, we introduce two novel GA operators called *fission* and *creation*, and also an optimization technique called *two-phase tail pruning* to improve the quality of solutions. Finally, we present our experimental results which show that GAroute achieves both good scalability (approximate linear) and quality (0.95 in 100 peers) in some large scaled P2P network topologies. Moreover, the query initiating peer network traffic reduces more than 90 percents for 1,000 peers and the semi-parallel search problem is greatly improved.

論文摘要

近來，以點對點網路進行資訊檢索正成爲一個普遍和有活力的研究題目。在是次研究當中，我們研究兩個不同的個案，它們各有不同的問題，但我們分別用我們所建議的「站對站搜尋」(Site-to-Site Searching)用於純粹點對點網路上，和「嘉禾」(GAroute)用於混合點對點網路上來解決這兩個個案的問題。

第一個個案是：以集中式搜尋器如 Google 進行資訊檢索會有三個缺點。首先，集中式搜尋器需要昂貴的資源來處理所有搜尋要求。其次，搜尋的結果總不是最更新的。最後，站長沒有他們分享內容的控制權，例如防止已發表的內容被他人搜尋。爲了改善以上的缺點，我們參考過 Gnutella 的做法來分佈一些名爲站對站的搜尋器到不同的網站上，以成爲一個純粹點對點的網路，那些搜尋器會保持最更新的搜尋索引，並給予站長完全的控制權。因此，每一個網站會成爲一個自主的搜尋器，而它們可以連結在一起，以成爲一個站對站的搜尋網。在這份論文中，我們會展示站對站搜尋器的系統構造，索引和配對的演算法。另外，我們會解釋以分佈式登錄器爲基礎的詢問路線安排演算法，這演算法可有效地防止 Gnutella 的詢問氾濫問題。再者，我們會描述以通用閘道界面爲基礎的站對站通訊協議。最後，我們會展示實驗結果來證明站對站搜尋是可以大規模化(約線性增長)的。

第二個個案是：在混合點對點網路如 YouSearch 使用直接聯絡模型進行內容基礎的資訊檢索會有兩個缺點。首先，詢問者會消費較多的網

路傳輸頻寬。其次，如果有太多相關的搜尋點，詢問者會有較差的搜尋表現。爲了改善以上的缺點，我們把詢問傳播模型注入 YouSearch 內，從而產生一種新的混合點對點網路。爲了要在新的網路中獲得一些優質的詢問路線，我們把這個難題模仿成最長路徑問題，因爲最長路徑問題是完整非決定多項式的，所以我們建議一種名爲嘉禾的遺傳基因演算法，用多項式時間來獲得優質的答案。在這份論文中，我們會描述所建議的網路和嘉禾演算法。另外，我們會介紹兩個新穎的遺傳基因運算子，名爲分裂和創造，還有一種最佳化的技術，名爲二期尾部剪除法，以改善答案的質素。最後，我們會展示實驗結果來證明嘉禾是可以大規模化(約線性增長)的，並能提供高質素的答案(在 1000 搜尋點中有 0.95 質素)。此外，詢問者搜尋 1000 搜尋點能減少 90%的交通量，並大大地改善了搜尋表現。

To my dearest family

Acknowledgment

First, I would like to thank my Lord, Jesus Christ, for loving me, helping me, and leading me. I would like to give the glory to Him who bestows the wisdom upon me. Amen!

Second, I would also like to thank my supervisor, Prof. Irwin King, for helping me on my research and teaching me on writing papers. When I was an undergraduate, he was also my Final Year Project supervisor. He encouraged me to continue on studying. He taught me a lot of things, not only the academic, but also the life philosophy. I saw Jesus Christ from his love and behavior. The Bible (New International Version) says, "*For I was hungry and you gave me something to eat, I was thirsty and you gave me something to drink, I was a stranger and you invited me in, I needed clothes and you clothed me, I was sick and you looked after me, I was in prison and you came to visit me.*" [Matthew 25:35–36] These are what he did to me.

Third, I would like to thank Prof. Michael Lyu, Prof. Kwong Sak Leung, and Prof. Jiangchuan Liu for giving me suggestions on my proposed Site-to-Site Searching and GAroute.

Finally, I would like to thank my dearest family for giving me support and care from time to time.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Symbols in S2S Searching

Symbol	Meaning
A	1. The set of first alphabets in keywords matching
	2. The adjacency matrix
a	The first alphabet
CL	The confidence level of a hash table block
C	The number of collisions
c	The ASCII code of a character
d	1. The depth of a tree
	2. The document in VSM
f	The traffic reduction factor
f_{ij}	The raw frequency of the term (word) w_i in the document d_j
$f(w)$	The frequency of the word w in a document or a site
HS	The hash set of all words which have the same hash code
$H(w)$	The hash function of content summary with a given word w
I	The index table which contains file offsets and lengths of each first alphabet group
$I(w)$	The word importance of the word w in a document or a site
idf	The inverse document frequency
K	The set of keywords
k	The keyword
L	The word occurrence locations of the inverted index
l	The length of a word
m	1. The number of different first alphabets of keywords
	2. The number of blocks in a content summary hash table
N	The number of different words in a document or a site
N_a	The number of adjacent sites
N_a'	The number of adjacent sites that the query is routed
N_d	The total number of documents
N_t	The total number of index terms
n	1. The number of different keywords

	2. The degree of an *n*-nary tree
$\overline{n}$	The expected fan-out degree of a site
n_i	The number of documents in which the term (word) w_i appears
p	1. The weight of the priority in the ranking parameters
	2. The probability of infrequent query flooding
priority	The priority value of a document
q	The query
rank	The ranking value of a document
S	The content summary of a site
s	1. The weight of the similarity in the ranking parameters
	2. The similarity value between a document and a keyword in the matching algorithm
s_i	The i^{th} element of a content summary hash table
score	The score (relevance level) of a site
sim	The similarity value between keywords and a document
T_{flood}	The total traffic cost of query flooding
T_{route}	The total traffic cost of query routing
t_{ij}	The term weighting of the term (word) w_i in the document d_j
t_{iq}	The query term weighting of the term (word) w_i in the query q
tf	The term frequency
W	The set of words in a document or a site
w	The word in a document or a site

List of Symbols in GAroute

Symbol	Meaning
A	The adjacency matrix
b	The number of batches
C	The chromosome representing a path
$d(i, j)$	The shortest path distance between the nodes i and j
E	1. The set of edges in a graph representing the connectivity between peers 2. The expected number of existing peers to be linked by a new peer used in the graph generation algorithm
f	The fitness of a chromosome
G	The directed graph
G_{max}	The maximum number of generations
G_{min}	The minimum number of generations
H	The information gain of a path
L	1. The available peer list used in creation and mutation 2. The list of potential crossing points used in crossover
l	The length of a chromosome
λ_c	The crossover proportion
λ_m	The mutation proportion
λ_n	The creation rate
m	The mutation point
N	The population size
N_c	The number of crossovers in a generation
N_f	The number of fissions in a generation
N_g	The number of good chromosomes for selection
N_m	The number of mutations in a generation
N_n	The number of creations in a generation
N_{path}	The number of different paths in a graph
n	The maximum number of paths to be returned
P	The list of query routing paths

P_n	The new population after selection
P_o	The original population before selection
p	The query routing path
ρ	The penalty of a peer
Q	The quality of query routing paths relative to BS
Q'	The quality of query routing paths relative to GA
(r, s)	The crossing point
S	The score vector
T	The network traffic
T_O	The network traffic overhead
T_R	The network traffic reduction
t	The query time
(u, v)	The fission point
V	The set of vertices in a graph representing peers
V'	The set of current visited peers
x	The peer or the corresponding gene
x_1	The query initiating peer
x_{last}	The last peer or gene in a path or chromosome
y	The peer or the corresponding gene

List of Abbreviations

Abbreviation	Meaning
AS	Autonomous System
BS	Brute-force Search
CGI	Common Gateway Interface
CPU	Central Processing Unit
CSE	Centralized Search Engine
DCM	Direct Connection Model
DDR	Double Data Rate
DHT	Distributed Hash Table
FTP	File Transfer Protocol
GA	Genetic Algorithm
GS	Greedy Search
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Provider
I/O	Input/Output
NFS	Network File System
NP	Nondeterministic Polynomial
OS	Operating System
P2P	Peer-to-Peer
QPM	Query Propagation Model
RAM	Random Access Memory
S2S	Site-to-Site
TTL	Time-to-Live
URL	Uniform Resource Locator
VM	Virtual Machine
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

1. Introduction

Recently, information retrieval based on Peer-to-Peer (P2P) networks is becoming a popular and dynamic research topic. In this work, we research on both pure P2P networks and hybrid P2P networks. Pure P2P networks refer to those P2P networks without any centralized component, whereas hybrid P2P networks depend on centralized components for storing indices or content summaries of each peer. We are interested in how information can be efficiently retrieved in these networks and hence we analyze the existing solutions. However, the existing solutions have some problems in some specific cases. Therefore, we refer to the existing solutions and propose two different P2P network models with their effective query routing strategies, namely *Site-to-Site (S2S) Searching* [49], [50] and *GAroute* [51], to improve the information retrieval in the specific cases. S2S Searching is for the Web information retrieval in our proposed pure P2P network model, while GAroute is for the content-based information retrieval in our proposed hybrid P2P network model. In this chapter, we give the problem definition of the existing solutions (see Section 1.1) and the major contributions of both S2S Searching and GAroute (see Section 1.2). We also show the thesis chapter organization (see Section 1.3).

1.1 Problem Definition

Nowadays, information retrieval on the Web is popular and significant.

Web search engines become essential applications. However, Centralized Search Engines (CSEs) like Google [21], AltaVista [4], and Yahoo [55] have three shortcomings which are (1) centralization of resources used, (2) outdated search results, and (3) no control over information shared by content owners.

1) Centralization of Resources Used: CSEs are centralized which require powerful servers to handle search requests. They also need a large storage space to store crawled contents and indices. Hardware cost is expensive for achieving high performance. Take Google as an example, their centralized server contains hundreds of computers inter-connecting together. And they need many hard-disks for storing crawled contents and indices. Moreover, CSEs have single point of failure. If their servers are down, we cannot perform any search.

2) Outdated Search Results: CSEs preprocess the search by crawling Web contents and building the corresponding indices. Usually, the crawled contents and indices are outdated as Web pages are being updated from time to time [13]. The freshness of indices depends on the crawling strategy. Take Google as an example, there are often some dead and outdated links in search results.

3) No Control over Information Shared: CSEs crawl published contents on the Web and make them become searchable without their owners' permissions. The owners may only want their contents like private information to be accessed by their authorized people by giving them secret

Uniform Resource Locators (URLs). Although they can make their contents escape from crawlers by setting passwords or removing links in their Web pages, it is inflexible and requires technical knowledge. In addition, the owners cannot alter their ranking strategy for their prioritized contents. Although they can use meta-tags and different headings in their contents, it is inflexible and does not guarantee how their contents are ranked.

The aforementioned shortcomings can be circumvented by distributing search engines over peers which maintain their updated local contents with full control by their owners. Gnutella [20] is a typical protocol designed for sharing and searching files in personal computers in a pure P2P network. However, Gnutella does not have an effective query routing strategy so that queries are flooded to all peers including irrelevant peers in the network which generates a lot of network traffic and wastes resources of all irrelevant peers. This is known as the *query flooding problem* [26]. To analyze the traffic cost of such network, let us consider a simple *n*-nary tree topology (see Figure 1). The root of the tree is the query initiating peer. The depth *d* of the tree is the TTL value of the query in the query initiating peer. Let the traffic cost for sending a query between two peers be one unit. The total traffic cost T_{flood} for searching in the whole network is obviously the sum of the geometric progression such that

$$T_{flood} = \sum_{i=1}^{d} n^i = \frac{n(n^d - 1)}{n - 1} \text{ units.} \quad (1)$$

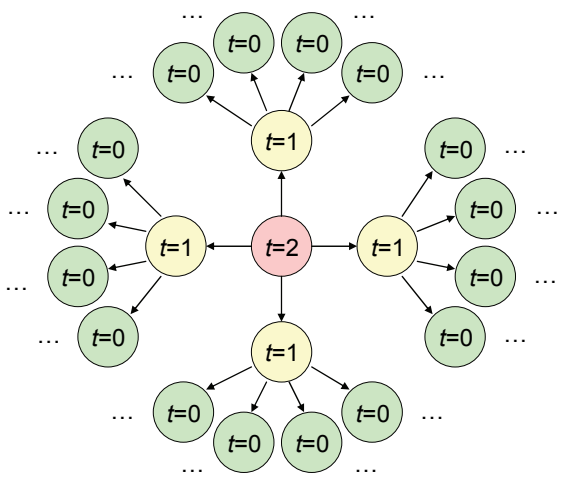

Figure 1. Simple Four-nary Tree Topology with Depth Two

In order to reduce the exponential traffic cost, routing query to relevant peers only is necessary. The query flooding problem in pure P2P networks is not only solved by some existing query routing algorithms like CAN [46] and Chord [28], but also some hybrid P2P networks like Kazaa [31] and YouSearch [36]. Pure P2P networks refer to those P2P networks without any centralized component, whereas hybrid P2P networks depend on centralized components (like *super-nodes* in Kazaa and the *registrar* in YouSearch) for storing indices or content summaries of each peer. By querying centralized components in hybrid P2P networks, each peer obtains a list of relevant peers so that it directly connects to all relevant peers to obtain document lists. Thus, the query flooding problem does not exist due to the *Direct Connection Model* (DCM). However, such model has two shortcomings which can further be improved. They are (1) high bandwidth consumption

and (2) poor semi-parallel search.

1) High Bandwidth Consumption: The query initiating peer directly connects to all relevant peers and then sends a query packet to each peer individually. If the number of relevant peers is large, then the query initiating peer consumes high bandwidth for its own network transmission especially the case of content-based multimedia retrieval [26], [12].

2) Poor Semi-parallel Search: The query initiating peer spawns a thread to concurrently handle each direct connection to a relevant peer. However, a computer has a limited thread resource, which makes parallel connections to all relevant peers impossible if there are many [3]. Although we can still semi-concurrently connect to them by spawning a few threads in each batch, it is slow as we need to wait for a batch to finish before spawning another batch. This results in poor semi-parallel search. We may circumvent this by setting the maximum number of relevant peers to be searched or reduce the number of relevant peers by increasing the relevance threshold. However, we retrieve less information by using this solution.

1.2 Major Contributions

The major contributions of our work are *S2S Searching* [49], [50] and *GAroute* [51] which circumvent the aforementioned shortcomings of CSEs and DCM.

1.2.1 S2S Searching

To circumvent the three aforementioned shortcomings of CSEs (centralization of resources used, outdated search results, and no control over information shared by content owners), we refer to Gnutella and propose a pure P2P network model for the Web information retrieval based on Common Gateway Interface (CGI). In our proposed pure P2P network model, each website is a peer and we call it a *site*. Moreover, we solve the query flooding problem of Gnutella by our proposed S2S Searching which routes queries based on *distributed registrars* for storing content summaries of adjacent sites. We also develop the S2S search engine which is an application of S2S Searching written in Java Servlet [30]. It helps site owners, whose websites are hosted by Internet Service Providers (ISPs), to turn their websites into autonomous search engines without extra hardware and software cost. Finally, S2S Searching provides (1) decentralized searching, (2) updated search results, and (3) full control over information shared by content owners.

1) Decentralized Searching: S2S Searching is decentralized so each site needs less powerful machines to handle search requests and less storage space to store the local index. A normal Web server is sufficient for a high performance searching of a site. We can use a search form in any site which joins the S2S network to start searching Web contents. The query initiating site propagates the query request to its adjacent sites. Each site propagates the request, searches its own Web contents, and gathers search results.

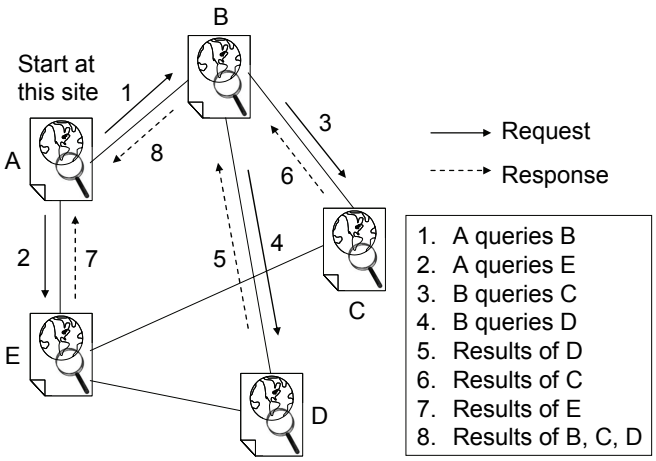

Figure 2. Query and Result Propagation in S2S Searching

Finally, all search results are propagated back to the query initiating site which are ranked and displayed to users (see Figure 2). Since S2S Searching uses CGI as the communication protocol which involves the request-response mechanism, we model the query as the request and the results as the response. In addition, there is no single point of failure. If some sites are down, we can still use other search forms in other sites. Moreover, S2S Searching skips those sites which are currently down and continues to search. Therefore, we can still obtain results from other live sites.

2) Updated Search Results: S2S Searching always provides most updated search results because each site maintains its own local index which is always up-to-date. When a local content in a site is updated, the corresponding index is recalculated. Therefore, we do not have any dead and outdated link in search results.

3) Full Control over Information Shared: S2S Searching allows site owners to fully control their information shared as they become administrators of their own search engines. They can selectively disable their published contents to be searchable so as to increase the privacy. They can also prioritize their contents and ranking strategy in order to advertise and rank results in a more customized way.

1.2.2 GAroute

The two aforementioned shortcomings of DCM (high bandwidth consumption and poor semi-parallel search) can be circumvented by the *Query Propagation Model* (QPM) which is commonly applied in pure P2P networks. By referring to Kazaa and YouSearch networks, we propose our hybrid P2P network model which is based on QPM. Instead of directly connecting to all relevant peers, the query initiating peer queries the *zone manager* (like the super-node in Kazaa and registrar in YouSearch) for some optimal query routing paths searched by our proposed Genetic Algorithm (GA), and then propagates the query to all relevant peers through these paths. Therefore, we achieve two improvements which are (1) lower bandwidth consumption and (2) better parallel search. We verify them by our experimental results (see Section 4.3).

1) Lower Bandwidth Consumption: The query initiating peer only sends query packets to the next peers in the query routing paths instead of all relevant peers. Hence, the query initiating peer consumes lower

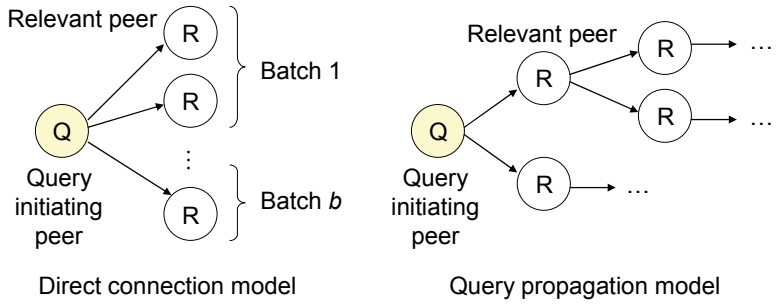

Figure 3. Comparison between DCM and QPM

bandwidth for its own network transmission.

2) Better Parallel Search: The query initiating peer only directly connects to the next peers in the query routing paths. Hence, the maximum number of threads to be spawned by the query initiating peer, which equals to the number of its logically linked peers, is greatly reduced due to the small degree property of our proposed P2P network. Hence, the query initiating peer may be able to spawn all threads at one time which results in better parallel search.

Figure 3 shows the comparison between DCM and QPM. If there are r relevant peers, the query initiating peer needs to consume r units of its own bandwidth in DCM, whereas it only needs to consume two units of its own bandwidth in QPM. If there are only two threads available, the query initiating peer needs to spawn b batches of threads for DCM, whereas it only needs to spawn one batch of threads for QPM.

Besides our proposed hybrid P2P network model and zone managers, we propose a novel GA called GAroute used by zone managers as a function

to search for some optimal query routing paths. By giving the current P2P network topology and relevance level of each peer, GAroute returns a list of query routing paths that cover as many relevant peers as possible. We model this as a *Longest Path Problem* in a directed cyclic graph which is NP-complete [37]. Nonetheless, we obtain high quality approximate solutions in polynomial time by using GA.

1.3 Thesis Chapter Organization

The rest of this thesis is organized as follows. Chapter 2 introduces the related work including different P2P networks, query routing strategies, and P2P network security. Our proposed S2S Searching and GAroute are described in Chapter 3 and Chapter 4 respectively. For S2S Searching, we describe the system architecture, algorithms for indexing, matching, and query routing, communication protocol, and experiments. For GAroute, we describe our proposed hybrid P2P network model, genetic algorithm for finding optimal query routing paths, and experiments. Moreover, Chapter 5 contains the discussion for both S2S Searching and GAroute. Finally, we give the conclusion in Chapter 6, bibliography in Chapter 7, and appendix including S2S search engine and GAroute library in Chapter 8.

2. Related Work

Since P2P applications are widely used nowadays, there are some related research on P2P technologies. In this chapter, we give a literature review on different P2P networks (see Section 2.1) and query routing strategies (see Section 2.2). We also compare them with our proposed S2S Searching and GAroute. Finally, we briefly introduce the research that is related to P2P network security (see Section 2.3).

2.1 P2P Networks

Mostly, P2P networks are used for sharing contents like audio, video, software, and other data files. P2P networks are logical networks that rely on computing power at the ends of a connection rather than in the networks themselves [41]. In Chapter 1, we mention that P2P networks can be divided into two types which are pure and hybrid P2P networks. Pure P2P networks do not have the concept of clients and servers. Instead, they have the concept of equal nodes which act like both “clients” and “servers” simultaneously to other nodes in the network. We call those nodes *peers*. The main difference between the client-server network model and P2P network model is that the data is sent between a client and a centralized server for the client-server network model, whereas the data is sent between peers for the P2P network model. Hybrid P2P networks use the client-server network model for some functions such as searching file indices, and use the

P2P network model for other functions such as downloading files. They are called “hybrid” because they combine the client-server and P2P network models. Hybrid P2P networks may have the scalability problem due to the use of centralized servers. However, there are three advantages of P2P network model over client-server network model which are distributed resource, increased reliability, and comprehensiveness of information [26]. In this section, we introduce eight different P2P networks from generation to generation. They are (1) Napster [40], (2) Gnutella [20], (3) Kazaa [32], (4) BitTorrent [8], (5) Gnutella2 [21], (6) YouSearch [36], (7) Discovir [26], and (8) Freenet [17].

Before the start of P2P, information sharing is usually through websites. When we search for some files, we go to some search engine websites [21], [4], [55] and search which return several lists of relevant websites. Then we go through some of these relevant websites to download the files that we want. This is the client-server network model where the search engines and relevant websites act as the servers. Obviously, this scenario has a major shortcoming that both search engines and relevant websites are centralized and suffer from high load.

1) Napster: In order to decentralize servers, P2P networks are researched and developed. The first generation P2P networks still have a centralized file list like Napster which is a hybrid P2P network for music file sharing applications before its shutdown due to the piracy problem. It has a centralized server which requires the entire song lists of peers (personal

computers) to be uploaded for the indexing purpose. When we search for a song in a peer, it queries the central index server to find out all peers which have the song. Then we download the song from those relevant peers. However, this solution is too similar to the client-server network model. The main difference is that the files are no longer hosted in some Web servers. They are distributed over peers, but the searching process is still done in the central index server which is un-scalable because when the number of peers grows, then it requires more centralized resource to handle search requests. On the other hand, our proposed hybrid P2P network model is scalable due to the use of distributed index servers.

2) Gnutella: In order to fully decentralize servers for the load balancing, the second generation P2P networks are developed which emphasize on pure P2P networks. Gnutella is a pure P2P network for file sharing applications. It does not have any centralized server. When we search for files by a peer, it broadcasts the query to all its connecting peers. Then the peers propagate the query to their adjacent peers and this process continues until exceeding the Time-to-Live (TTL) value. Each peer looks up its locally shared collection and responds to its requester. This model is very similar to S2S Searching but Gnutella is designed for searching files in personal computers instead of websites. We extend this model for Web information retrieval. Gnutella supports those peers which frequently join and leave the network. S2S Searching also supports this although peers do not frequently join or leave because websites are usually persistence. Although Gnutella

solves the scalability problem of Napster, it has the query flooding problem which generates a lot of network traffic and wastes resources of all irrelevant peers [26]. On the other hand, S2S Searching solves this problem by applying the proposed query routing algorithm using *distributed registrars*.

3) Kazaa: In order to solve the query flooding problem of Gnutella, most P2P networks adopt a hybrid scheme like Kazaa. Kazaa is a file sharing application for a hybrid P2P network which uses FastTrack [17] as the P2P protocol. Although it does not have a centralized server, it is classified into hybrid P2P network because of the use of *super-nodes*. A super-node is a temporary index server for other peers. Any peer with a high computation power and fast network speed automatically becomes a super-node. When a peer joins the network, it finds an active super-node from its list of initial super-nodes. It also queries the active super-node for a list of other active super-nodes. Then it chooses a super-node as its index server and uploads a list of shared files. When it searches for a file, it directly queries the super-node, which also communicates with other super-nodes, for a list of relevant files with peer addresses. Then it directly downloads the files from those peers. This is very similar to our proposed hybrid P2P network model. The main difference is that the content summary instead of file index of the peer is uploaded to the *zone manager* (similar to the super-node) in our proposed hybrid P2P network model for content-based information retrieval.

4) BitTorrent: Besides searching the location of the target file, downloading the target file is also an important issue because it takes most time to do if the file is large. If the target file is only stored in a peer and we always download it, the peer becomes centralized server which is easy to be overload. Therefore, Kazaa may have this problem. BitTorrent is a P2P file distribution tool where files are broken into smaller fragments and distributed to peers in a pure P2P network. The fragments can be reassembled on a requesting machine in a random order. Thus, the parallel connections to all peers speed up the download process. To share a file, we create the corresponding torrent file which contains the file and *tracker* information. However, BitTorrent does not support indexing of torrent files. Thus, the torrent file is usually distributed to other users through websites. To download a file, we first open the torrent file and obtain the peers where its fragments reside through the tracker. Then we download the fragments in the available peers. When we finish downloading the entire file, our peer becomes an additional source for the file. BitTorrent uses the P2P concept for downloading files, whereas S2S Searching uses the P2P concept for searching Web documents.

5) Gnutella2: By combining Kazaa and BitTorrent, Gnutella2 is a reworking of Gnutella which gives up its pure P2P structure and uses a new hybrid P2P structure in order to solve the query flooding problem and speed up the download process. The old Gnutella protocol is discarded except for the connection handshake. Similar to Kazaa, Gnutella2 divides peers into

two types which are *leaves* and *hubs* (super-nodes). Leaves have one or two connections to hubs, while hubs have hundreds connections to leaves and other hubs. The searching mechanism is very similar to Kazaa so that we do not describe again. Besides the improvement of the query flooding, Gnutella2 allows the file to be downloaded from multiple sources which is like BitTorrent. In addition, Gnutella2 improves Gnutella by the extensible binary XML-like packet format so that the future network improvements and individual vendor features can be added easily. Other features include network data compression and advanced metadata system. The switching of Napster to Gnutella and Gnutella to Gnutella2 shows that there is no absolute advantage of pure P2P networks over hybrid P2P networks.

6) YouSearch: So far, the P2P networks that we introduce are not for the content-based information retrieval. YouSearch is a Web search engine for content-based full text search which is designed for searching contents in a hybrid P2P network of personal Web servers [41]. YouSearch adopts the method in Napster with a few improvements on the load of the index server. Similar to Napster, YouSearch depends on a centralized *registrar* which is a light-weight index server. However, the registrar stores only the content summary instead of the full index of each peer because storing the full index is impractical as the size may be very large. Each peer creates its own content summary by using Bloom filter [5] and pushes the summary to the registrar. When we search for contents by a peer, it queries the registrar to obtain a list of relevant peers with some false positives as the content

summary is only a hash table contained the index terms. Then it directly queries the relevant peers to remove those false positives. Although this direct connection model solves the query flooding problem, such model has two shortcomings which are high bandwidth consumption and poor semi-parallel search (see Section 1.1). On the other hand, our proposed hybrid P2P network model circumvents these shortcomings by introducing the query propagation model in hybrid P2P networks.

7) Discovir: Besides YouSearch, Discovir is a content-based image retrieval application for a pure P2P network. It is built on top of LimeWire [35] which uses the Gnutella protocol. Each peer is responsible for performing the feature extraction on those shared image files. These features include the color, texture, and shape. When we search for similar images by a peer, we can select a sample image and specify a feature extraction method. Then the peer transforms the image content to a feature vector based on what extraction method we choose. The feature vector and extraction method are propagated from peer to peer. Each peer tries to match the sample image with their local images by the feature vector. Moreover, Discovir solves the query flooding problem by using the *firework query model* which is based on the document clustering (see Section 2.2). Content-based image retrieval can also be plugged into S2S Searching. In that case, the query is the feature vector.

8) Freenet: The third generation peer-to-peer networks are those which have anonymity features built in. Freenet is a censorship-resistant data store

for a pure P2P network. Since it emphasizes on the censorship and anonymity, there is a tradeoff that it is slower and does not have integrated search functionality. All stored documents are segmented, encrypted, and distributed over anonymous peers so that it is difficult for a person to trace which peers are hosting a particular file. Freenet does not have the query flooding problem because it adopts a *key-based routing*. To publish a document, it is routed from peer to peer and stored in the anonymous destination peer. Those intermediate peers do not know which the initiating peer is. Since the same routing algorithm is used for all published documents, they form a cluster of similar documents which is similar to Discovir. To search a document with a given key, the peer uses the same key-based routing to locate the destination peers that may contain the target. Since the routing algorithm is heuristic in nature, we cannot guarantee that it always find the target. The main difference between Freenet and S2S searching is that peers of the former one stores other peers' contents, whereas peers of the later one stores local contents. Third generation networks, however, have not reached mass usage for file sharing because of the extreme overhead which anonymity features introduce, multiplying the bandwidth required to send a file with each intermediary used [41].

Table 1 and Table 2 show the comparison of the aforementioned pure and hybrid P2P networks respectively.

Table 1. Comparison of Pure P2P Networks

	Sharing	Peer	Query routing
Gnutella	Any file	Store local files	No
BitTorrent	Mainly video files	Store file fragments	Tracker
Discovir	Image files	Store local files	Firework query model
Freenet	Documents	Store documents published from other peers	Key-based
S2S Searching	HTML documents	Store a website	Distributed registrars

Table 2. Comparison of Hybrid P2P Networks

	Sharing	Peer	Hybrid model
Napster	Audio files	Store local files	Index server
Kazaa	Any file	Store local files and other peers' indices	Super-nodes
Gnutella2	Any file	Store local and other peers' indices and files	Leaves and hubs
YouSearch	HTML documents	Store local files	Registrar
GAroute	Any file	Store local files	Zone managers

2.2 Query Routing Strategies

In a P2P file sharing system, files are always indexed in order to improve the performance of searching. When we search for a file, we search in the file index instead of the files themselves. File indices can be stored by using the *centralized approach*, *localized approach*, or *distributed approach*. For the centralized approach, a centralized server stores the index of all peers. Examples are Napster and YouSearch. For the localized approach, each peer stores the index of its locally shared files. Examples are Gnutella and Discovir. For the distributed approach, the index of locally shared files is distributed to other peers. Freenet is an example. In this section, we introduce two different query routing strategies which are (1) document clustering and (2) Distributed Hash Table (DHT) for the localized approach and distributed approach respectively. Moreover, query routing is not necessary for the centralized approach because all indices are stored in the centralized server.

1) Document Clustering: One of the query routing strategies for the localized approach is the firework query model [36] in Discovir which is based on the document clustering. There are two types of links which are *random links* and *attractive links*. Random links are connections of peers which peers randomly make to other peers in the network. Attractive links are connections of peers which peers explicitly make to other peers when two peers share the similar data. The attractive links self-organize or cluster

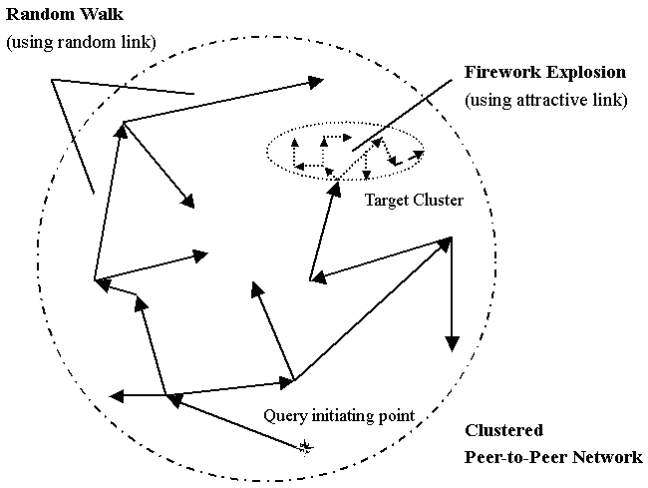

Figure 4. Firework Query Model

a group of peers with similar contents. In this model, a query first walks around the network from peer to peer by random links. Once it reaches the target cluster, it is broadcasted by peers using attractive links inside the cluster (see Figure 4). The TTL value of the query message does not decrease in attractive links. Unfortunately, this query routing strategy depends on a random walk which fails if it does not walk to the target cluster. In addition, Discovir uses the firework query model to route queries based on summarized indices which is like S2S Searching. In that case, the summarized index is the feature vector.

2) DHT: The distributed approach requires some query routing strategies so that we are able to look up the index location by given a key. DHT is one of the most popular techniques to distribute document keys or indices over peers [22]. CAN [45] models the key as the point on a

d-dimension Cartesian coordinate space, while each peer is responsible for the key-value pairs inside its specific region. pSearch [10] is a modification of CAN for content-based full text search applications which models the key as a latent semantic index [43]. Chord [28] models the key as an *m*-bit identifier and arranges the peers into a logical ring topology to determine which peer is responsible for storing which key-value pair. Pastry [2] and Tapestry [7] are similar which are based on the Plaxton mesh. Identifiers are assigned based on a hash on the IP address of each peer [15]. Pastry differs from Tapestry in the method by which it handles network locality and replication.

2.3 P2P Network Security

Besides the aforementioned query routing strategies, there is also some research about the security of P2P networks. Here, we briefly introduce two of them.

Cornelli proposes an approach [16] to P2P security where *servents* can keep track and share the information about the reputation of their peers with each other. The reputation sharing depends on a distributed polling algorithm such that resource requestors can assess the reliability of perspective providers before initiating the download. As the result, it complements the existing P2P protocols and keeps the current level of anonymity of requestors and providers.

Kamvar proposes the *EigenTrust* algorithm [43] for the reputation management in P2P networks which decreases the number of downloads of inauthentic files. The algorithm assigns each peer a unique global trust value which is based on the peer upload history. The peers use these global trust values to choose the peers from whom they download. As the result, the network effectively identifies malicious peers and isolates them from the network after.

3. S2S Searching

In the previous chapters, we introduce the related work of different P2P networks. In this chapter, we describe our proposed pure P2P network model and S2S Searching for the Web information retrieval. The chapter is organized as follows. In Section 3.1, we describe the system architecture of S2S search engines. The indexing and matching algorithms used in S2S search engines are shown in Section 3.2. We also describe the query routing algorithms and communication protocol in Section 3.3 and Section 3.4 respectively. Finally, we show the experimental results with some discussions in Section 3.5. For the perspectives of site owners and search engine users, please refer to Section 8.1 in the appendix.

3.1 System Architecture

There are two modules with several components in a S2S search engine. They are the *administration module* and *search module*. In this section, we describe their components and functions.

3.1.1 Administration Module

The administration module is accessed by site owners to administrate their S2S search engines. As shown in Figure 5, it has four components. They are the (1) *administrator*, (2) *local index manager*, (3) *ranking manager*, and (4) *network manager*.

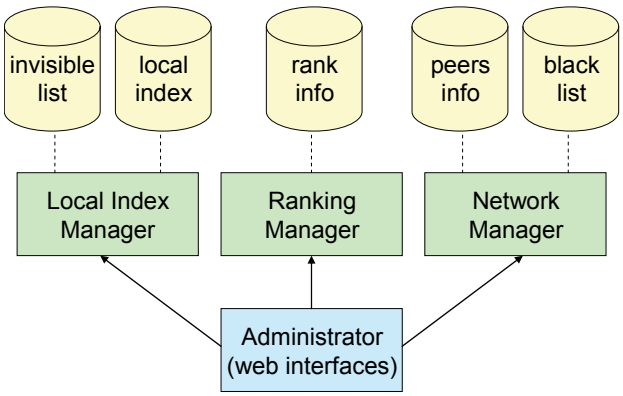

Figure 5. Administration Module and Its Components

1) Administrator: The administrator provides Web interfaces for site owners to manage their sites. Before managing their sites, they are required to login. After successful login, they can manage the local index, ranking parameters, and S2S network in the administration pages.

2) Local Index Manager: The local index manager is responsible for managing the local index. It is invoked by the *indexing CGI* when site owners want to refresh the local index and content summary after updating their Web contents. S2S search engines have no background job running because all programs are invoked by CGI as a request. Therefore, it is necessary to invoke the indexing CGI manually. However, it is still possible to do this automatically by using Hyper Text Markup Language (HTML) tag. For example, the tag

<meta http-equiv = "refresh" content = "60; url=indexing">

tells the browser to refresh CGI URL “indexing” every sixty seconds. Hence, site owners can keep their browsers open to ensure that the local index is

updating from time to time. For the indexing algorithm, we modify the existing Vector Space Model (VSM) [38] to be adaptive (see Section 3.2). Its performance is analyzed in Section 3.5. When the indexing CGI is called, it first traverses document directories and obtains all filenames. But it skips those CGI programs directories and other files which are listed in the invisible list. The invisible list determines whether a file is searchable. For each file, it compares the last indexing date with the last modification date. If the last modification date is more recent, then it recalculates the index. Finally, both index summary and content summary are built. For the index summary, it stores the meta-data of all documents such as filename, size, and indexing date. For the content summary, it is a fixed size hash table which stores the importance and confidence of the words. The updated content summary is then compared with the old content summary. If they are not the same, then the updated content summary is broadcasted to all adjacent sites for registrar maintenance (see Section 3.3).

3) Ranking Manager: The ranking manager is responsible for managing the priority value of each local document and the ranking parameters p and s (refer to (2)). The priority value is used to determine the importance of the corresponding document for advertising purpose. It is a real number which is normalized between zero and one. The higher priority value the document has, the higher position it is ranked. In addition, the priority value of other sites' documents are unchangeable and always set to 0.5. This is to prevent other site owners from always setting the priority

values of their documents to have maximum value. Only site owners can alter the ranking strategy in their own sites because the search engines belong to them.

4) Network Manager: The network manager is responsible for managing the S2S network. Site owners can add or remove some sites through this manager. To locate a site, we need to know the corresponding *starting URL* which is the root URL of CGI programs. This is similar to Gnutella that we need to know the Internet Protocol (IP) address of a peer in order to locate it. Before adding a site, site owners can ping it in the administration pages to obtain its information such as its response time and current state. When they try to add a site by giving the starting URL, the network manager also pings it by calling its *pinging CGI* (see Section 3.4). If the given URL is reachable, the network manager adds a record in the peers information file. The network manager also calls the *joining CGI* (see Section 3.4) of that site because joining is two-way. In addition, site owners can manage the black list which stores a list of banned IP addresses of other sites and the masks. It acts like a firewall. When some sites are in the black list, the current site does not accept any request from them. Using black list mechanism prevents those malicious sites to attack a site.

3.1.2 Search Module

The search module is the core of S2S Searching. It is accessed by search engine users to search the target information in both local site and

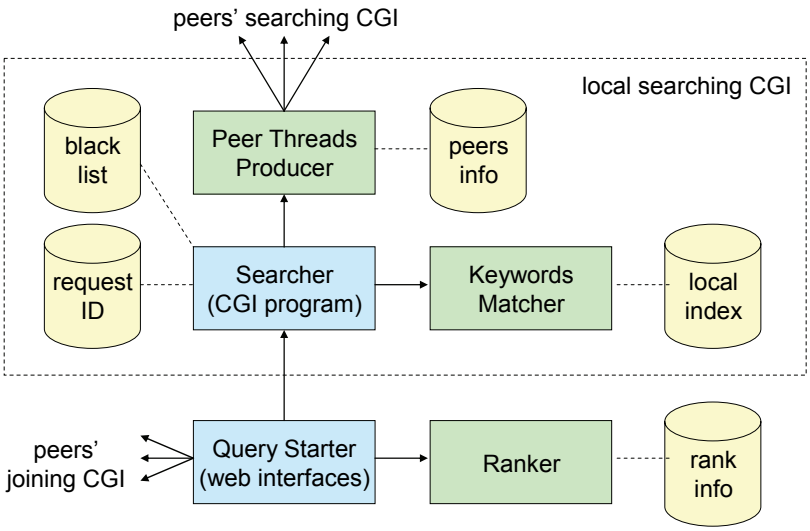

Figure 6. Search Module and Its Components

other sites which are in the same S2S network. As shown in Figure 6, it has five components. They are the (1) *query starter*, (2) *searcher*, (3) *peer threads producer*, (4) *keywords matcher*, and (5) *ranker*.

1) Query Starter: The query starter provides Web interfaces for search engine users to search the target information. When it receives a query request from the search form, it first generates a unique request ID. The ID is composed of the current time and a random number to ensure the uniqueness. Then it is passed to the local *searching CGI* (see Section 3.4) together with keywords and other parameters in the search form. The local searching CGI program searches the target information in the local site and also forwards the query request to adjacent sites. Two sites are adjacent if they know the starting URLs of each other. The local searching CGI program returns a list of results to the query starter together with the starting

URLs of the sites that contain any document which similarity is greater than the configurable quality threshold. The query starter joins those high quality sites by calling their joining CGIs. The more frequent we search in a site, the larger number of qualified sites we join. Finally, it forwards the results to the ranker and gets back ranked results. Then it outputs the ranked results in the HTML or Extensible Markup Language (XML) format which is specified in CGI parameters.

2) Searcher: The searcher is the entry point of the local searching CGI. When it receives a query request from the local query starter or other sites' searching CGIs, it first checks whether the requester is in the black list. If it is, then the query request is dropped. After passing the black list test, the searcher checks if the request ID exists in the file. If it exists, the current request is a repetitive request due to some loops in the S2S network. Therefore, the query request is dropped. If it does not exist, it passes the request ID test and the searcher adds the current request ID to the file. Request ID records are cleaned from time to time in order to save the storage space. The maximum number of request ID stored is configurable. Similar to Gnutella, the search scope of S2S Searching is controlled by using the Time-to-Live (TTL) mechanism. Therefore, the next step is to check whether the TTL value from CGI parameters is greater than zero. If it is, then the searcher asks the peer threads producer to route the query request to adjacent sites (see Section 3.3). At the same time, it asks the keywords matcher to search local contents by giving keywords. The peer

threads producer and keywords matcher work in parallel. After some time, both of them return results which include documents' information and starting URLs of the sites. The searcher then gathers these results and returns to the requester.

3) Peer Threads Producer: The peer threads producer is responsible for spawning threads to route a query request to adjacent sites. When it is called by the searcher, it spawns a requested number of threads. Each thread calls a unique site's searching CGI and waits for its return. The starting URLs of the sites are stored in the peers information file. A timeout mechanism is used to prevent some threads from waiting for too long time. The timeout time is configurable. Since the waiting time for other sites to return their results is dominant, the peer threads producer is always idle after sending the query request to all adjacent sites. Therefore, the keywords matcher gets full CPU resource to search local contents at that time. Other sites' searchers also work in parallel. Hence, the searching process is highly distributed and efficient. Finally, the peer threads producer finishes waiting all threads to join and returns gathered results to the searcher. The reason for waiting is that we need to gather all results for ranking so that they are synchronized.

4) Keywords Matcher: The keywords matcher is responsible for searching local contents by giving keywords. When it is called by the searcher, it first extracts the keywords and gets the index summary in the local index file. It tries to match the keywords with the index by using our matching algorithm based on the modified VSM. Once it matches, the

similarity is calculated. Usually, the keywords matcher utilizes the full CPU resource as the peer threads producer is idle for waiting other sites' searching CGIs to return. This makes the searching process very efficient. Finally, the keywords matcher returns the results to the searcher.

5) Ranker: The ranker is responsible for ranking search results based on priorities and similarities of documents which are real numbers between zero and one. Priorities are stored locally. Therefore, only local documents take effect of their priority values because site owners should have rights to advertise their documents in their own search engines. Other site owners are not allowed to rank their documents higher in other sites by setting higher priorities. This avoids cheating. If a document does not belong to a site, it is always set to the normal priority 0.5. The final ranking value *rank* is calculated by

$$rank = p \times priority + s \times sim \quad \text{where} \quad p + s = 1. \tag{2}$$

The ranking parameters p and s are real numbers between zero and one which are configurable by site owners according to their preferences. The range of the ranking value is also between zero and one. Different sites have their own ranking parameters which result in different ranking for the same document. In addition, it is possible that the ranking partially depends on the importance of the Web pages such as PageRank [34], HITS [29], and Affinity Rank [53]. Finally, the ranker sorts search results in the descending order by the ranking value. The ranked results are returned to the query starter.

3.2 Indexing and Matching

In this section, we introduce the background of indexing and matching. We also describe our indexing and matching algorithms based on the modified VSM.

3.2.1 Background of Indexing and Matching

In order to improve the matching speed, indexing documents is necessary. VSM is one of the popular indexing algorithms. It represents documents and queries by term vectors. The term weighting t_{ij} of the term (word) w_i in the document d_j is calculated by

$$t_{ij} = tf_{ij} \cdot idf_i \quad \text{where} \quad tf_{ij} = \frac{f_{ij}}{\max_l f_{lj}} \quad \text{and} \quad idf_i = \log \frac{N_d}{n_i}, \tag{3}$$

tf is known as the *term frequency*, *idf* is known as the *inverse document frequency*, f_{ij} is the raw frequency of w_i in d_j, N_d is the total number of documents, and n_i is the number of documents in which w_i appears. Similarly, the query term weighting t_{iq} is calculated by the above equations. The similarity *sim* between the document d_j and query q is calculated by cosine the angle between the document and query such that

$$sim = \cos(d_j, q) = \frac{d_j \cdot q}{|d_j| \cdot |q|} = \frac{\sum_{i=1}^{N_t} t_{ij} \cdot t_{iq}}{\sqrt{\sum_{i=1}^{N_t} t_{ij}^{\,2} \cdot \sum_{i=1}^{N_t} t_{iq}^{\,2}}}, \tag{4}$$

where N_t is the total number of index terms. However, when there are some documents added, deleted, or updated, the new *idf* values are completely

different. Therefore, the whole term weighting matrix needs to be recalculated and the whole index file needs to be overwritten. Since the N_t-by-N_d term weighting matrix is very large, writing the index file is very time-consuming which results in slow indexing time.

3.2.2 Indexing Algorithm

Due to the aforementioned shortcoming, we give up the *idf* value of each term. We only store the *tf* value of each term which results in faster indexing time. Therefore, our indexing algorithm treats every document independently. The index of each document is stored in an independent index file. When there is a document added, we calculate its index independently and store it in an independent index file. A new record is added to the index summary which contains the path and filename of the document, indexing date, and the corresponding index file ID. When there is a document deleted, we delete its index file as well and then delete the corresponding record in the index summary. When there is a document updated, we recalculate its index independently and update its corresponding index file. We also update its indexing date in the index summary. Other index files of other documents remain unchanged. Hence, we achieve a fast and adaptive update.

For the index calculation, we extract alpha-numerical words in a text document and filter out stop words like “a”, “an”, “the”, etc. Numbers are also filtered away. Then we convert all meaningful words to lower-cased

words. Let N be the number of different words in a particular document. We define the word set W as

$$W = \{w_i \mid 1 \le i \le N\}. \tag{5}$$

For each word w_i in W, its corresponding frequency $f(w_i)$ is calculated. We define the word importance $I(w_i)$ of the i^{th} word relative to the whole document as

$$I(w_i) = \frac{f(w_i)}{\max_{k=1}^{N} f(w_k)}. \tag{6}$$

Actually, the word importance is the same as the term frequency of VSM. Then we group the words by their first alphabets. There are at most 26 groups. N grouped vectors which are in the form of $(w_i, I(w_i))$ are stored in a local index file of a particular document. The word occurrence locations L of the inverted index can also be stored. However, it consumes much space and its overhead is shown in Section 3.5. In addition, we build an index table I which contains file offsets and lengths of each group. The index table is also stored in the same index file.

3.2.3 Matching Algorithm

After indexing each document, we can quickly match the keywords with each document by looking at the corresponding index file. Therefore, to obtain updated results, recalculate the index is necessary if some documents are updated.

For the keywords matching, by given the keywords, we extract

alpha-numerical keywords and filter out stop words like “a”, “an”, “the”, etc. Numbers are also filtered away. Then we convert all meaningful keywords to lower-cased keywords. Let n be the number of different keywords. We define the keyword set K as

$$K = \{k_i \mid 1 \le i \le n\}. \tag{7}$$

To perform keywords matching in a document, we may sequentially scan the corresponding index file and quit the scanning procedure at once if they are matched. The file scanning takes $O(N)$ time. However, it can be improved by the following method. Let m be the number of different first alphabets in K. We define the first alphabet set A as

$$A = \{a_i \mid 1 \le i \le m \le 26 \wedge m \le n\}. \tag{8}$$

To perform keywords matching in a document, for each alphabet a_i in A, we look up the index table I for the file offset and length. Then we jump to the corresponding position and perform the sequential scanning within the length. When we compare with the former keywords matching, the file scanning time is greatly reduced to a fraction of m / 26 approximately, assuming those 26 alphabets are evenly distributed. Hence, The file scanning takes $O(N \cdot m / 26)$ time. The improvement is shown in Section 3.5. We define the similarity value s_i of k_i as

$$s_i = \begin{cases} I(w_j) & \text{if } \exists_{w_j \in W} w_j = k_i \\ 0 & \text{otherwise} \end{cases}. \tag{9}$$

There are two types of keywords matching which are (1) *OR* and (2) *AND*.

1) OR Matching: For each local index file of a particular document, the

similarity value *sim* is calculated by

$$sim = \frac{1}{n}\sum_{i=1}^{n} s_i \ . \tag{10}$$

2) AND Matching: For each local index file of a particular document, the similarity value *sim* is calculated by

$$sim = \begin{cases} \frac{1}{n}\sum_{i=1}^{n} s_i & \text{if } \ \forall_{1\le i\le n} s_i > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{11}$$

3.3 Query Routing

In this section, we introduce the background of query routing. We describe our proposed query routing algorithm based on distributed registrars. The content summary generation and registrar maintenance are also presented.

3.3.1 Background of Query Routing

P2P networks like Gnutella have the query flooding property. All peers broadcast query requests to all their connecting peers. This model has two advantages. (1) The search is complete because all peers in the same P2P network within a specific TTL receive the query request. Therefore, all peers can search their local contents and return results to the query initiating peer. (2) The results obtained are global optimal because all peers return their optimal results to the query initiating peer. Therefore, the query initiating peer can select the most relevant results. However, this model

introduces the query flooding problem which is mentioned in Section 1.1. This problem arises from all peers flooding the query to their connecting peers as well as those irrelevant peers. Consequently, this problem generates a lot of network traffic and wastes resources of all irrelevant peers. The network traffic of a simple *n*-nary tree topology (see Figure 1) is analyzed in Section 1.1.

In order to reduce the exponential traffic cost (refer to (1)), routing query to relevant peers only is necessary. The query flooding problem can be solved by some existing query routing algorithms like CAN [45] and Chord [28] in pure P2P networks. They use Distributed Hash Table (DHT) to distribute indices to other peers. Since S2S Searching targets on those websites hosted by ISP Web servers which space is very limited and a site may need to store many indices of other sites if we apply the DHT model, this makes S2S Searching to be impractical. Content-based full text search applications like YouSearch [36] depends on a centralized *registrar* for storing content summaries of each peer. By querying the registrar, each peer obtains a list of relevant peers so that it directly connects to all relevant peers to obtain document lists. Thus, the query flooding problem does not exist. However, this model has two shortcomings. (1) It depends on a centralized registrar which is un-scalable. The centralized registrar also needs to store many data of all peers. (2) It has the registrar flooding problem because all peers query the centralized registrar from time to time. Moreover, when peers update their local contents, they also push their

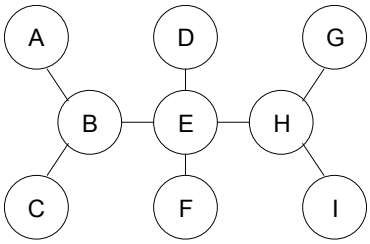

Figure 7. A S2S Network Topology

content summaries to the centralized registrar.

3.3.2 Distributed Registrars and Content Summary

In order to solve the query flooding problem in S2S Searching, we improve the method of YouSearch and propose our own query routing algorithm based on distributed registrars which is fast and scalable. The idea is to distribute registrars over sites. Each site manages its own registrar which contains content summaries of all adjacent sites. This model solves the scalability problem of YouSearch. Figure 7 shows an example. The registrar of site *B* stores the content summaries of sites *A*, *C*, and *E*. When we use site *B* to search with some keywords, site *B* first looks up its own registrar. If site *E* has the highest chance to match the keywords (higher relevance level). Then site *B* routes the query to site *E*. Site *E* receives this request and also follows the same strategy to route the query.

The registrar is a file which contains starting URLs as IDs of adjacent sites and their corresponding content summaries. YouSearch uses Bloom filter [5] to generate content summaries. However, we can only know

whether a site contains the given keywords instead of knowing its relevance level. Therefore, Bloom filter cannot meet our requirement. In S2S Searching, the content summary of a site is a fixed size hash table which stores the scores of different words of all documents in a site. We define the content summary S as

$$S = \{s_i \mid 0 \le s_i \le 1 \wedge 1 \le i \le m\}, \tag{12}$$

where m is the number of blocks in the hash table. In order to obtain an even distribution, m should be a prime number. A better choice for m is that

$$m = 2^p - 1, \tag{13}$$

where p is a prime number. In S2S search engines, p is 11. Hence, S contains 2,047 blocks which take 8,188 bytes, assuming a floating point number takes four bytes. If the maximum number of adjacent sites is 100, then the registrar takes less than 800KB to store content summaries. Given a lower-cased alpha-numerical word w, the hash function $H(w)$ of S is defined as

$$H(w) = \left[\sum_{i=1}^{l} 27^{i-1} (c_i - 96) \right] \pmod{m}, \tag{14}$$

where l is the length of w. In S2S search engines, the maximum value of l is fixed to 13 to prevent the integer overflow. If the i^{th} character is a number, then c_i is fixed to 96. Otherwise, c_i is the ASCII code of the alphabet. The quality of H is analyzed in Section 3.5. We use Horner Scheme [25] for a faster calculation and the hash function becomes

$$H(w) = h_l \pmod m \quad \text{where} \quad h_i = \begin{cases} 27h_{i-1} + (c_i - 96) & \text{if} \quad i > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{15}$$

To build a content summary, we traverse index files and get the information about the words and frequencies. Let N be the total number of different words in a site. We define the word set W as

$$W = \{w_i \mid 1 \le i \le N\}. \tag{16}$$

For each word w_i in W, its total frequency $f(w_i)$ of all documents is calculated. We define the word importance $I(w_i)$ of the i^{th} word relative to the whole site as

$$I(w_i) = \frac{f(w_i)}{\max_{k=1}^{N} f(w_k)}. \tag{17}$$

We also define the hash set HS_i of all words which have the same hash code i as

$$HS_i = \{w_j \mid w_j \in W \wedge H(w_j) = i\}. \tag{18}$$

Then the i^{th} element s_i of the content summary is calculated by

$$s_i = \begin{cases} \dfrac{CL_i}{|HS_i|} \sum_{w_j \in HS_i} I(w_j) & \text{if} \quad |HS_i| > 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{where} \quad CL_i = |HS_i|^{-1}. \tag{19}$$

Actually, s_i not only stores the average word importance for all w_j in HS_i, but also stores the confidence level CL_i which is inversely proportional to the number of collisions in s_i. When we compare s_i in different sites, the larger value of s_i has, the more important and confident the word w_j appears in that site. Finally, S contains the generated content summary of the adjacent site

and the registrar stores a list of adjacent sites and their corresponding *S*.

3.3.3 Query Routing Algorithm

When a site needs to route a query, it first looks up its own registrar. For each content summary *S* in the registrar, it calculates the score (relevance level) of an adjacent site with the given lower-cased alpha-numerical keywords. Let *n* be the number of different keywords. We define the keyword set *K* as

$$K = \{k_i \mid 1 \le i \le n\}. \tag{20}$$

There are two types of keywords matching which are (1) *OR* and (2) *AND*.

1) OR Matching: The score of a site, which is normalized between zero and one, is defined as

$$score = \frac{1}{n}\sum_{i=1}^{n} s_{H(k_i)} . \tag{21}$$

2) AND Matching: The score of a site, which is normalized between zero and one, is defined as

$$score = \begin{cases} \frac{1}{n}\sum_{i=1}^{n} s_{H(k_i)} & \text{if } \forall_{1\le i\le n} s_{H(k_i)} \neq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{22}$$

After calculating scores of N_a adjacent sites, a list of relevant sites with some false positives is obtained. It routes the query to N_a' sites, which have the highest scores, such that

$$N_a' = \lceil f \cdot N_a \rceil, \tag{23}$$

where the *traffic reduction factor f* is a real number between zero and one

for reducing the network traffic. For example, if f is 0.2 and the current site has 10 adjacent sites, then the query is routed to the two sites which have the highest scores. If there are some sites which have the same highest scores, then we randomly pick them.

Our proposed query routing algorithm greatly solves the query flooding problem. However, it has two shortcomings. (1) The search is incomplete because not all sites in the same S2S network within a specific TTL receive the query request. Therefore, not all sites search their local contents and return their results to the query initiating site. (2) The results obtained are local optimal because our proposed algorithm performs a greedy search. Not all relevant sites search their local contents and return their results to the query initiating site. Therefore, the query initiating site cannot obtain global optimal results. Since this is a tradeoff, we make a balance between the query routing and query flooding. We enable *infrequent query flooding* in a site with a small probability p. When it receives a query request, it has the probabilities p and 1-p to use the infrequent query flooding and query routing algorithm respectively. With infrequent query flooding, our proposed algorithm improves the search to be semi-complete and semi-global optimal. We analyze the previous simple n-nary tree topology (see Figure 1) for the new query routing model. The depth d of the tree is the TTL value of the query in the query initiating peer. Assume the infrequent query flooding probability is p and traffic reduction factor is f. The expected fan-out degree $\overline{n}$ of each site is calculated by

Table 3. Comparison of Query Routing Algorithms

	Query flooding	Query routing	Infrequent flooding
Search	Complete	Incomplete	Semi-complete
Results	Global optimal	Local optimal	Semi-global optimal
Traffic	Expensive: $\sum_{i=1}^{d} n^i$	Very cheap: $\sum_{i=1}^{d} (f \cdot n)^i$	Cheap: $\sum_{i=1}^{d} \{n[p + (1-p)f]\}^i$

$$\overline{n} = n[p + (1-p)f]. \tag{24}$$

Let the traffic cost for sending a query between two peers be one unit. The total traffic cost T_{route} for searching in the whole network is

$$T_{route} = \sum_{i=1}^{d} \overline{n}^i = \frac{\overline{n}(\overline{n}^d - 1)}{\overline{n} - 1} \text{ units.} \tag{25}$$

Now, we compare it with the query flooding model. If every node has 10 degrees of fan-out (n = 10) and the TTL value is also 10 (d = 10), then the total traffic cost for the query flooding model T_{flood} (refer to (1)) is 11,111,111,110 units. On the other hand, if every node has a query flooding probability 0.1 (p = 0.1) and traffic reduction factor 0.2 (f = 0.2), then the expected fan-out degree $\overline{n}$ (refer to (24)) is 2.8 and the total traffic cost for the query routing model T_{route} (refer to (25)) is only 46,073 units. When the TTL value increases, the total traffic cost significantly decreases comparing with the query flooding model. Hence, our proposed algorithm solves the query flooding problem well. Table 3 shows the comparison of different query routing algorithms.

3.3.4 Registrar Maintenance

In order to maintain the most updated content summaries, the maintenance of registrars is necessary. Recall that every site stores its adjacent sites' content summaries in its own registrar. It is necessary for adjacent sites to send their updated content summaries if their contents are updated. Therefore, when a site updates its local contents, it recalculates its local index and also content summary. If the updated content summary is different from the old one, then it broadcasts its updated content summary to all adjacent sites by calling their *updating CGIs* (see Section 3.4). This model does not introduce the registrar flooding problem of YouSearch because update is usually infrequent and it only disturbs adjacent sites in one level. Figure 7 shows an example. When site *B* is updated, it broadcasts its content summary to sites *A*, *C*, and *E*. Other non-adjacent sites are unaffected as their registrars do not store the content summary of site *B*. In addition, broadcasting content summary in one level is rather cheap. Assume it takes four bytes to store a floating point number. Then it takes $4 \cdot m \cdot N_a$ bytes to broadcast, where m is the number of blocks in a content summary and N_a is the number of adjacent sites. In S2S search engines, m is 2,047. If the maximum number of adjacent sites is 100, then it takes less than 800KB to broadcast and each site receives 8KB data.

3.4 Communication Protocol

S2S Searching targets on those websites which is hosted by ISP Web servers. Therefore, we assume that site owners have very limited privilege to administrate their sites. For example, they are only allowed to transfer their files between their local computers and Web servers through File Transfer Protocol (FTP). It is a challenge to make S2S search engines plug into most sites easily and do not require any system administrator to install some special software. Taking these into consideration, CGI seems to be the best choice for the communication protocol which has four advantages. (1) Most Web servers support CGI programming languages such as Java Servlet. (2) Site owners can install CGI programs by themselves. The usual step is to copy CGI programs to the CGI directory. Thus, they do not need to ask any system administrator to install. (3) CGI is on top of Hyper Text Transfer Protocol (HTTP) where firewalls usually allow these packets to pass through. Thus, they do not need to ask any system administrator to open other ports in firewalls. (4) CGI programs are located by URLs which are location transparent [19]. This is very important because if the IP address of the Web server is changed due to a server migration, then the CGI URLs are still unchanged. There are six CGIs for the communication protocol. They are the starting CGI, searching CGI, pinging CGI, joining CGI, leaving CGI, and updating CGI which are described in this section. Table 4 shows their summary.

Table 4. Summary of Six CGIs

	Name	Parameter and Type	Return
Starting CGI	start	key (string), type (“or” / “and”), scope (“global” / “local”), ttl (integer), style (“html” / “xml”)	HTML / XML code
Searching CGI	search	id (string), key (string), type (“or” / “and”), ttl (integer), threshold (float)	Documents’ info, starting URL
Pinging CGI	ping	option (“status” / “peers” / “echo”), value (string)	Depend on option
Joining CGI	join	url (string)	Successfulness
Leaving CGI	leave	url (string)	Successfulness
Updating CGI	update	url (string), summary (binary)	Successfulness

3.4.1 Starting CGI

The starting CGI is called by search forms for starting search requests. After it is called, it calls the local searching CGI for obtaining search results. The CGI name is *start*. There are five parameters. (1) The parameter *key* (string type) specifies the keywords to be searched. (2) The parameter *type*

(string type, either “or” or “and”) specifies the keywords matching type. (3) The parameter *scope* (string type, either “global” or “local”) specifies the searching scope of the S2S network. (4) The parameter *ttl* (integer type) specifies the maximum level of sites (excluding the local site) that the query request passes through. (5) The parameter *style* (string type, either “html” or “xml”) specifies whether the style of the search results is either in the HTML or XML format. It returns the HTML or XML code which contains ranked results.

3.4.2 Searching CGI

The searching CGI is called by the starting CGI or other sites for searching the target information. After it is called, it also calls other searching CGIs of adjacent sites to route the query request. The CGI name is *search*. There are five parameters. (1) The parameter *id* (string type) specifies the unique ID of the query request. (2) The parameter *key* (string type) specifies the keywords to be searched. (3) The parameter *type* (string type, either “or” or “and”) specifies the keywords matching type. (4) The parameter *ttl* (integer type) specifies the current TTL value of the query request. (5) The parameter *threshold* (float type) specifies the quality threshold for site joining. It returns a list of results which includes documents’ filenames, URLs, dates, sizes, and similarities. If there is any document which similarity is greater than the quality threshold, then the starting URL of the current site is also returned.

3.4.3 Pinging CGI

The pinging CGI is called by the joining CGI or other sites for querying the information about the current site such as the response time and number of sites joined. The CGI name is *ping*. There is a parameter *option* (string type) which specifies the query option. It returns the information which depends on the query option. There are three options which are “status”, “peers”, and “echo”. (1) The option “status” is to check if the current site is alive which returns the string “ok”. (2) The option “peers” is to query the number of sites joined. (3) The option “echo” takes one parameter *value* (string type) and then echoes the input string. It is used to calculate the response time of the current site.

3.4.4 Joining CGI

The joining CGI is called by other sites for requesting the current site to join another site which starting URL is specified in the parameter. After it is called, it calls the pinging CGI of the target site to check whether it is valid. The CGI name is *join*. There is a parameter *url* (string type) which specifies the target site’s starting URL. It returns the successfulness of joining.

3.4.5 Leaving CGI

The leaving CGI is called by other sites for requesting the current site

to leave another site which starting URL is specified in the parameter. The CGI name is *leave*. There is a parameter *url* (string type) which specifies the target site's starting URL. It returns the successfulness of leaving.

3.4.6 Updating CGI

The updating CGI is called by other sites for updating another site's content summary in the current site's registrar which starting URL is specified in the parameter. The CGI name is *update*. There are two parameters. (1) The parameter *url* (string type) specifies the target site's starting URL. (2) The parameter *summary* (binary data in string type) specifies the content summary of the target site. It returns successfulness of updating.

3.5 Experiments and Discussions

In this section, we summarize the experimental results with some discussions. We measure the (1) performance of indexing, (2) performance of matching, (3) performance of S2S Searching, and (4) quality of the content summary. All experiments are performed with the same computer configuration (see Table 5). The computer has enough physical memory so that it does not require any memory swapping. It also has a fast network speed to simulate those Web servers which are placed in data centers. However, it has a slow file Input/Output (I/O) speed as they only use

Table 5. Computer Configuration of S2S Searching Experiments

Item	Setting
CPU	Sun Blade 1000 at 900MHz
Memory	2GB RAM
Network	100Mbps
Disk	NFS
OS	Sun Solaris 8
Java VM	Java 2 Standard Edition 1.4.2_05
Web server	Jakarta Tomcat 3.3.2

Network File System (NFS) instead of local raid-disks. Its overall performance is less than a recent dedicated Web server.

3.5.1 Performance of Indexing

This experiment is to measure the performance of indexing in both size and time. In Section 3.2, we mention that the word occurrence locations *L* of the inverted index can also be stored in index files. To measure the size and time differences between the presence and absence of *L*, we randomly select 31 HTML posters in the Twelfth International World Wide Web Conference [47]. The total document size of the text data to be indexed is 441KB. We incrementally add the document size from 11KB (first poster) to 441KB (last poster) and measure the corresponding (1) indexing size and (2) indexing time.

1) Indexing Size: Figure 8 shows the relationship between the original document size and indexing size with and without *L*. The average

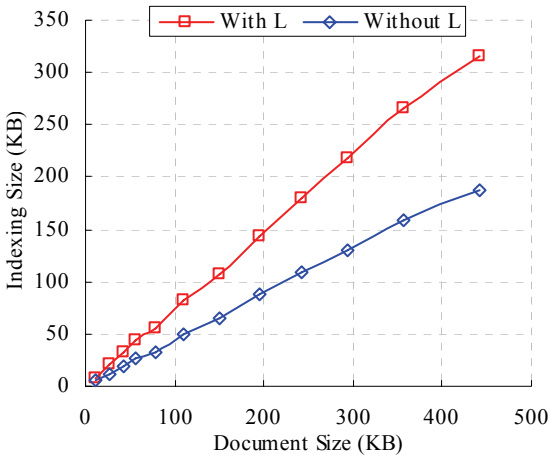

Figure 8. Indexing Size of S2S Search Engine

index-to-document ratio (slope of the line) with and without *L* are about 0.74 and 0.45 respectively. Both presence and absence of *L* take linear size. However, storing *L* in index files consumes much space as the overhead. Without *L*, the size is reduced by about 40 percent. The usage of *L* is that when we display the search result of a document, we may also display its part of text that contains the given keywords. Using *L* reduces the time for locating the target text but increases the space consumption. Due to the space limitation of websites hosted by ISP Web servers, we simply discard *L* which is like YouSearch (see Figure 2 of [36]).

2) Indexing Time: Figure 9 shows the relationship between the original document size and indexing time with and without *L*. Recall that our indexing algorithm is adaptive so that when there is a document added or updated, we only recalculate the index of the corresponding document. Other index files are unaffected. However, the time measured in this

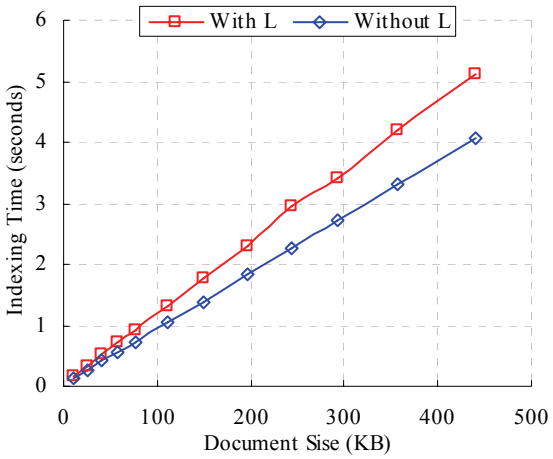

Figure 9. Indexing Time of S2S Search Engine

experiment is the worst case that all documents are updated so that they need to be re-indexed. Both presence and absence of *L* take linear time. Actually, the bottleneck is in the file I/O as we store the document and index files in NFS which is much slower than local raid-disks. Therefore, it requires some time to read the document file and write the index files. Without *L*, the time is reduced by about 20 percent because we do not need to write *L* to the disk.

3.5.2 Performance of Matching

This experiment is to measure the performance of matching in time. In Section 3.2, we mention that we build an index table *I* which contains file offsets and lengths of each first alphabet group for a faster matching. To measure the time differences between the presence and absence of *I*, we randomly select 31 HTML posters in the Twelfth International World Wide

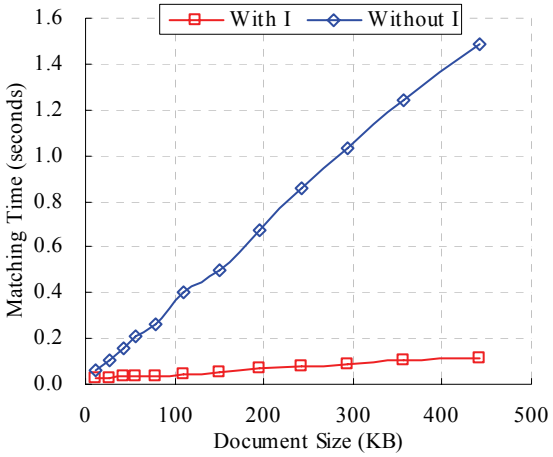

Figure 10. Matching Time of S2S Search Engine

Web Conference. The total document size of the text data to be matched is 441KB. We incrementally add the document size from 11KB (first poster) to 441KB (last poster) and measure the corresponding matching time with some random keywords.

Figure 10 shows the relationship between the original document size and matching time with and without *I*. Both presence and absence of *I* take linear time. Actually, the bottleneck is in the file I/O as we store index files in NFS which is much slower than local raid-disks. Therefore, it requires some time to read the index files. With *I*, the time is reduced by about 84 percent which is very significant. The reason of the great improvement is that without *I*, we sequentially scan the index files which take a lot of time. However, with *I*, we look up *I* for the file offset and length of the first alphabet of keywords. Then we jump to the corresponding position and perform the sequential scanning within the length. Theoretically, the file

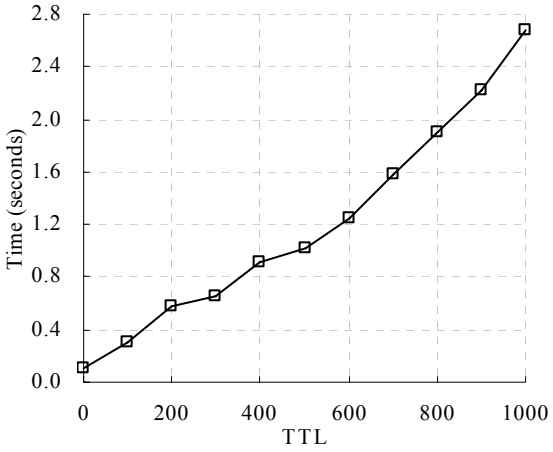

Figure 11. S2S Searching Time

scanning time is greatly reduced to a fraction of m / 26 approximately where *m* is the total number of different first alphabets of the keywords, assuming those 26 alphabets are evenly distributed.

3.5.3 Performance of S2S Searching

This experiment is to measure the performance of S2S Searching in both (1) searching time and (2) searching time dependency by simulation.

1) Searching Time: We measure the performance of S2S Searching in searching time by simulation. The total number of virtual sites to be searched is 10,000 which are randomly connected and evenly distributed in two computers. Each virtual site contains 400KB documents and the matching time is about 0.1 second according to Figure 10. We search in these 10,000 virtual sites and measure the searching time with different TTL values. During searching, query packets are propagated between two

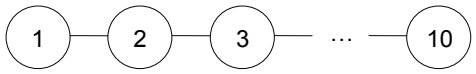

Figure 12. Linear Structure of 10 Sites

computers through the network to simulate the real query propagation scenario. Figure 11 shows the relationship between the TTL value and searching time. The time measured includes the matching time and query propagation time of all virtual sites. However, the result propagation time, which is proportional to the amount of search results, is not included. From the experimental results, we demonstrate that S2S Searching is efficient in some large scaled S2S networks. The efficiency is due to the fact that the searching process is highly distributed and is done in parallel. When a site receives a query, it first concurrently forwards the query to its adjacent sites and then performs the matching during the wait of returned results. Therefore, the query can reach all sites quickly. On the other hand, if the site first performs the matching and then concurrently forwards the query to its adjacent sites, the searching time increases a lot as the query reaches all sites slowly.

2) Searching Time Dependence: We measure the performance of S2S Searching in searching time dependence by simulation. There are 10 sites which are connected by a linear structure (see Figure 12) which adjacency matrix A is

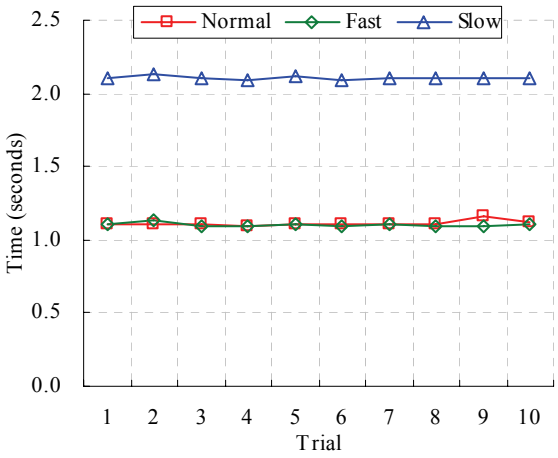

Figure 13. S2S Searching Time Dependence

$$A_{ij} = \begin{cases} 1 & \text{if } |i-j| = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{26}$$

Figure 13 shows the searching time in 10 trials with different matching time. The results are obtained by the following procedure. First, we set the matching time of the 10 sites to one second and measure the searching time which is indicated by the curve labeled “normal”. Second, we change the matching time of nine sites to half second and measure the searching time which is indicated by the curve labeled “fast”. However, the searching time does not improve. Then we change the matching time of the nine sites back to one second. Finally, we change the matching time of one site to two seconds and measure the searching time which is indicated by the curve labeled “slow”. The searching time increases to about two seconds. From the experimental results, we demonstrate that the searching time depends on the slowest site which involves in searching. The reason is that the query is

first propagated to adjacent sites. Then each site performs the matching. Those fast sites which finish their matching always wait for those slow sites to return their results. If there is any slow site that joins the S2S network, the searching time may be bad. Therefore, S2S Searching solves this problem by applying the timeout mechanism. During searching, slow sites that are timeout are skipped.

3.5.4 Quality of Content Summary

This experiment is to measure the quality of the content summary in our proposed query routing algorithm. In Section 3.3, we mention that the content summary is a fixed size (2,047 blocks) hash table which stores the scores of different words of all documents in a site. Therefore, the quality of the content summary can be measured in terms of the number of collisions. We randomly select some HTML posters in the Twelfth International World Wide Web Conference as the documents in a site. Then we build the corresponding content summary and measure the number of collisions in each block of the hash table. We only present one of our most representative results.

Figure 14 shows the pie chart of the usage and the number of collisions C of the content summary hash table. The content summary contains 31 documents. After removing the stop words, the total number of different words to be hashed is 5,423. There are 2,047 blocks in the hash table. For an even distribution, each block should ideally have about 1.65 collisions. First,

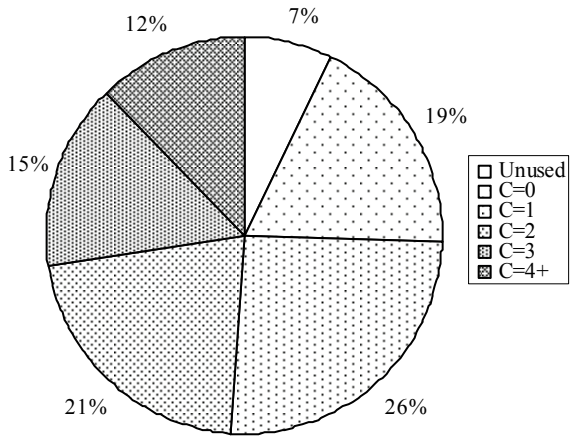

Figure 14. Quality of Content Summary Hash Table

we calculate the usage of the hash table which is about 93%. This result shows that the hash table is quite efficient. Second, we calculate the average number of collisions of the hash table which is about 1.85 (near to the ideal case 1.65) with less standard deviation 1.53. This result shows that the hash table is quite evenly distributed and confident. Therefore, the content summary is quite high in the quality. Actually, the quality depends on l and m in the hash function (refer to (14) and (15)). If they are larger, then the quality is higher. The larger value of l has, the more characters are involved in the calculation, but a bigger integer type is needed. The larger value of m has, the more blocks are available in the hash table, but a bigger memory size is needed.

4. GAroute

GA is a general search algorithm that imitates the evolution process in the nature [31]. Recently, GAs have been widely used to solve different network and graph problems. Koo [45] proposes a GA to solve the neighbor-selection problem in the BitTorrent network which enhances the decision process performed at the tracker for transfer coordination. Ahn [11] proposes a GA to solve the shortest path routing problem in a physical network. One of the popular network and graph problems, which can be efficiently solved by GA, is the query routing problem in P2P networks.

In the previous chapter, we describe our proposed pure P2P network model and S2S Searching for the Web information retrieval. In this chapter, we describe our proposed hybrid P2P network model and GAroute for the content-based information retrieval based on GA to solve the query routing problem. The chapter is organized as follows. In Section 4.1, we introduce the background of our proposed hybrid P2P network model. In Section 4.2, we describe our proposed GAroute with problem modeling and detail explanation of the GA operators. Finally, we show the experimental results with some discussions in Section 4.3. For the GAroute library, please refer to Section 8.2 in the appendix.

4.1 Proposed Hybrid P2P Network Model

In this section, we introduce the background of hybrid P2P networks.

We also briefly describe our proposed hybrid P2P network model based on zones and zone managers.

4.1.1 Background of Hybrid P2P Networks

YouSearch is a content-based full text search application which uses the registrar as the centralized server to store content summaries of each peer. By querying the registrar, we obtain a list of relevant peers so that we query the relevant peers to obtain document lists. We cannot obtain the document lists by only querying the registrar because it only stores the content summaries rather than the indices of each text document of each peer. Otherwise, it is too space consuming and impractical. The use of the registrar and DCM makes the application un-scalable when the network scale increases.

Kazaa is a file sharing application which improves the scalability by using super-nodes. If a peer is fast in both computation and network speed, it becomes a super-node. Each super-node is like a small registrar which stores indices of each file of a few peers, and there are some communications between super-nodes. Therefore, by querying a super-node, we obtain relevant file lists in the whole network. However, Kazaa is not designed for content-based information retrieval. It is also space consuming if we store indices of multimedia objects in a super-node.

In order to solve the scalability problem of content-based information retrieval and improve the overhead of the DCM, we refer to Kazaa and

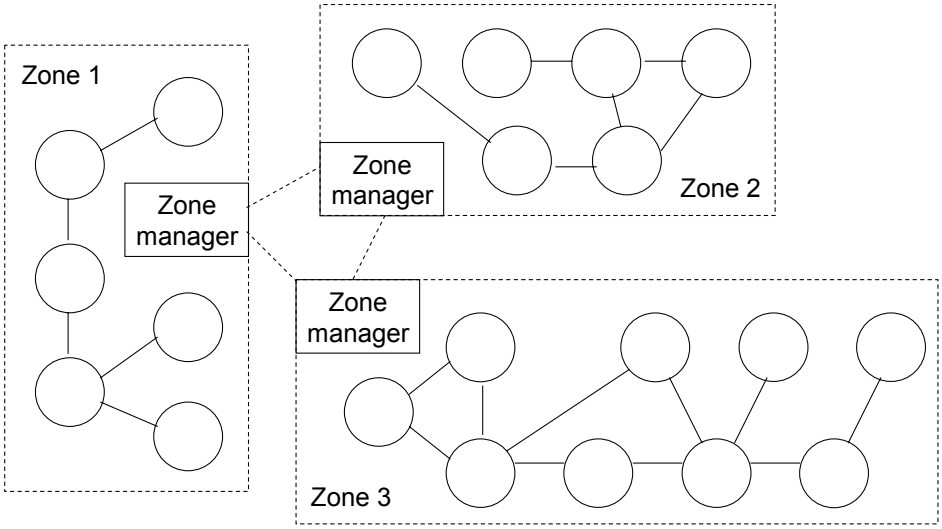

Figure 15. Structured P2P Network with Three Zones

YouSearch, and propose a hybrid P2P network model based on QPM. The whole network is divided by several zones. A *zone* is a set of peers logically linking together by a structure which is based on their inter-connection speed. Hence, it prevents the topology mismatching problem [54]. Each zone is managed by a *zone manager* which is a dedicated server like a big super-node in Kazaa and distributed registrar in YouSearch (see Figure 15). The size (number of peers) of a zone depends on the computation power of its corresponding zone manager. We may interpret a zone as an Internet Service Provider (ISP). The peers that belong to the same ISP have fast inter-connection speed so that they join the same zone. We describe the roles of zone manages in the next subsection.

4.1.2 Roles of Zone Managers

Similar to the registrar, the zone manager stores content summaries of each peer within its zone. When a peer updates its local content, it recalculates its content summary and pushes its summary to its zone managers. YouSearch and our proposed S2S Searching generate content summaries for content-based full text search applications by using Bloom filter [5] and simple hash function (see Section 3.3) respectively. We use the method in our proposed S2S Searching to generate content summaries because it efficiently calculates the relevance level of each peer by given a query. In addition to managing the content summaries, the zone manager manages the current P2P network topology within its zone. We outline the three actions of each peer that involve zone managers.

1) Joining Network: We assume that each peer knows some initial zone managers before it joins the network and there are some communications between zone managers. When a new peer joins the network, it queries an arbitrary initial zone manager to obtain other zone managers in different zones. Then it chooses a zone with the fastest zone manager and smallest zone size to join. When it joins the zone, it queries the zone managers to obtain a set of peers within the zone. Then it chooses one or two peers with the fastest inter-connection speed to link. The link between two peers is logical so that logically linking two peers means physically updating the current P2P network topology stored in the zone manager. It also pushes its content summary to the zone manager. In addition, when a peer detects that

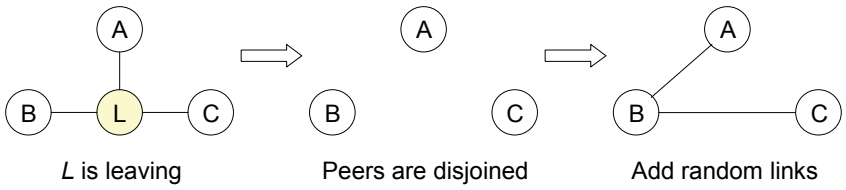

Figure 16. Problem of Peers Disjointing Due to a Leaving Peer

its zone manager is down, it joins another zone by the same joining procedure.

2) Leaving Network: When an existing peer leaves the network, it informs its zone manager to remove its content summary and update the current P2P network topology within its zone. If other existing peers are disjointed due to the leaving peer, then the zone manager adds random links to link the neighbors of the leaving peer together. We assume that if two peers are close (fast inter-connection speed) to each other and linked together, then their neighbors are also close. Therefore, randomly linking the neighbors of the leaving peer retains the structure of the zone as close peers are still linked together. Figure 16 shows an example of the leaving scenario. When *L* left, we check whether there exists any path between *AB*, *BC*, and *AC*. If not, then *A*, *B*, and *C* are disjointed. We add random links *AB* and *BC* to link them together. Since *LA*, *LB*, and *LC* are close, *AB* and *BC* are also close according to our assumption. In addition, when a zone manager detects that a peer is down, it treats that peer as a leaving peer and uses the same leaving procedure.

3) Querying: When a query initiating peer initiates a query, it queries

its zone manager to obtain query routing paths in its zone. The zone manager also queries other zone managers (similar to Kazaa) to obtain query routing paths in other zones. In this case, another zone manager becomes the query initiating peer which links all peers in its zone like a hub and finds query routing paths in its zone. Finally, all query routing paths from different zones are returned to the query initiating peer. Then it propagates the query to all relevant peers through these paths to obtain document lists. There are two ways that the results (document lists) can be sent. For the first way, the results are propagated from peer to peer back to the query initiating peer through the inverted query routing path which is like QPM. However, this increases the whole network traffic as each peer receives, adds and forwards the results to the next peer in the inverted query routing path. For the second way, the results are directly sent to the query initiating peers which is like DCM. This reduces the whole network traffic and does not lead to the two aforementioned shortcomings of DCM. Since the same results are received at the query initiating peers by both ways, the query initiating peer does not have lower bandwidth consumption by the first way. Since the results usually do not arrive at the query initiating peer at the same time by both ways, the query initiating peer does not have better parallel search by the first way. In addition, the query initiating peer can obtain partial results by the second way which has a faster response, whereas the query initiating peer needs to wait for the whole results by the first way which has a slower response. Therefore, the second way is better

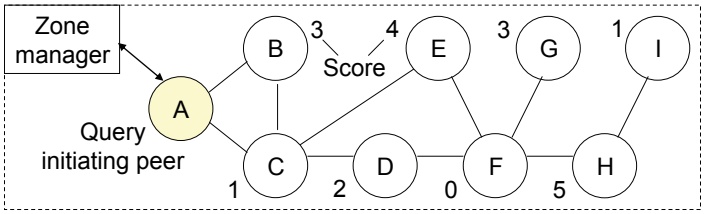

Figure 17. Structured P2P Network in a Zone with Scores

than the first way and we adopt the second way in our proposed hybrid P2P network model.

To find optimal query routing paths within a zone, we propose a novel GA called GAroute. We focus on our proposed GAroute in the next section.

4.2 Proposed GAroute

By giving the current P2P network topology and relevance level of each peer (see Figure 17), GAroute returns a list of query routing paths that cover as many relevant peers as possible. We model the query routing problem as a directed graph problem G such that

$$G = (V, E), \tag{27}$$

where V is a set of vertices representing peers and E is a set of edges representing the connectivity between peers. We use an adjacency matrix A to represent the current P2P network topology which is stored in the zone manager. Two peers are adjacent if they are linked together. A non-zero value in A_{ij} represents that there is a link from peer i to peer j. For the relevance level of a peer, different applications may have different

definitions. For example, file sharing applications may define the relevance level of a peer as the number of files that the query matches in that peer. Content-based information retrieval applications may treat the relevance level of a peer as the average similarity between the query and all contents in that peer. In our application, we use the latter definition. We model the relevance level as a *score* and we use a score vector S to store the scores of all peers. We denote S_i as the score of peer i which is query-sensitive and is calculated by the zone manager. We also denote the query initiating peer as x_1 and the maximum number of paths to be returned as n. Then we pass A, S, x_1, and n to GAroute as parameters which returns a list of query routing paths P such that

$$P = \left(p_i \mid 1 \le i \le n\right). \tag{28}$$

where a query routing path p is

$$p = \left\langle x_i \,\middle|\, \forall_{i,j} (A_{x_i x_{i+1}} \neq 0) \wedge (i \neq j \Leftrightarrow x_i \neq x_j) \right\rangle. \tag{29}$$

The if-and-only-if statement in the above equation constrains the path to be simple (loop-free). Loops in a path are meaningless for query routing because some peers receive duplicated queries and return duplicated results.

Given a list of query routing paths P, we define the *information gain* H_p of a path p in P as the sum of the scores of those unvisited peers such that

$$H_p = \sum_{i=2}^{|p|} (S_{x_i} - \rho_{x_i}) \quad \text{where} \quad \rho_x = \begin{cases} S_x & \text{if} \quad x \in V' \\ 0 & \text{otherwise} \end{cases}, \tag{30}$$

ρ_x is known as the *penalty* of the peer x and V' is a set of the current visited peers. Different applications may have different penalty equations. We define the penalty of a peer to be the same as its score because those visited peers give us duplicated results for a duplicated query so their scores are zero. For example, we have to route two paths $p_1 = <A, C, E, F, G>$ and $p_2 = <A, C, E, F, H, I>$ where A is the query initiating peer (see Figure 17). After routing a query by p_1, the information gain is eight units. However, after routing a query by p_2, the information gain is six units instead of 14 units because the peers C, E, and F are visited by the first path. Therefore, their scores are zero in the second path.

Our problem is to find at most n query routing paths P from a query initiating peer to any destination peer which maximize the total information gain where

$$P = GAroute(A, S, x_1, n) = \arg\max_{(p_i | 1 \le i \le n)} \sum_{i=1}^{n} H_{p_i} \ . \tag{31}$$

We model this as a *Longest Path Problem* which is NP-complete. Although $O(|V|^2)$ time Dijkstra's algorithm can be modified to find the longest path, it only works for directed acyclic graphs [37].

Since it is unlikely to have a polynomial time algorithm for finding the longest path in cyclic graphs, related approximation algorithms are proposed to solve this problem in polynomial time with a path length bound [48]. Both Monien's algorithm [6] and Bodlaender's algorithm [23] find a long path with a length bound $\Omega(\log L / \log \log L)$ where L is the length of the

optimal solution. Alon's Color-coding algorithm [39] finds a better long path with a length bound $\Omega(\log L)$. Bjorklund's algorithm [1] improves the length bound by dividing a graph into connected components such that the length bound is $\Omega((\log L / \log \log L)^2)$. For bounded degree graphs, it further improves the length bound to $\Omega(\log^2 L / \log \log L)$. Moreover, Karger's algorithm [14] finds a long path with a length bound $\Omega(L^{1/2} / \log L)$ in sparse Hamiltonian graphs.

However, all aforementioned algorithms cannot be easily modified for our specific longest path query routing problem because some of them only work in un-weighted graphs and some of them have a fixed destination. Therefore, we propose GAroute to solve this specific problem which obtains high quality approximate solutions in polynomial time by using GA. Our proposed GA refers to Ahn's GA [11] which finds the shortest path from a given source node to a given destination node. In the following subsections, we describe our proposed GA for finding long paths and optimization technique which include the *genetic representation*, *population initialization*, *mutation*, *crossover*, *fission*, *creation*, *selection*, *stopping criteria*, and *optimization*. We also compare our proposed GA with Ahn's GA. Figure 18 shows our proposed GA flow chart. Besides conventional GA operators, we propose two extra GA operators fission and creation to improve the quality of solution.

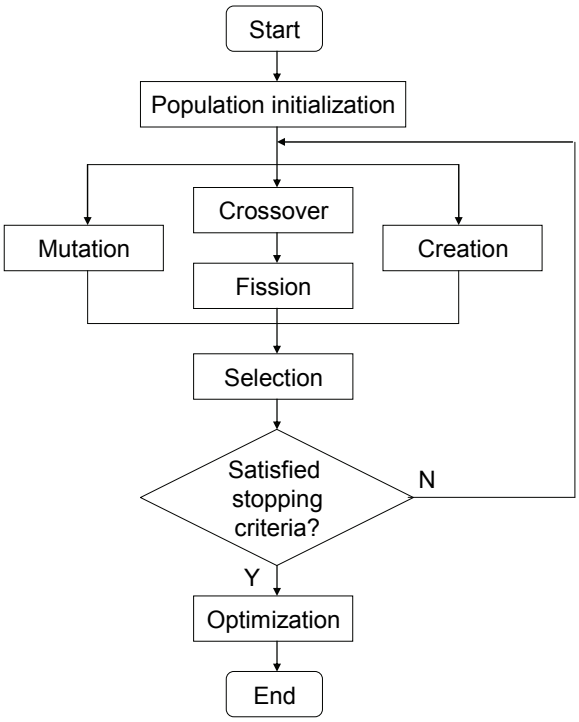

Figure 18. GA Flow Chart

4.2.1 Genetic Representation

A gene represents the ID of a peer. A chromosome contains a sequence of genes which represents the locus of a query routing path. Unlike conventional GAs, the length of a chromosome is variable. According to the definition in (29), any loop in a path is invalid. Hence, every gene in a chromosome is unique and the maximum length of any chromosome equals to the total number of peers in the network. Inside a chromosome, the first gene always represents the query initiating peer and the last gene represents a destination peer which can be any peer in the network. Figure 19 shows an example for representing two paths. The left figure shows the graph and two paths (solid line and dotted line) from the query initiating peer *A*. The right

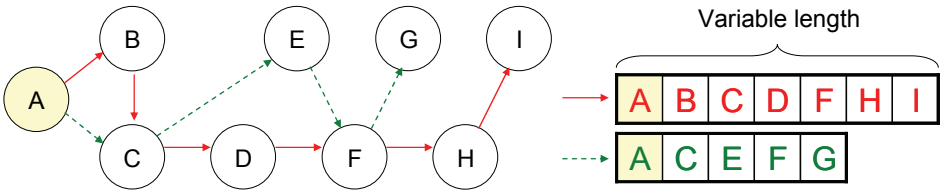

Figure 19. Genetic Representation of Two Paths

figure shows the corresponding chromosomes.

The genetic representation of our proposed GA is similar to that of Ahn's GA except that the last gene always represents the given destination node for Ahn's GA.

4.2.2 Population Initialization

The purpose of the population initialization is to create chromosomes for the first generation. Similar to Ahn's GA, we use random initialization instead of heuristic initialization for a better diversity of chromosomes.

The procedure of population initialization is that given an adjacency matrix A and a query initiating peer x_1, we randomly create N unique chromosomes where N is the population size and $n \leq N$. If there are not enough unique chromosomes, we randomly fill up some duplicated chromosomes to the remaining population. The population size should be proportional to the number of peers in the network in order to have a better quality of solution. To create a chromosome C, we first add x_1 to C. We also create an available peer list L and initialize L by adding all peers in the

Algorithm 1. Chromosome Creation

Creation (A, x_1) **returns** C

C := <x_1>, L := {all peers except x_1}, x := x_1
While $\exists_{y \in L} A_{xy} \neq 0$ **do**
  $x := y$
  Append x to C
  Remove x from L
End while

network to L except x_1. If a peer is in L, that means it has not been added to C so it is available. The purpose of using L is to prevent any loop formed in a path during the creation of chromosomes. This is similar to the use of a topological information database for Ahn's GA. Let x_{last} be the last peer in C. We randomly choose an adjacent peer x of x_{last} in L. Then we add x to C and remove x from L. This process continues until there is no more adjacent peer in L. Algorithm 1 shows the creation procedure of a chromosome which takes $O(|V|)$ time where $|V|$ is the total number of peers in the network. Hence, the population initialization takes $O(N \cdot |V|)$ time. Figure 20 shows a creation example of a chromosome.

The population initialization of our proposed GA is similar to that of Ahn's GA except that the last gene must be the given destination node for Ahn's GA. Therefore, all invalid chromosomes are reinitialized for Ahn's GA.

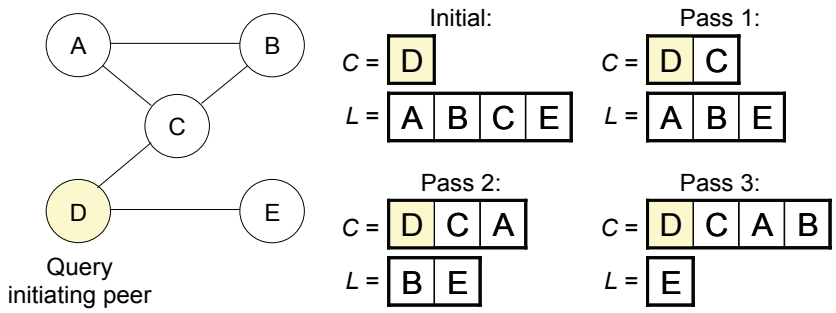

Figure 20. Creation of a Chromosome

4.2.3 Mutation

Similar to conventional GAs, the purpose of mutation is to reach the optimal solution by mutating some genes in a chromosome. In each generation, $N_m = \lceil N \cdot \lambda_m \rceil$ chromosomes are randomly chosen to be mutated and added to the new population for the selection where N is the population size and $0 \leq \lambda_m \leq 1$ is the mutation proportion.

The procedure of mutation is that given an adjacency matrix A, a score vector S and a chromosome C to be mutated, we randomly choose a mutation point m which is between the second gene and last gene. Then we delete all genes in C starting from m. We also create an available peer list L and initialize L by adding all peers in the network to L except those peers existing in C. If a peer is in L, this means that it has not been added to C so it is available. The purpose of using L is to prevent any loop being formed in a path during the mutation. Let x_{last} be the last peer in C. We choose an adjacent peer x of x_{last} in L, which has the highest score in S, by a greedy search. Then we add x to C and remove x from L. This process continues

Algorithm 2. Chromosome Mutation

Mutation (A, S, C) **returns** C

Randomly choose m where $2 \le m \le |C|$
For $i := m$ **to** $|C|$ **do**
  Remove x_i from C
End for
$L :=$ {all peers}, $x := x_{m-1}$
For each x' **in** C **do**
  Remove x' from L
End for
While $\exists_{y \in L} A_{xy} \neq 0 \wedge S_y$ is maximum **do**
  $x := y$
  Append x to C
  Remove x from L
End while

until there is no more adjacent peer in L. Algorithm 2 shows the mutation procedure of a chromosome which takes $O(|V|)$ time where $|V|$ is the total number of peers in the network. Hence, the mutation of N_m chromosomes in each generation takes $O(N_m \cdot |V|)$ time. Figure 21 shows a mutation example of a chromosome. E is adjacent to C with the highest score (four) and H is adjacent to F with the highest score (five).

The mutation of our proposed GA is slightly different from that of Ahn's GA. After deleting genes starting from m, Ahn's GA randomly choose an adjacent node instead of using the greedy search. We found that using the greedy search in the mutation has a fast convergence. Although the quality of solution is a bit lower, we improve it by using crossover and creation.

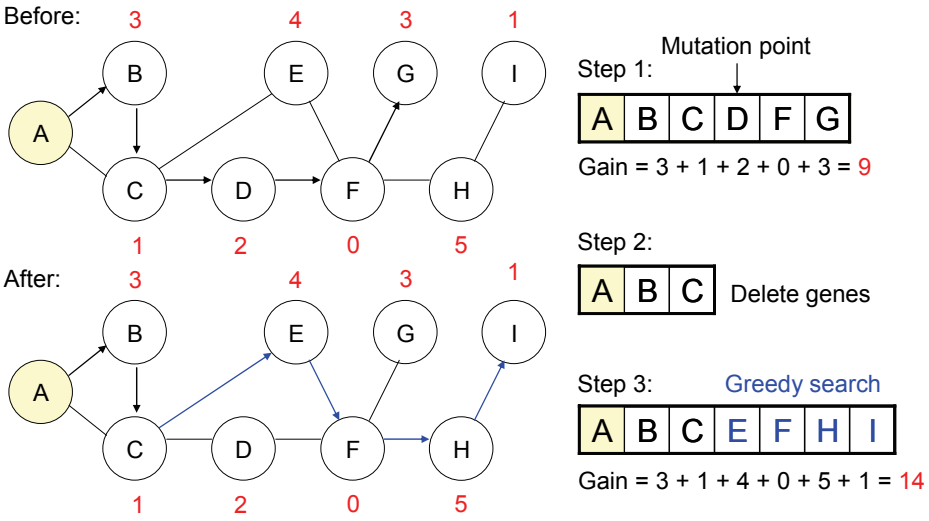

Figure 21. Mutation of a Chromosome

4.2.4 Crossover

Since the mutation adopts a greedy search which may be trapped by local optima, crossover is proposed to escape these traps by crossing two chromosomes which produce better chromosomes. In each generation, $N_c = \lceil N \cdot \lambda_c \rceil$ chromosomes are randomly chosen to cross with other chromosomes and added to the new population for the selection where N is the population size and $0 \leq \lambda_c \leq 1$ is the crossover proportion.

The procedure of crossover is that given two chromosomes C_1 and C_2, we find out all pairs of common genes which form a list of potential crossing points L. We randomly choose a pair of crossing points (r, s) in L where $2 \leq r \leq |C_1|$ and $2 \leq s \leq |C_2|$. Then we perform the crossover which produces two new chromosomes C_1' and C_2' such that C_1' contains the genes from the first gene to the gene just before r in C_1 and from the gene at s to

the last gene in C_2, while C_2' contains the genes from the first gene to the gene just before s in C_2 and from the gene at r to the last gene in C_1. If there is no common gene, then crossover is impossible. Algorithm 3 shows the crossing points finding procedure of two chromosomes which takes $O(l \cdot \log l)$ time and Algorithm 4 shows the crossover procedure of two chromosomes which takes $O(l)$ time where l is the length of a chromosome. Hence, the crossover of N_c chromosomes in each generation takes $O(N_c \cdot l \cdot \log l)$ time. Figure 22 shows a crossover example of two chromosomes. The crossover produces better chromosomes because both parents contain optimal partial paths. Both C and F are common genes so there are two potential crossing points. We choose C as the crossing point in this example. The crossover of our proposed GA is exactly the same as that of Ahn's GA.

Algorithm 3. Crossing Points Finding

Crossing-points-finding (C_1, C_2) **returns** L

Quick-sort peers' IDs in C_1 and C_2 in non-descending order
$L := \phi$, $i := 1$, $j := 1$
While $i \leq |C_1|$ **and** $j \leq |C_2|$ **do**
  If i^{th} sorted ID in $C_1 < j^{th}$ sorted ID in C_2 **then**
    $i := i + 1$
  Else if i^{th} sorted ID in $C_1 > j^{th}$ sorted ID in C_2 **then**
    $j := j + 1$
  Else
    If i^{th} sorted ID in $C_1 \neq$ query initiating peer ID **then**
      $r :=$ original position of i^{th} sorted ID in C_1
      $s :=$ original position of j^{th} sorted ID in C_2
      Add (r, s) to L
    End if
    $i := i + 1$
    $j := j + 1$
  End if
End while

Algorithm 4. Chromosomes Crossover

Crossover (C_1, C_2, L) **returns** C_1', C_2'

Randomly choose (r, s) in L
$C_1' := \phi$, $C_2' := \phi$, $x := C_1$, $y := C_2$
For $i := 1$ **to** $r-1$ **do**
  Append x_i to C_1'
End for
For $i := s$ **to** $|C_2|$ **do**
  Append y_i to C_1'
End for
For $i := 1$ **to** $s-1$ **do**
  Append y_i to C_2'
End for
For $i := r$ **to** $|C_1|$ **do**
  Append x_i to C_2'
End for

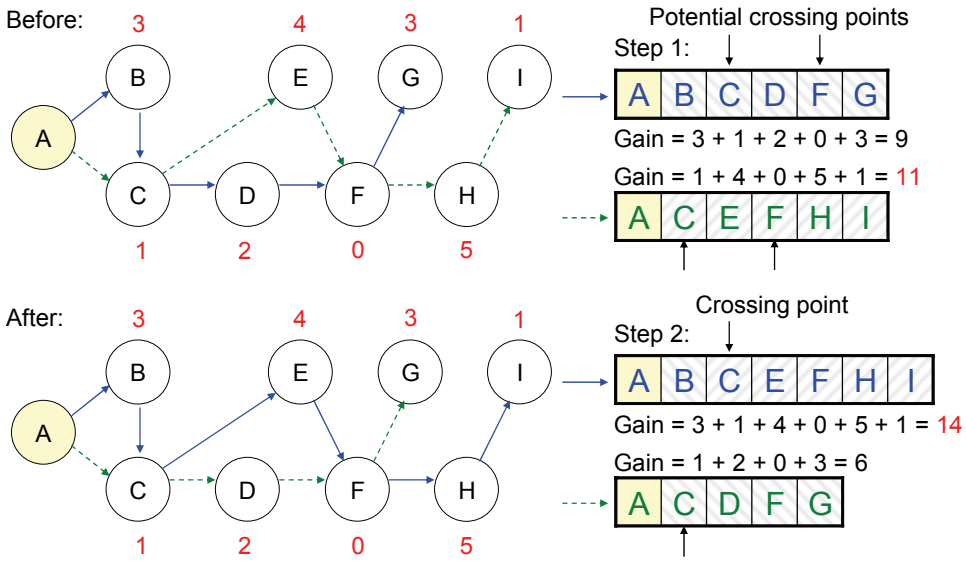

Figure 22. Crossover of Two Chromosomes

4.2.5 Fission

Since the crossover may produce invalid chromosomes which violate the loop constraint in (29), fission is proposed to break an invalid chromosome down to several valid chromosomes. Consider two chromosomes $C_1 = <A, C, B>$ and $C_2 = <A, B, C, D, F, G>$ with the crossing point (2, 3) in Figure 22, the two new chromosomes produced after the crossover are $C_1' = <A, C, D, F, G>$ and $C_2' = <A, B, C, B>$. However, C_2' is invalid since it violates the loop constraint. One of the solutions is to remove any invalid chromosome after the crossover. But this wastes some produced chromosomes because those invalid chromosomes can be repaired by using *fission* which is a novel GA operator.

The procedure of fission is that given an invalid chromosome C, we find out the first pair of common genes x which is the fission point (u, v)

where $u < v$. Then we perform the fission which produces two new chromosomes C_1' and C_2' such that C_1' contains the genes from the first gene to the gene just before v in C, while C_2' contains the genes from the first gene to the gene just before u and from the gene at v to the last gene in C. We recursively perform the fission procedure on C_1' and C_2' until all new chromosomes are valid. Algorithm 5 shows the fission point finding procedure of a chromosome which takes $O(l)$ time and Algorithm 6 shows the fission procedure of a chromosome which also takes $O(l)$ time where l is the length of a chromosome. Hence, the fission of N_f chromosomes in each generation takes $O(N_f \cdot l)$ time. Figure 23 shows a fission example of a chromosome. The first pair of common genes is C. The information gain of both new chromosomes decreases but they become valid after fission.

Since the crossover of our proposed GA is exactly the same as that of Ahn's GA, the problem of invalid chromosomes also exists in Ahn's GA. Ahn's GA uses a repair function to solve the problem by deleting genes from $u + 1$ to v. Consider the example in Figure 23, the valid chromosome obtained by the repair function is <A, B, C, D, F, G> which is a subset of the valid chromosomes obtained by our proposed fission. The other chromosome <A, B, C, D, F, E> is invalid for Ahn's GA because the last gene is not the given destination node G. Hence, our proposed fission is the generalization of the repair function of Ahn's GA.

Algorithm 5. Fission Point Finding

Fission-point-finding (C) **returns** (u, v)

$u = 0$, $v = 0$, $L := \phi$
For $i := 1$ **to** $|C|$ **do**
  If $x_i \notin L$ **then**
    Add x_i to L
  Else
    $x := x_i$
    $v := i$
    $i := |C|$
  End if
End for
For $i := 1$ **to** v–1 **do**
  If $x_i = x$ **then**
    $u := i$
    $i := v$–1
  End if
End for

Algorithm 6. Chromosome Fission

Fission (C, (u, v)) **returns** C_1', C_2'

$C_1' := \phi$, $C_2' := \phi$
For $i := 1$ **to** v–1 **do**
  Append x_i to C_1'
End for
For $i := 1$ **to** u–1 **do**
  Append x_i to C_2'
End for
For $i := v$ **to** $|C|$ **do**
  Append x_i to C_2'
End for

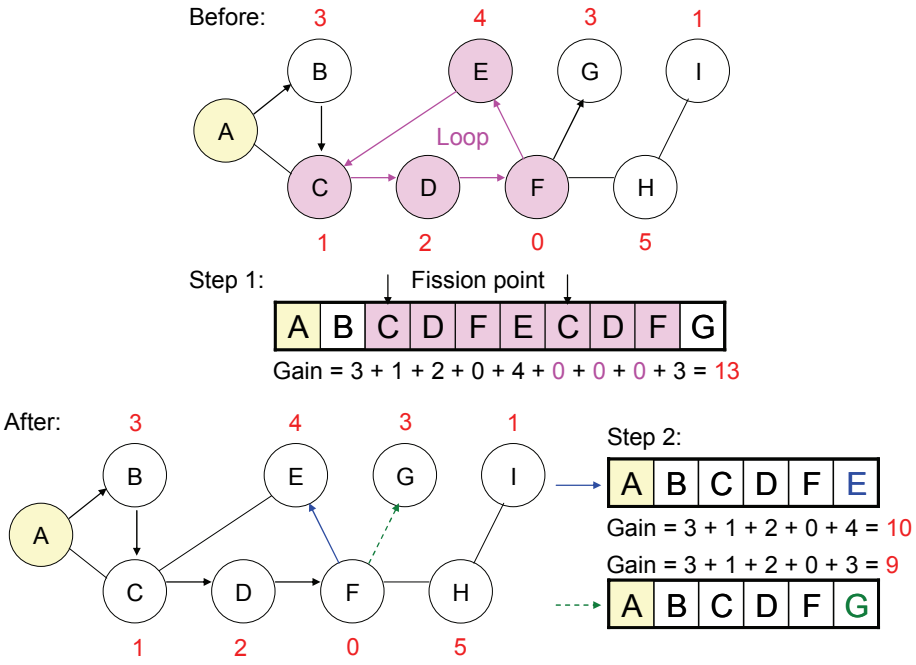

Figure 23. Fission of a Chromosome

4.2.6 Creation

Since mutation and crossover produce evolved chromosomes which provide fast convergence but less diversity, a novel GA operator called *creation* is proposed to randomly create non-evolved chromosomes which provide extra diversity. Creation is also significant to the quality of solution when the crossover cannot be performed due to the lack of the potential crossing point. On the other hand, the problem of the quality of solution does not exist in Ahn's GA even though the crossover cannot be performed. The reason is that the mutation of Ahn's GA adopts a random search instead of a greedy search which already provides enough diversity.

In each generation, $N_n = \lceil N \cdot \lambda_n \rceil$ chromosomes are randomly created

and added to the new population for the selection where N is the population size and $\lambda_n \geq 0$ is the creation rate. The algorithm of creation is the same as that of population initialization. Algorithm 1 shows the creation procedure of a chromosome which takes $O(|V|)$ time where $|V|$ is the total number of peers in the network. Hence, the creation of N_n chromosomes in each generation takes $O(N_n \cdot |V|)$ time. Figure 20 shows a creation example of a chromosome.

4.2.7 Selection

The selection process is to select the best chromosomes from the new population for the next generation to ensure the population size is N as mutation, crossover, fission, and creation produce new chromosomes which exceed the fixed population size. In each generation, N_g good chromosomes with the highest fitness are first selected where $n \leq N_g \leq N$ and n is the maximum number of paths to be returned by GAroute. The remaining population is randomly filled by $N - N_g$ poor chromosomes. This is to enhance the diversity because those poor chromosomes may produce good chromosomes in the future. The fitness f_C of a chromosome C is exactly the same as the information gain H_p of its corresponding path p (refer to (30)).

The sequence of selecting paths is important because it affects the number of paths that cover the relevant peers. Consider two paths $p_1 = <A, B, C>$ and $p_2 = <A, B, C, D>$ where A is the query initiating peer. If we first select p_1 and route the query through p_1, we gain the information of B and C.

Then we select p_2 and route the query through p_2. We gain the information of D only because the information of B and C is gained through p_1. Totally, we need to route two times to gain the information of B, C, and D. On the other hand, if we first select p_2 and route the query through p_2, we gain the information of B, C, and D. Therefore, we do not need to further select p_1 and route the query through p_1 because there is no more information gain. Totally, we only need to route one time. From the above observation, we should always select paths in a descending order by the information gain.

The procedure of selection is that given a score vector S, an original population P_o before the selection and the aforementioned N and N_g, we first create a set of the current visited peers V' and initialize V' to an empty set. We also calculate the fitness f_i for each unselected chromosome C_i based on S and V'. Then we select the chromosome C with the highest fitness (primary condition) and maximum length (secondary condition). If there is more than one chromosome with the highest fitness and maximum length, then we randomly select one. Moreover, we update V' by adding all peers in C to V'. This process continues until the number of selected chromosomes reaches N_g. Finally, $N - N_g$ unselected chromosomes are randomly selected and a new population P_n is returned after the selection. Algorithm 7 shows the selection procedure in each generation which takes $O(N_g \cdot |P_o| \cdot l)$ time where l is the length of a chromosome. Figure 24 shows an example for selecting two out of four chromosomes. In Pass 2, both second and third chromosomes have the same fitness. We select the second chromosomes

Algorithm 7. Chromosome Selection

Selection (S, P_o, N, N_g) **returns** P_n

$P_n := \phi$, $V' := \phi$
For $k := 1$ **to** N_g **do**
  For $i := 1$ **to** $|P_o|$ **do**
    Calculate fitness f_i of C_i based on S and V'
  End for
  C := chromosome with highest fitness and max. length
  Move C from P_o to P_n
  For each x **in** C **do**
    Add x to V'
  End for
End for
For $k := 1$ **to** $N - N_g$ **do**
  Randomly move a chromosome from P_o to P_n
End for

because its length is longer.

The fitness of a chromosome in Ahn's GA is inversely proportional to the total cost of its corresponding path because Ahn's GA aims at finding the shortest path. Furthermore, the selection algorithm of Ahn's GA is based on the pair-wise tournament selection without replacement such that we select the fitter chromosome in each two chromosomes.

4.2.8 Stopping Criteria

After the selection, a generation cycle is completed and there are three ways for us to terminate which are the (1) solution convergence, (2)

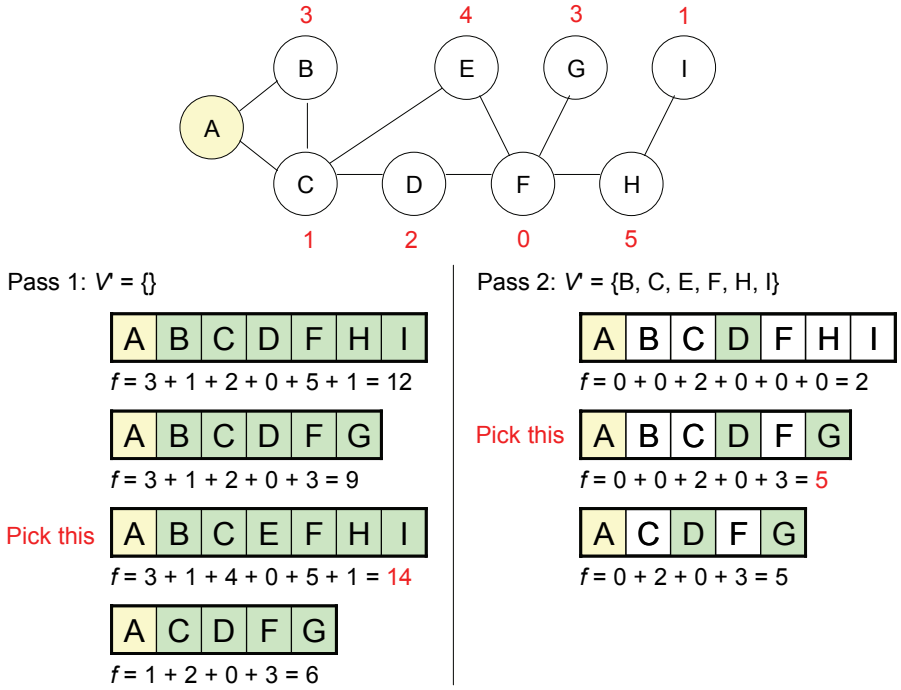

Figure 24. Selection of Four Chromosomes

minimum number of generations, and (3) maximum number of generations.

1) Solution Convergence: We compare all chromosomes in the current generation with those in the previous generation. If they are all the same, then our solution converges. However, it may take long time to converge because each generation contains some randomly selected chromosomes which are difficult to be the same. A better strategy is to compare N_g good chromosomes only where N_g is the number of good chromosomes during the selection. Such strategy makes a balance between the quality and time. Since N_g good chromosomes are always sorted by their fitness after the selection, the comparison of N_g good chromosomes between two generations takes $O(N_g \cdot l)$ time where l is the length of a chromosome. On

Algorithm 8. Two-phase Tail Pruning

```
Two-phase-tail-pruning (S, P) returns P'
P' := ϕ, V' := ϕ
For p := first path to last path in P do
   For x := last peer down to second peer in p do
      If S_x = 0 or x ∈ V' then
         Delete x from p
      Else
         x := second peer
      End if
   End for
   Calculate information gain H_p of p based on S and V'
   If H_p > 0 then
      Append p to P'
      For each x in p do
         Add x to V'
      End for
   Else
      p := last path
   End if
End for
```

the other hand, Ahn's GA stops when all chromosomes in the population are the same which is slow.

2) Minimum Number of Generations: We introduce the minimum number of generations G_{min} for preventing under-training which yields low quality solutions. We do not stop the iteration if the current number of generations does not reach G_{min} even if N_g good chromosomes are the same between two generations.

3) Maximum Number of Generations: We introduce the maximum number of generations G_{max} for preventing overtime. We stop the iteration at once if the current number of generations exceeds G_{max} even if N_g good chromosomes are different between two generations.

4.2.9 Optimization

Optimization procedure can be performed after satisfying the stopping criteria so that a better result is obtained. Since we obtain a sorted list of N_g good chromosomes where N_g is the number of good chromosomes during the selection, we can return the first n good chromosomes in the last generation representing the required paths P where n is the maximum number of paths requested by the query initiating peer and $n \leq N_g$. However, these returned paths can still be optimized by our proposed *two-phase tail pruning* optimization technique.

The procedure of two-phase tail pruning is that given a score vector S and a sorted list of paths P to be optimized. In Phase I, we prune away each path p in P which information gain H_p is zero. In Phase II, we start from the last peer in a path and prune away each peer x which score S_x is zero or is visited through the previous paths (i.e. $x \in V'$ where V' is a set of the current visited peers). Finally, a sorted list of optimized path P' is returned after the optimization. Algorithm 8 shows a faster two-phase tail pruning procedure which takes $O(n \cdot l)$ time where l is the length of a path. Both Phase I and Phase II are running at the same time. Figure 25 shows an example for the

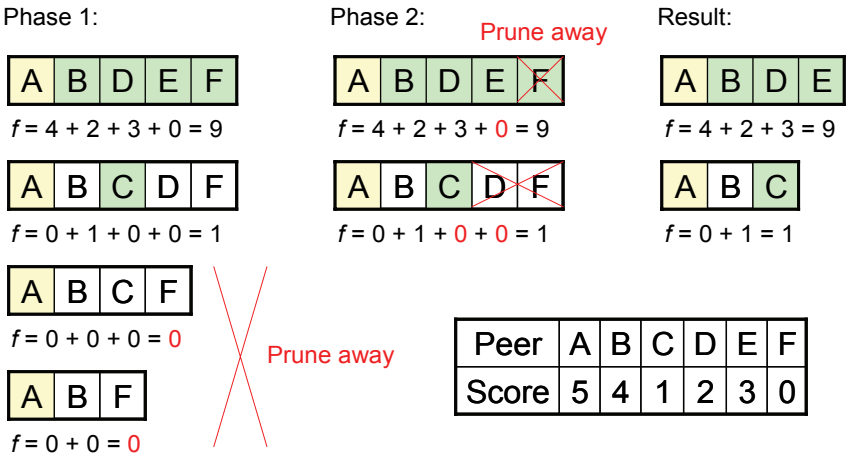

Figure 25. Two-phase Tail Pruning of Four Paths

two-phase tail pruning of four paths. Consider the second path in Phase II, we do not remove *B* because the adjacent peer of *A* is not *C*.

Table 6 shows the difference between Ahn's GA and GAroute. Finally, Table 7 shows the time complexities of GAroute which are all in the polynomial time. We verify them by the experimental results which are discussed in Section 4.3.

Table 6. Difference between Ahn's GA and GAroute

	Ahn's GA	**GAroute**
Purpose	Find one shortest-path from one given source to one given destination	Find *n* long paths from one given source to any destination
Genetic representation	The last gene is always the given destination	The last gene is any destination
Population initialization	May produce invalid chromosomes	Always produce valid chromosomes
Mutation	Randomly choose an adjacent node	Greedily choose an adjacent node
Loop elimination	Use a repair function to eliminate loops in an invalid chromosome	Use fission to break an invalid chromosome down to valid chromosomes
Diversity enhancement	Use mutation to enhance diversity	Use creation to provide extra diversity
Fitness function	Inversely proportional to the total cost of its corresponding path	Directly proportional to the information gain of its corresponding path
Selection	Use pair-wise tournament selection without replacement	Base on the fitness and chance to select good and poor chromosomes
Stopping criteria	Stop if all chromosomes in the population are the same	Besides G_{min} and G_{max}, it stops if all good chromosomes between two consecutive generations are the same
Optimization	None	Two-phase tail pruning

Table 7. Time Complexities of GAroute

Procedure	Time complexity
Population initialization	$O(N \cdot \|V\|)$
Mutation	$O(N_m \cdot \|V\|)$
Crossover	$O(N_c \cdot l \cdot \log l)$
Fission	$O(N_f \cdot l)$
Creation	$O(N_n \cdot \|V\|)$
Selection	$O(N_g \cdot \|P_o\| \cdot l)$
Stopping criteria checking	$O(N_g \cdot l)$
Optimization	$O(n \cdot l)$

4.3 Experiments and Discussions

In this section, we summarize the experimental results which measure the scalability and quality of different searching algorithms for finding query routing paths in different network topologies and peer quantities. We also verify the two improvements (lower bandwidth consumption and better parallel search) of the query initiating peer in our proposed hybrid P2P network model. Before presenting our experiments, we introduce our computer configuration and GAroute parameters. We also outline our P2P network topology generation algorithm.

1) Configuration and Parameters: All experiments are performed with the same computer configuration (see Table 8). We conduct our experiments with different parameter sets and choose the most suitable one (see Table 9) so that we obtain the most representative results. In fact, the maximum number of paths n and population size N should be proportional to the peer

Table 8. Computer Configuration of GAroute Experiments

Item	Setting
CPU	Intel Pentium 4 at 3GHz
Memory	512MB DDR RAM
OS	Red Hat Linux 9.0 with Kernel 2.4.20-31.9
Java VM	Java 2 Standard Edition 1.4.2_05

Table 9. GAroute Parameters

Parameter	Value
Max. no. of paths (n)	10
Population size (N)	100
No. of mutations (N_m)	50
No. of crossovers (N_c)	50
No. of creations (N_n)	10
No. of good chromosomes (N_g)	20
Min. no. of generations (G_{min})	0
Max. no. of generations (G_{max})	100

quantity. However, we fix them to a specific value in order to measure their effects on the quality of solution. In each generation, half of the population performs mutation and another half of the population performs crossover as the convergence and diversity are both important. We also create a few non-evolved chromosomes so that a better quality of solution is obtained. In addition, the parameters should satisfy the constraints that $n \leq N_g \leq N$ and $N_m, N_c \leq N$.

2) P2P Network Topology Generation: We use our P2P network topology generation algorithm, which simulates the joining scenario of

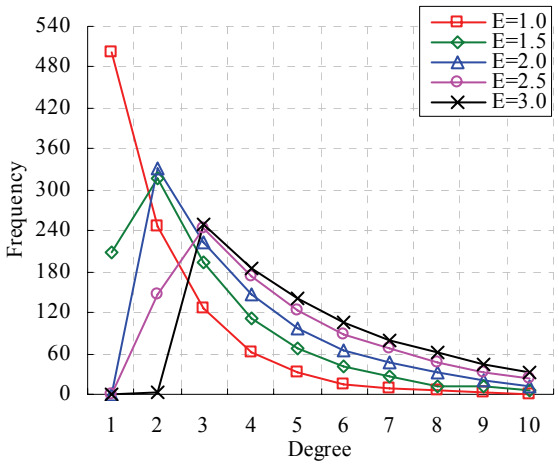

Figure 26. Property of Different Network Topologies

peers in a zone, to randomly generate undirected graphs for all experiments. The graph is initialized to have only one peer. Each time when a new peer is added to the graph, it links a random number of existing peers in the graph based on their inter-connection speed. In our proposed hybrid P2P network model, each peer chooses one or two peers to link and the expected number E of existing peers to be linked by a new peer is 1.2. Both E and the inter-connection speed are the parameters used to generate different graphs.

4.3.1 Property of Different Topologies

The motivation of this experiment is to study how different network topologies would affect the scalability and quality of different searching algorithms for finding query routing paths. Figure 26 shows the property of different network topologies for 1,000 peers, which are generated by our

P2P network topology generation algorithm. The *x*-axis shows the degree of peers representing the number of edges and the *y*-axis shows the number of peers that have the corresponding degree. The results show that the degree is inversely proportional to the frequency because each new peer links to old peers from time to time. Therefore, older peers are usually linked by more peers that results in a higher degree and lower frequency, and newer peers are usually linked by fewer peers that results in a lower degree and higher frequency. The results also show that when the expected number E of existing peers to be linked by a new peer increases, the number of edges of each peer increases. Increasing edges in a graph dramatically increases the time of brute-force search which is discussed in the following subsection.

4.3.2 Scalability and Quality in Different Topologies

The motivation of this experiment is to measure the scalability and quality of Brute-force Search (BS), GAroute (GA) and Greedy Search (GS) for finding query routing paths in different network topologies. We demonstrate that GAroute achieves both good scalability and quality in some network topologies. Before we analyze and interpret the results, we outline the algorithm of (1) BS and (2) GS.

1) Algorithm of BS: We start at the query initiating peer and traverse the graph by using depth-first search until a path p is generated. Then we calculate the information gain H_p of p based on the score vector S. If H_p is greater than the current maximum information gain H_{max}, then p is the

current best query routing path and H_{max} becomes H_p. We back-track p from the last peer, look for another edge and traverse it until another path is generated. This process continues until all paths are searched and we obtain the best path. To obtain the second-best path, we update S and set the score of those visited peers to zero. Then we reapply the same procedure. This process continues until we obtain n paths. BS guarantees the paths obtained are always optimal but it takes $O(n \cdot N_{path})$ time where N_{path} is the number of different paths in a graph. A path is an edge combination so N_{path} can be very large.

2) Algorithm of GS: We start at the query initiating peer and traverse the graph by using greedy search. We always select the next peer with the highest score based on the score vector S until a path p is generated. Thus, we obtain a good query routing path though it may not be the best. To obtain another good path, we update S and set the score of those visited peers to zero. Then we reapply the same procedure. This process continues until we obtain n paths. GS takes $O(n \cdot |V|)$ time where $|V|$ is the number of peers in a graph. Sometimes, it may give low quality solutions.

To measure the (1) scalability and (2) quality of BS, GA, and GS for finding query routing paths in different network topologies, we randomly generate 10 different graphs for each E containing 50 peers where E is the expected number of existing peers to be linked by a new peer. Then we run BS, GA, and GS on a graph for 10 times and measure their average searching time and quality. In all, we run each algorithm with each E for 100

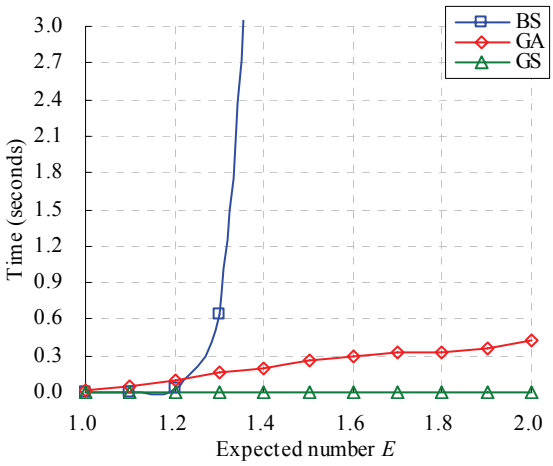

Figure 27. Scalability in Different Network Topologies

times.

1) Scalability: Figure 27 shows the scalability in different network topologies. We measure the real searching time in order to compare the practicability of different algorithms. The curve BS is exponential because BS takes $O(n \cdot N_{path})$ time. When E increases, the number of edges of each peer increases. Thus, the number of edge combinations N_{path} dramatically increases. On the other hand, the searching time of GA is good and directly proportional to E because when E increases, the number of edges of each peer increases. Therefore, the connectivity increases and the number of peers in a path increases which requires more time to perform GA operations (see Table 7). Finally, the line GS is constant with ultrahigh searching speed because it only takes $O(n \cdot |V|)$ time where n and $|V|$ are fixed (see Table 9).

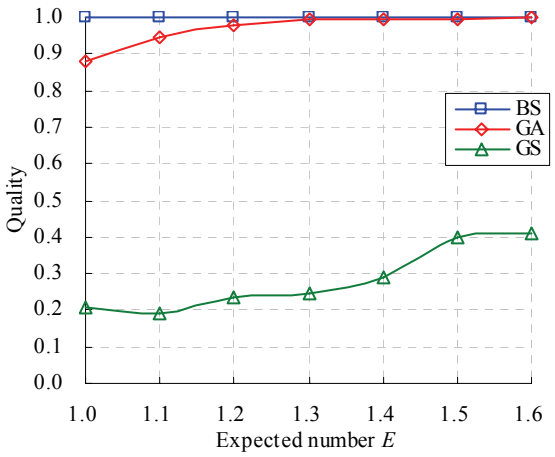

Figure 28. Quality in Different Network Topologies

2) Quality: Figure 28 shows the quality Q_A from Algorithm A in different network topologies such that

$$Q_A = \frac{H_A}{H_{BS}}, \tag{32}$$

where H_A and H_{BS} are the total information gain of n paths (refer to (30)) obtained by A and BS respectively. We use BS as the reference because BS always gives global optimal solutions. Therefore, Q_{BS} is always one. Since BS takes too long time to run if E is greater than 1.6, we can only calculate the quality up to this value. We observe that Q_{GA} is high and Q_{GS} is low because GS returns local optimal solutions. Both curves GA and GS tend to the line BS when E increases because the number of edges of each peer increases. Thus, the connectivity increases and the number of peers in a path p increases. This makes the information gain of p increase and hence the quality increases. The definition of the quality does not consider the length

of the paths because we only focus on the total information gain in our proposed hybrid P2P network model. However, we may also consider the path length as a factor because the total length of the paths affects the bandwidth consumption of the whole network and the maximum path length affects the latency. To achieve this, we can simply modify (30) to normalize the information gain of a path by its length.

From the experimental results, we demonstrate that BS is un-scalable as it is highly dependent of the network topology though it always gives global optimal solutions. Moreover, GS gives low quality solutions though it is scalable. On the other hand, GAroute is scalable and gives high quality solutions though there is a tradeoff between searching speed and quality.

4.3.3 Scalability and Quality in Different Quantities

The motivation of this experiment is to measure the (1) scalability and (2) quality of BS, GA, and GS for finding query routing paths in different peer quantities. We demonstrate that GAroute achieves both good scalability and quality in some large scaled network topologies. To measure them, we randomly generate 10 different graphs for each peer quantity and E is 1.2 (default value used in our proposed hybrid P2P network model), where E is the expected number of existing peers to be linked by a new peer. Then we run BS, GA, and GS on a graph for 10 times and measure their average searching time and quality. In all, we run each algorithm with each peer quantity for 100 times.

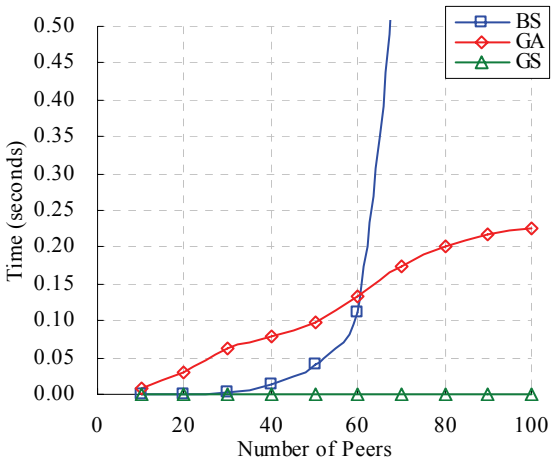

Figure 29. Scalability in 100 Peers

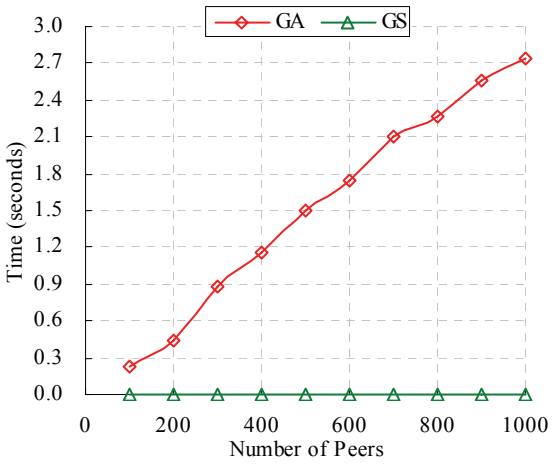

Figure 30. Scalability in 1,000 Peers

1) Scalability: Figure 29 and Figure 30 show the scalability in 100 peers for every 10 peers and 1,000 peers for every 100 peers respectively. We measure the real searching time in order to compare the practicability of different algorithms. The curve BS is exponential because BS takes $O(n \cdot$

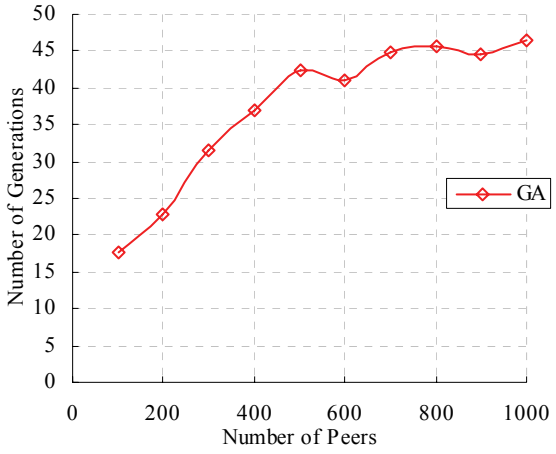

Figure 31. Generation Requirement of GA in 1,000 Peers

N_{path}) time. When the peer quantity increases, the total number of edges increases. Thus, the number of edge combinations N_{path} dramatically increases. On the other hand, the curve GA is approximately linear which is interesting as it is theoretically super-linear (see Table 7). Figure 31 shows the generation requirement of GA in 1,000 peers, which gives a possible reason on the approximately linear searching time. We observe that the generation requirement, which is a factor of searching time complexity, is sub-linear. Also, the parameters n, N, N_m, N_c, N_n, and N_g are fixed (see Table 9) which do not affect the searching time complexity. Due to the effect of sub-linear generation requirement and constant parameters, the curve GA becomes approximately linear. Finally, the searching speed of GS is ultrahigh because it only takes $O(n \cdot |V|)$ time where n is fixed.

2) Quality: Figure 32 shows the actual quality Q_A from Algorithm A

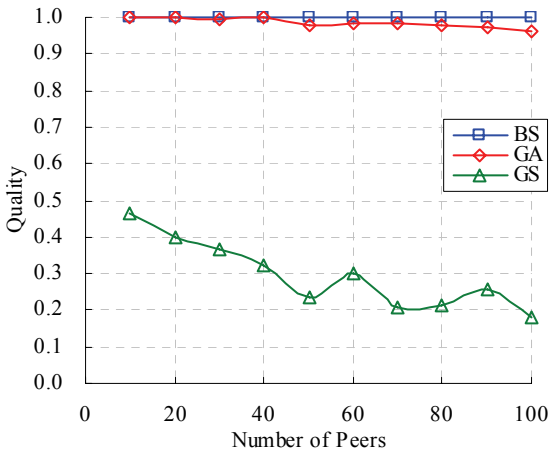

Figure 32. Actual Quality in 100 Peers

(refer to (32)) in 100 peers for every 10 peers obtained by A and BS respectively. We use BS as the reference because BS always gives global optimal solutions. Therefore, Q_{BS} is always one. Since BS takes too long time to obtain Q_{GA} and Q_{GS} if the peer quantity is more than 100, we calculate the relative quality instead. Figure 33 shows the relative quality Q_A' from Algorithm A in 1,000 peers for every 100 peers such that

$$Q_A' = \frac{H_A}{H_{GA}}, \tag{33}$$

where H_A and H_{GA} are the total information gain of n paths (refer to (30)) obtained by A and GA respectively. We use GA as the reference so Q_{GA}' is always one. We observe that Q_{GA} is high in 100 peers. Q_{GS}' is low intuitively represents that Q_{GA} is still high in 1,000 peers. However, Q_{GA} decreases when the peer quantity increases because the total number of edges increases. Thus, the number of different paths in a graph increases.

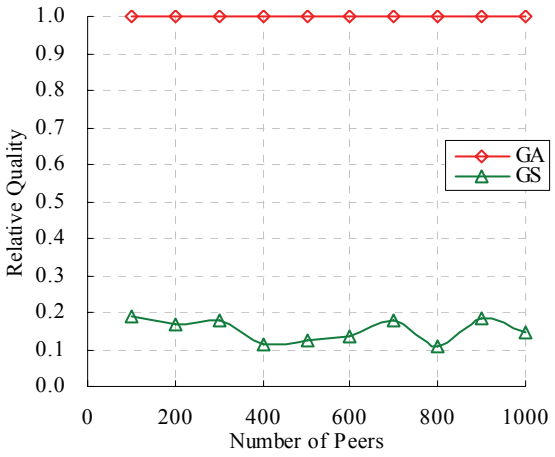

Figure 33. Relative Quality in 1,000 Peers

Furthermore, the parameters N, N_m, N_c, N_n, and N_g are fixed (see Table 9) which make the quality decrease. If the value of the parameters increases, then the quality increases. However, the searching time also increases. Moreover, Q_{GS} is low because GS returns local optimal solutions. Q_{GS} decreases when the peer quantity increases because the chance for GS to give low quality solutions increases when the number of different paths in a graph increases. Finally, Figure 34 shows the convergence of GA in 100 peers. The curve GA tends to the line BS when the number of generations increases. However, Q_{GA} slightly increases when the number of generations is large. Therefore, introducing the maximum number of generations G_{max} reduces the unnecessary search if there is no big difference in the quality.

From the experimental results, we demonstrate that BS is un-scalable as it is highly dependent of the peer quantity though it always gives global

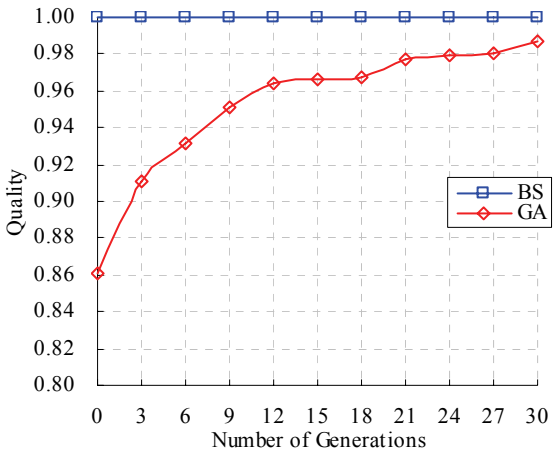

Figure 34. Convergence of GA in 100 Peers

optimal solutions. Moreover, GS usually gives low quality solutions though it is scalable. On the other hand, GAroute is scalable and gives high quality solutions though there is a tradeoff between searching speed and quality. Combining all experimental results, GAroute achieves both good scalability (approximate linear) and quality (0.95 in 100 peers) in some large scaled P2P network topologies.

4.3.4 Verification of Lower Bandwidth Consumption

The motivation of this experiment is to measure the improvement of the query initiating peer bandwidth consumption in our proposed hybrid P2P network model. We demonstrate that our model greatly reduces the query initiating peer network traffic with a small overhead of the whole network traffic. To measure the network traffic, we randomly generate 10 different

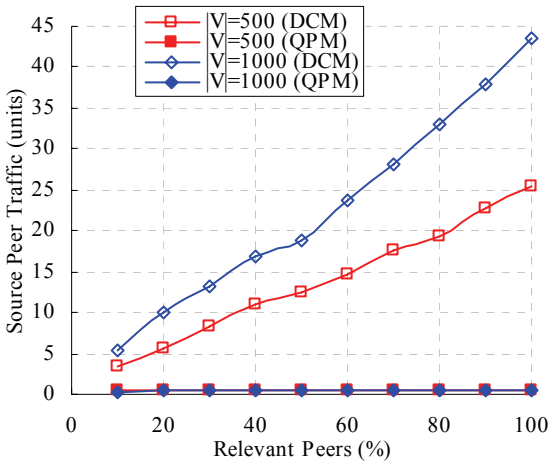

Figure 35. Query Initiating Peer Network Traffic of DCM and QPM

graphs for each peer quantity $|V|$ and E is 1.2 (default value used in our proposed hybrid P2P network model), where E is the expected number of existing peers to be linked by a new peer. Then we run GAroute on a graph with different relevant peer proportions to obtain query routing paths. We measure the average network traffic of the (1) query initiating peer and (2) whole network based on DCM and QPM.

1) Query Initiating Peer Network Traffic: Figure 35 shows the query initiating peer network traffic for every 10 percent relevant peer proportion. Both DCM curves are approximately linear because the query initiating peer sends a query packet to each relevant peer individually. If the number of relevant peers increases, then the query initiating peer bandwidth consumption increases. On the other hand, both QPM curves are nearly constant because the query initiating peer only sends query packets to the

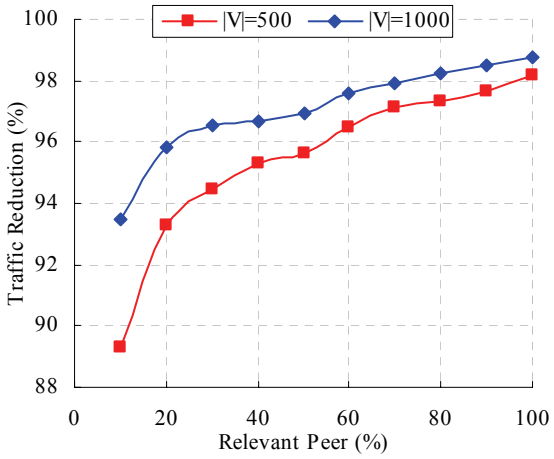

Figure 36. Network Traffic Reduction of Query Initiating Peer by QPM

next peers in the query routing paths instead of all relevant peers. Since the degree of a peer is usually small according to our P2P network topology property (see Figure 26), the query initiating peer consumes very low bandwidth. Figure 36 shows the corresponding traffic reduction T_R in percentage for every 10 percent relevant peer proportion that is calculated by

$$T_R = \frac{T_{DCM} - T_{QPM}}{T_{DCM}}, \tag{34}$$

where T_{DCM} and T_{QPM} are the traffic of DCM and QPM respectively. When $|V|$ increases, the number of relevant peers increases. T_{DCM} also increases but T_{QPM} is unchanged. Hence, the traffic reduction increases which shows the good scalability of our proposed hybrid P2P network model.

2) Whole Network Traffic: Figure 37 shows the whole network traffic for every 10 percent relevant peer proportion. When $|V|$ increases, the

number of relevant peers increases. Hence, the whole network traffic of both DCM and QPM increases. In addition, the whole network traffic of QPM is always larger than or equals to that of DCM because query packets are propagated from peer to peer for QPM. However, not all peers in the query routing path are relevant. On the other hand, the query initiating peer always sends query packets to relevant peers for DCM. Fortunately, the whole network traffic of QPM is always bounded by that of DCM when all peers are relevant. At this point (100 percent relevant peer proportion), they have the same network traffic. Figure 38 shows the corresponding traffic overhead T_O for every 10 percent relevant peer proportion that is calculated by the difference between the traffic of DCM and QPM. When the number of relevant peers increases, the traffic overhead decreases because there are more relevant peers in the query routing path. The overhead is zero if all peers are relevant.

From the experimental results, we demonstrate that the query initiating peer consumes high bandwidth for DCM, whereas it consumes very low bandwidth for QPM. Moreover, using DCM is un-scalable. On the other hand, using QPM is scalable though it has a small overhead on the whole network traffic. In conclusion, we verify that the query initiating peer has lower bandwidth consumption by using QPM.

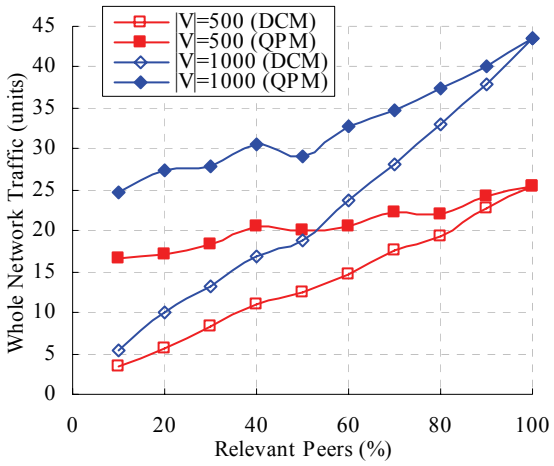

Figure 37. Whole Network Traffic of DCM and QPM

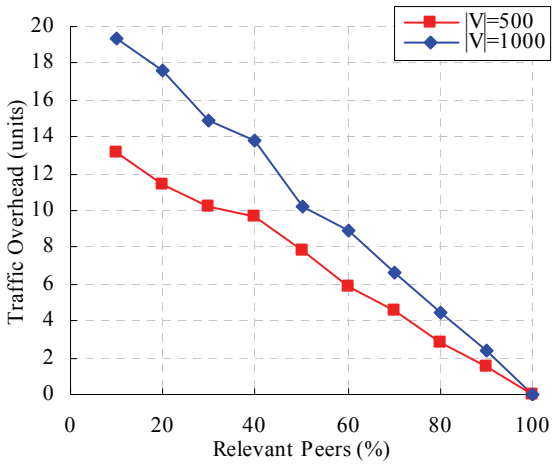

Figure 38. Network Traffic Overhead of Whole Network by QPM

4.3.5 Verification of Better Parallel Search

The motivation of this experiment is to measure the improvement of the query time in our proposed hybrid P2P network model. We demonstrate that our model is efficient if the peer quantity is large. To simulate the query

routing in our network, we generate two types of topologies which are physical network topology and P2P network topology. The former one represents the real topology which has the properties of the Internet. The latter one represents the logical topology which has the properties of our proposed hybrid P2P network model. A recent paper [54] mentions that the Internet follows Autonomous System (AS) Model which has the small world and power law properties. Therefore, we use BRITE [9] which is a tool to generate physical network topologies based on the AS Model. We also use our P2P network topology generation algorithm to generate P2P network topologies based on the inter-connection speed of peers which is inversely proportional to the physical distance between peers.

To measure the query time, we randomly generate 10 different sets of network topologies and present the results of three selected sets. Each set contains a physical network topology with 10,000 nodes, and a P2P network topology with 1,000 peers and E is 1.2 (default value used in our proposed hybrid P2P network model), where E is the expected number of existing peers to be linked by a new peer. Those peers are the subset of the nodes of the physical network topology. Then we run GAroute on each P2P network topology to obtain the query routing paths, and calculate the query time for both DCM and QPM. The query propagation time between two peers x_i and x_j is the shortest path distance $d(x_i, x_j)$ between the corresponding nodes in the physical network topology. Recall that DCM has the poor semi-parallel search problem that we spawn a few threads in each batch if we have a

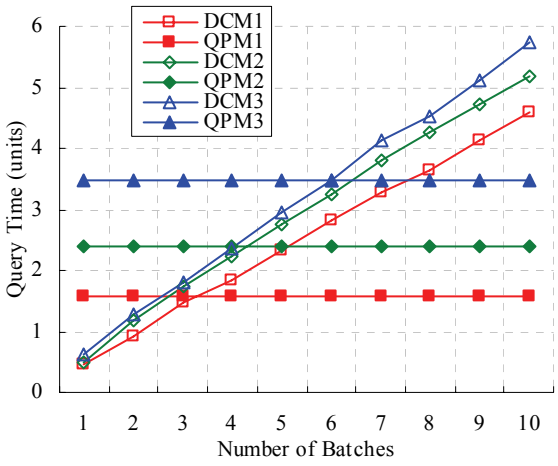

Figure 39. Query Time of Network Topologies for DCM and QPM

limited thread resource. We need to wait for a batch to finish before starting another batch. Thus, the query time t_{DCM} for DCM is calculated by

$$t_{DCM} = \sum_{i=1}^{b} \max_j d(x_1, x_{i,j}), \tag{35}$$

where b is the number of batches, x_1 is the query initiating peer, and $x_{i,j}$ is the j^{th} peer in the i^{th} batch. For QPM, we do not have the same problem because the degree of a peer is very small according to our P2P network topology property (see Figure 26). Thus, the query time t_{QPM} for QPM is calculated by

$$t_{QPM} = \max_p \sum_{i=1}^{|p|-1} d(x_i, x_{i+1}), \tag{36}$$

where x_i is the i^{th} peer in the query routing path p.

Figure 39 shows the query time of the three selected network topologies for DCM and QPM. All DCM curves are approximately linear. When the peer quantity increases, b also increases due to the constant thread

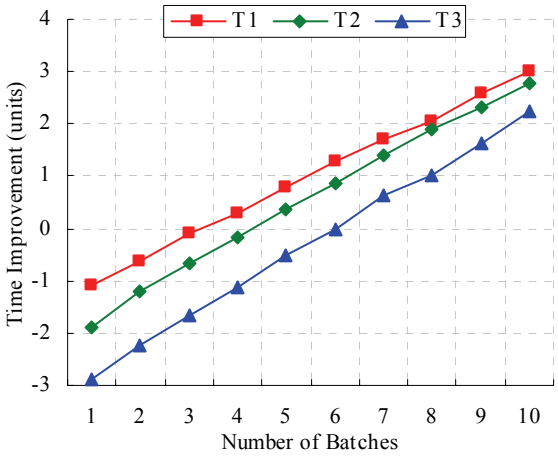

Figure 40. Query Time Improvement of Network Topologies by QPM

resource. Hence, the query time increases because (35) is dependent of *b*. On the other hand, all QPM lines are constant because (36) is independent of *b*. Figure 40 shows the corresponding time improvement that is calculated by the difference between the query time of DCM and QPM. When the peer quantity is small, *b* is small so that the time improvement is negative because the query time is proportional to the path length for QPM. However, the path length is always one for DCM. Thus, DCM is faster than QPM for a small *b*. On the other hand, when the peer quantity increases, *b* increases so that the time improvement increases and becomes positive which shows the good scalability of our proposed hybrid P2P network model.

From the experimental results, we demonstrate that using DCM is un-scalable though it is faster than QPM in a small scaled P2P network. On

the other hand, using QPM is scalable and efficient if the P2P network scale is large. In conclusion, we verify that the query initiating peer has better parallel search by using QPM.

5. Discussion

In this chapter, we have a discussion on both S2S Searching and GAroute for answering the questions and comments given by the reviewers. For S2S Searching, we summarize the related questions and comments by the following four points:

1) Commercial Significance: Some reviewers do not understand the strength of S2S Searching over centralized search engines like Google. Besides circumventing the three shortcomings (centralization of resources used, outdated search results, and no control over information shared by content owners) of centralized search engines, S2S Searching is optimal for community information sharing. One of the problems of Google is that it returns too many search results but only a few of them are useful. This is due to the fact that Google is a universal topic search engine. On the other hand, information retrieval by S2S Searching can deal with specific topics. Sites with similar topics can join together to form a cluster so that more relevant search results are useful with respect to the specific topic. Therefore, S2S Searching competes commercially with Google in terms of its small world property and simplicity of sharing and searching the information like other P2P applications.

2) Global View of Results: Some reviewers think that S2S Searching does not give a global view of the page ranking unlike those centralized search engines. Actually in S2S Searching, the similarity value of a

document already gives the global view on how relevant the document is. However, the priority value of a document only gives a local view on how important the document is. In addition, it is possible for S2S Searching to integrate with other page ranking algorithms. However, we focus more on its P2P model and communication protocol in our research.

3) Ranking Control: Some reviewers ask why we want to allow site owners to control the ranking of their own web pages because this is an invitation to spam. To answer this question, we should understand that advertising is necessary in the real world. Take Google as an example, those advertised relevant pages are also shown at the top or in a separate column. The one, who controls the ranking of the advertised pages, is the search engine administrator. On the other hand, the advertised web page owners cannot control their ranking in order to prevent spam. In S2S Searching, we follow the same idea. The site owner is the administrator of their own search engine so that they should have rights to control the ranking in their own search engine by adjusting the priority value of the documents in the current site. However, the priority value of the documents adjusted in other sites takes no effect to the current site ranking. Therefore, spam can be effectively prevented.

4) Security Issues: Some reviewers criticize that S2S Searching is insecure since it is on top of HTTP where firewalls usually allow the data to pass through. However, S2S Searching targets that it can be easily plugged and played in a website without any administrator's privilege. In addition,

the black list mechanism in S2S Searching can act as a firewall which disallows some requests from bad peers.

For GAroute, we summarize the related questions and comments by the following three points:

1) Future Work on Experiments: Some reviewers criticize that the experiments for measuring the scalability and quality of GAroute are too trivial as we only compare it with the Brute-force Search and Greedy Search. The reason is that GAroute deals with a very special case of Longest Path Problem which is multiple paths in a weighted graph with the cyclic property. Unfortunately, most cited approximation algorithms cannot be easily modified to solve our specific problem because some of them only work in un-weighted graphs and some of them have a fixed destination. Actually, we can still compare GAroute with Ahn's GA in terms of scalability, quality, and network traffic of the paths for the future work. Theoretically, the network traffic of the paths obtained by Ahn's GA should be smaller than GAroute, but the quality of the paths obtained by GAroute should be higher than Ahn's GA, because Ahn's GA only finds the shortest path. Moreover, we can conduct experiments under a dynamic environment to see how the rate of peer joining and leaving affects the overall performance, topology maintenance cost, and the overall traffic overhead including the maintenance.

2) Traffic Reduction and Overhead: Some reviewers misunderstand that the traffic reduction measured in the experiment is for the whole

network. In fact, the reduction is in the query initiating peer only. For the whole network traffic, there is a small overhead as shown in the experimental results. This is due to irrelevant peers in query routing paths which act as intermediate nodes to help propagating queries. The significance of the traffic reduction in the query initiating peer is that we can save a lot of bandwidth when we are searching by the P2P application. Therefore, more bandwidth is available for other applications.

3) Single and Multiple Routes: Some reviewers are unclear to distinguish between single route and multiple routes. Actually, GAroute returns multiple routes, whereas Ahn's GA returns single route. The purpose of GAroute is to find *n* long paths from one given source to any destination peer where the destination peer is not fixed. On the other hand, the purpose of Ahn's GA is to find one shortest-path from one given source to one given destination peer where the destination peer is fixed.

6. Conclusion

In this work, we give a literature review on both pure P2P networks and hybrid P2P networks. Then we propose S2S Searching and GAroute for information retrieval and query routing in P2P networks. Finally, our work is summarized as follows.

For the literature review, we survey on P2P networks and query routing strategies. The introduced pure P2P networks are Napster, Gnutella, Kazaa, BitTorrent, Gnutella2, YouSearch, Discovir, and Freenet. For the query routing strategies, we introduce the firework query model which uses document clustering. And we introduce CAN, pSearch, Chord, Pastry, and Tapestry which use distributed hash tables. We also briefly introduce the research that is related to the P2P network security.

For S2S Searching, we address the three shortcomings (centralization of resources used, outdated search results, and no control over information shared by content owners) of centralized search engines which can be circumvented by distributing search engines over peers which maintain their updated local contents with full control by their owners. Therefore, we propose a pure P2P network together with S2S Searching for Web information retrieval. It helps site owners to turn their websites into autonomous search engines without extra hardware and software cost. We also develop S2S search engines and describe its system architecture.

Our proposed S2S Searching is summarized as follows. We use the

modified vector space model for indexing and matching which is adaptive. We solve the query flooding problem by our proposed query routing algorithm based on distributed registrars. The content summary is a fixed size hash table which stores the importance and confidence level of each word. The relevance level of a site is the average score of the keywords. Queries are routed to those sites with the highest scores and flooded to the adjacent sites with a small probability. The S2S communication protocol depends on the six CGIs which are starting CGI, searching CGI, pinging CGI, joining CGI, leaving CGI, and updating CGI.

Finally, we summarize the experimental results which measure the performance of indexing, performance of matching, performance of S2S Searching, and quality of the content summary. According to these results, we conclude that S2S Searching is scalable (approximate linear) and works well in some large scaled S2S networks.

For GAroute, we address the two shortcomings (high bandwidth consumption and poor semi-parallel search) of the direct connection model which can be circumvented by the query propagation model. Therefore, we propose a hybrid P2P network based on this model and introduce the background of zones and zone managers. We also propose GAroute as a query routing function used in zone managers. By giving the current P2P network topology and relevance level of each peer, GAroute returns a list of query routing paths that cover as many relevant peers as possible. We show how to model this as a Longest Path Problem in a directed graph which is

NP-complete. Due to the efficiency of GA, we obtain high quality approximate solutions in polynomial time.

Our proposed GAroute is summarized as follows. We encode a path as a variable length chromosome. The population is initialized with random chromosomes. The mutation adopts a greedy search which provides fast convergence. The crossover solves the suboptimal problem of mutation. The fission breaks invalid chromosomes produced by crossover down to valid chromosomes. The creation creates non-evolved chromosomes for extra diversity. The selection is based on the fitness (information gain) of chromosomes (paths). The optimization technique called two-phase tail pruning removes dummy paths and cuts the path length to reduce the network traffic and query time.

Finally, we summarize the experimental results which measure the scalability and quality of different searching algorithms for finding query routing paths in different network topologies and peer quantities. According to these results, GAroute achieves both good scalability (approximate linear) and quality (0.95 in 100 peers) in some large scaled P2P network topologies. The two improvements (lower bandwidth consumption and better parallel search) of the query initiating peer in our proposed hybrid P2P network model are also verified by the experimental results. With our model, the query initiating peer network traffic reduces more than 90% for 1,000 peers. The semi-parallel search problem is greatly improved in a large scaled P2P network. We conclude that both our proposed hybrid P2P network model

and GAroute are scalable and work well.

7. Bibliography

[1] A. Bjorklund and T. Husfeldt. Finding a Path of Superlogarithmic Length. SIAM Journal on Computing, Volume 32, Issue 6, Pages 1395–1402, 2003.

[2] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, Pages 329–350, 2001.

[3] A. Silberschatz, P. B. Galvin, and G. Gagne. Operating System Concepts, Seventh Edition. John Wiley and Sons, 2004.

[4] AltaVista Website. http://www.altavista.com

[5] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, Volume 13, Issue 7, Pages 422–426, 1970.

[6] B. Monien. How to Find Long Paths Efficiently. Annals of Discrete Mathematics, Volume 25, Pages 239–254, 1985.

[7] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report, Computer Science Division, U. C. Berkeley, 2001.

[8] BitTorrent Website. http://www.bittorrent.com

[9] BRITE Website. http://www.cs.bu.edu/brite

[10] C. Tang, S. Dwarkadas, and Z. Xu. On Scaling Latent Semantic Indexing for Large Peer-to-Peer Systems. In Proceedings of ACM SIGIR, Pages 112–121, 2004.

[11] C. W. Ahn and R. S. Ramakrishna. A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations. IEEE Transactions on Evolutionary Computation, Volume 6, Issue 6, Pages 566–579, 2002.

[12] C. Yang. Peer-to-Peer Architecture for Content-Based Music Retrieval on Acoustic Data. In Proceedings of the 12th International World Wide Web Conference, Pages 376–383, 2003.

[13] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A Large-Scale Study of the Evolution of Web Pages. In Proceedings of 12th International World Wide Web Conference, Pages 669–678, 2003.

[14] D. R. Karger, R. Motwani, and G. D. S. Ramkumar. On Approximating the Longest Path in a Graph. Algorithmica, Volume 18, Issue 1, Pages 82–98, 1997.

[15] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report, HP, 2002.

[16] F. Cornelli, E. Damiani, S. D. C. Vimercati, S. Paraboschi, and P. Samarati. Choosing Reputable Servents in a P2P Network. In Proceedings of the 11th International World Wide Web Conference, Pages 376–386, 2002.

[17] FastTrack Introduction Website. http://en.wikipedia.org/wiki/FastTrack

[18] Freenet Project Website. http://freenetproject.org

[19] G. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems: Concepts and Design. Addison Wesley, Third Edition, 2001.

[20] Gnutella Website. http://www.gnutella.com

[21] Gnutella2 Website. http://www.gnutella2.com

[22] Google Website. http://www.google.com

[23] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking Up Data in P2P Systems. Communications of the ACM, Volume 46, Issue 2, Pages 43 – 48, 2003.

[24] H. L. Bodlaender. On Linear Time Minor Tests with Depth-First Search. Journal of Algorithms, Volume 14, Issue 1, Pages 1–23, 1993.

[25] Horner Scheme Introduction Website.
http://en.wikipedia.org/wiki/Horner_scheme

[26] I. King, C. H. Ng, and K. C. Sia. Distributed Content-Based Visual Information Retrieval System on Peer-to-Peer Networks. ACM Transactions on Information Systems, Volume 22, Issue 3, Pages 477–501, 2004.

[27] I. King, W. Y. Wong, and T. P. Lau. A Genetic Algorithm for Query Routing in Hybrid Peer-to-Peer Networks. Submitted to IEEE Transactions on Evolutionary Computation for review, 2005.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proceedings of ACM SIGCOMM, Pages 149–160, 2001.

[29] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. Journal of the ACM, Volume 46, Issue 5, Pages 604–632, 1999.

[30] Java Servlet Website. http://java.sun.com/products/servlet

[31] Java Technology Website. http://java.sun.com

[32] Kazaa Website. http://www.kazaa.com

[33] L. D. Whitley and M. D. Vose. Foundations of Genetic Algorithms 3. Morgan Kaufmann, 1995.

[34] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bring Order to the Web. Technical Report, Stanford University, 1998.

[35] LimeWire Website. http://www.limewire.com

[36] M. Bawa, R. J. Bayardo, S. Rajagopalan, and E. J. Shekita. Make it Fresh, Make it Quick – Searching a Network of Personal Webservers. In Proceedings of the 12th International World Wide Web Conference, Pages 577–586, 2003.

[37] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[38] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, Vector Spaces, and Information Retrieval. SIAM Review, Volume 41, Issue 2, Pages 335–362, 1999.

[39] N. Alon, R. Yuster, and U. Zwick. Color-coding. Journal of the ACM, Volume 42, Issue 4, Pages 844–856, 1995.

[40] Napster Website. http://www.napster.com

[41] Peer-to-Peer Introduction Website. http://en.wikipedia.org/wiki/Peer-to-peer

[42] R. J. Bayardo, R. Agrawal, D. Gruhl, and A. Somani. YouServ: A Web Hosting and Content Sharing Tool for the Masses. In Proceedings of 11th International World Wide Web Conference, Pages 345–354, 2002.

[43] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the 12th International World Wide Web Conference, Pages 640–651, 2003.

[44] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science, Volume 41, Issue 6, Pages 391–407, 1990.

[45] S. G. M. Koo, C. S. G. Lee, and K. Kannan. A Genetic-Algorithm-Based Neighbor-Selection Strategy for Hybrid Peer-to-Peer Networks. In Proceedings of the 13th International Conference on Computer Communications and Networks, Pages 469–474, 2004.

[46] 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.

[47] The 12th International World Wide Web Conference Website. http://www.www2003.org

[48] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition. The MIT Press, 2001.

[49] W. Y. Wong. Site-to-Site (S2S) Searching Using the P2P Framework with CGI. In Proceedings of 13th International World Wide Web Conference, Pages 360–361, 2004.

[50] W. Y. Wong and I. King. Site-to-Site (S2S) Searching with Query Routing Using Distributed Registrars. In Proceedings of the Asia Information Retrieval Symposium, Pages 241–244, 2004.

[51] W. Y. Wong, T. P. Lau, and I. King. Information Retrieval in P2P Networks Using Genetic Algorithm. In Proceedings of the 14th International World Wide Web Conference, Pages 922–923, 2005.

[52] XSLT Website. http://www.w3.org/Style/XSL

[53] Y. Liu, B. Zhang, Z. Chen, M. R. Lyu, and W. Y. Ma. Affinity Rank: A New Scheme for Efficient Web Search. In Proceedings of 13th International World Wide Web Conference, Pages 338–339, 2004.

[54] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang. Location-Aware Topology Matching in P2P Systems. In Proceedings of IEEE INFOCOM, Pages 2220–2230, 2004.

[55] Yahoo Website. http://www.yahoo.com

8. Appendix

This appendix contains the supplementary information of S2S search engine and GAroute library.

8.1 S2S Search Engine

In this appendix, we describe the perspectives of site owners and search engine users with some screenshots.

8.1.1 Site Owner Perspective

S2S search engines help site owners to turn their websites into autonomous search engines. To make their websites become the search engines, they need to follow the four steps which are (1) S2S software installation, (2) search engine administration, (3) S2S network management, and (4) search page customization.

1) S2S Software Installation: The first step is to install the S2S software in the website. The S2S software provides a search engine core together with some CGIs for sites to communicate with each other. It also provides the administration pages for site owners to administrate their search engines, and Web interfaces for search engine users to search for Web contents and then display results. The S2S software contains a basic set of HTML files and CGI programs. We use Java Servlet to implement the CGI programs because Java programs are platform-independent. The S2S

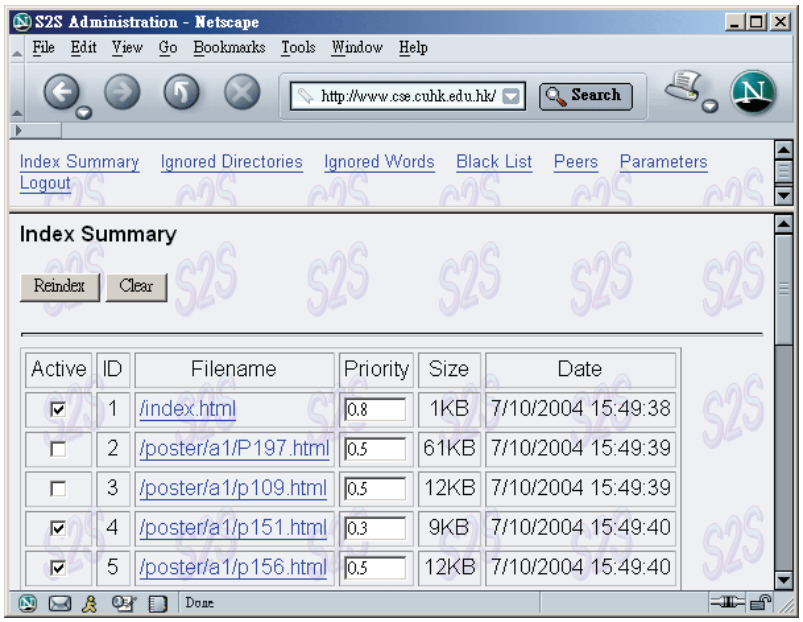

Figure 41. Screenshot of S2S Index Management

software is open source which can be downloaded at "http://www.cse.cuhk.edu.hk/~miplab/s2s". To install the S2S software, site owners are only required to copy those HTML files and Servlet class files to the document directory and Servlet class directory respectively and then configure the Java properties file. The properties file stores system variables such as the system paths, document paths, data paths, username, and password of the administration pages.

2) Search Engine Administration: After installing the S2S software, site owners can administrate their search engines in the administration pages. They can manage their Web contents such as refreshing the local index and content summary, setting the searchable status and priority value of each document for advertising propose (see Figure 41). They can also adjust the

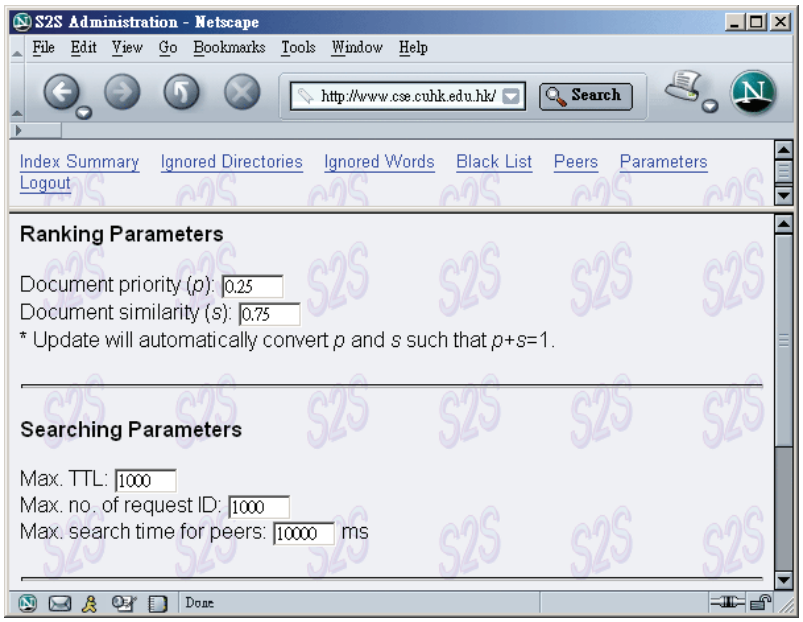

Figure 42. Screenshot of S2S Parameter Management

ranking parameters to customize the ranking equation (see Figure 42).

3) S2S Network Management: After managing local contents, site owners are required to join other sites (see Figure 43). To locate a site, we need to know the corresponding starting URL. For example, if the starting URL is "http://www.s2s.com/servlet/s2s.", then the corresponding URL for the joining CGI is "http://www.s2s.com/servlet/s2s.join" and the corresponding URL for the searching CGI is "http://www.s2s.com/servlet/s2s.search". In the case of Gnutella, we need to know the peer's IP address and port number so that we can locate it. Since different sites join other different sites, the more sites they join, the wider search they perform. In addition, site owners can manage the black list in the administration pages (see Figure 44).

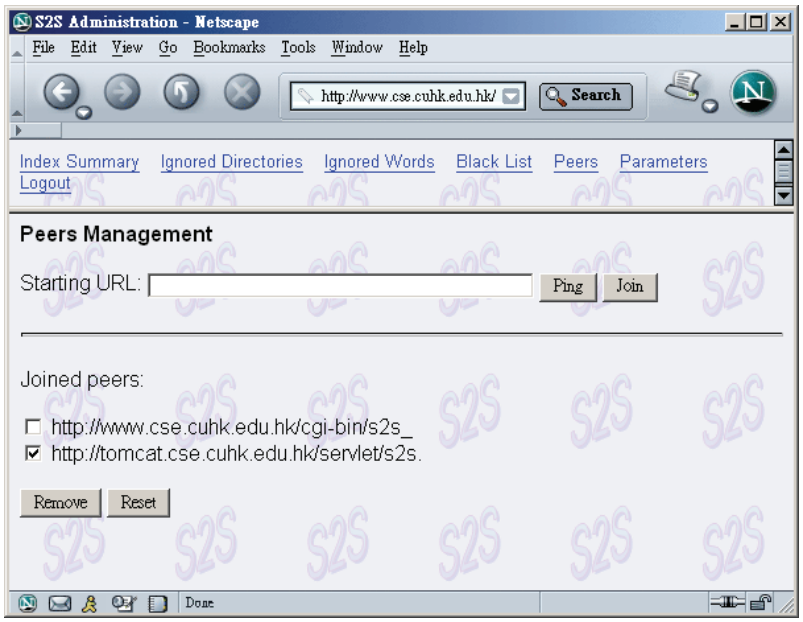

Figure 43. Screenshot of S2S Network Management

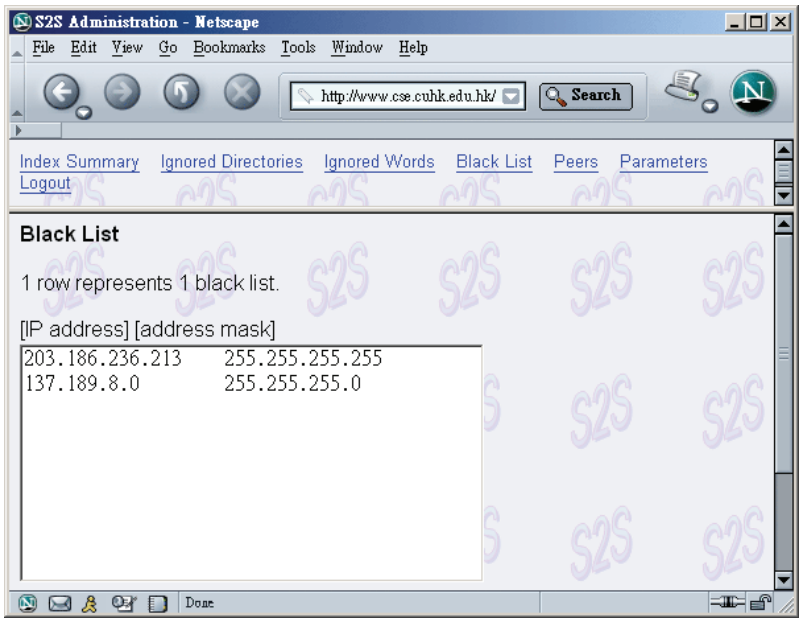

Figure 44. Screenshot of S2S Black List Management

4) Search Page Customization: After joining the S2S network, site

Figure 45. Screenshot of S2S Search Form

owners needs to add a search link or frame to their Web pages. The S2S software provides the default search form so that site owners can directly link to it. They only need to change the HTML form action of the default search form so that it points to the correct starting CGI. They can also customize the appearance of the search form by editing the HTML code. Moreover, they can choose to generate search results in the XML format so that they can write the Extensible Stylesheet Language Transformations (XSLT) [52] code to customize the appearance of search results. After performing the aforementioned four steps, the website becomes a S2S search engine.

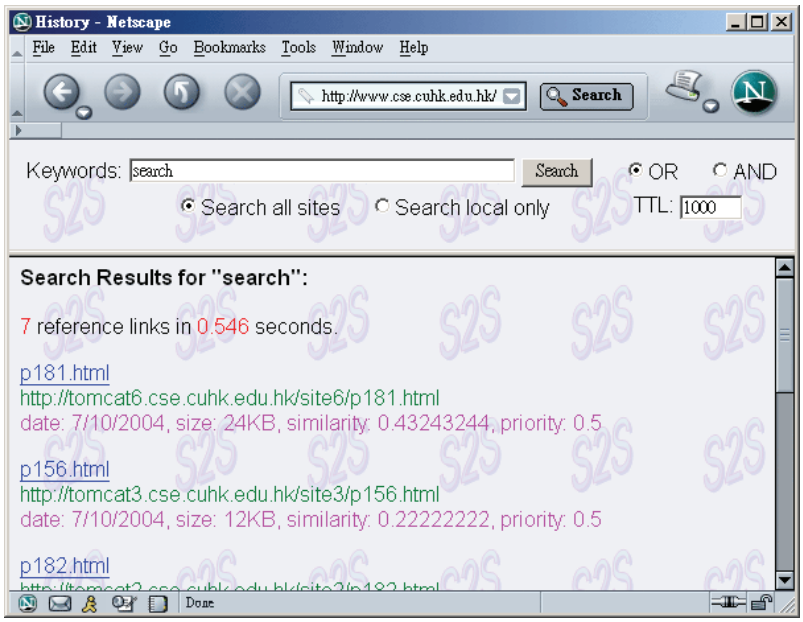

Figure 46. Screenshot of S2S Search Results

8.1.2 Search Engine User Perspective

S2S search engines are distributed in many websites. Therefore, search engine users can go to any website which joins the S2S network to search for the target information they want. After entering a website, users can find a search form (see Figure 45). When they type some keywords, they can choose to search for contents in the local site or in all sites which are in the same S2S network. They can also select the keywords matching type which includes “OR” and “AND”. Moreover, they can specify the TTL value to limit the search space. It is a non-negative integer that defines the maximum level of sites (excluding the local site) that the query request passes through. If they enter the value as zero, then they only search in the local site. The larger TTL value it has, the more results it obtains, but the longer time it

requires. After clicking the search button, the query request is propagated in the S2S network. Within a short period, search results are displayed on the screen. The results include documents' filenames, URLs, dates, sizes, similarities, and priorities. They are sorted by the ranking values which are calculated by the ranking equation (see Figure 46).

8.2 GAroute Library

In this appendix, we describe the GAroute library which is used in zone managers to find optimal query routing paths. It is implemented as the Java package because Java programs are platform-independent. The library is open source which can be downloaded at "http://www.cse.cuhk.edu.hk/~miplab/garoute". It contains only one package (directory) *garoute* which contains six public classes. They are (1) *AdjacencyMatrix*, (2) *Chromosome*, (3) *IdIndex*, (4) *Router*, (5) *ScoreVector*, and (6) *Score*.

1) AdjacencyMatrix: This class is used to create objects for storing the adjacency matrix of the P2P network topology. Table 10 shows the corresponding class summary.

2) Chromosome: This class is used to create objects for storing a chromosome which represents a path. Table 11 shows the corresponding class summary.

3) IdIndex: This class is used to create objects for storing an inverted

Table 10. Class Summary of AdjacencyMatrix

Constructor or method	Parameter → return	Description
AdjacencyMatrix	byte[][] → void	Constructor to set the adjacency matrix
getMatrix	void → byte[][]	Method to get the adjacency matrix
isAdjacent	int, int → boolean	Method to test if the two peers are adjacent
size	void → int	Method to get the size of the adjacency matrix

index of a gene in a chromosome. Table 12 shows the corresponding class summary.

4) Router: This class is to create objects for finding optimal query routing paths. Table 13 shows the corresponding class summary.

5) ScoreVector: This class is to create objects for storing the score vector. Table 14 shows the corresponding class summary.

6) Score: This class is to create objects for storing a score of a peer. Table 15 shows the corresponding class summary.

Table 11. Class Summary of Chromosome

Constructor or method	Parameter → return	Description
Chromosome	int → void	Constructor to set the chromosome with the initial gene
add	int → void	Method to add the gene to the chromosome
get	int → int	Method to get the gene of the chromosome with the specific index
getSortedIdIndex	void → IdIndex[]	Method to get the sorted genes of the chromosome
equals	Chromo-some → boolean	Method to test if the two chromosomes are equal
size	void → int	Method to get the size of the chromosome

Table 12. Class Summary of IdIndex

Constructor or method	Parameter → return	Description
IdIndex	int, int → void	Constructor to store the ID and inverted index of the gene
compareTo	Object → int	Method to compare the two IdIndex objects
getID	void → int	Method to get the ID of the gene
getIndex	void → int	Method to get the inverted index of the gene

Table 13. Class Summary of Router

Constructor or method	Parameter → return	Description
Router	void → void	Constructor to create the router
getInfoGains	void → float[]	Method to get the information gain of each path
getNoOf-Generations	void → int	Method to get the number of generations
getRoutingPaths	void → int[][]	Method to get the optimal query routing paths
setAdjacencyMatrix	Adjacency-Matrix → void	Method to set the adjacency matrix
setCreationRate	float → void	Method to set the creation rate
setCrossover-Proportion	float → void	Method to set the crossover proportion
setMaxGenerations	int → void	Method to set the maximum number of generations
setMinGenerations	int → void	Method to set the minimum number of generations
setMutation-Proportion	float → void	Method to set the mutation proportion
setNoOfGood-Chromosomes	int → void	Method to set the number of good chromosomes for selection
setNoOfPaths	int → void	Method to set the maximum number of paths to be returned
setNoOfPeers	int → void	Method to set the number of peer in

		the network
setPopulationSize	int → void	Method to set the population size
setScoreVector	ScoreVector → void	Method to set the score vector
setSourcePeer	int → void	Method to set the query initiating peer

Table 14. Class Summary of ScoreVector

Constructor or method	**Parameter → return**	**Description**
ScoreVector	float[][] → void	Constructor to set the score vector
getScore	int → float	Method to get the score of the specific peer
getSortedID	void → int[]	Method to get the sorted ID of the peers based on their scores
getVector	void → float[]	Method to get the score vector
size	void → int	Method to get the size of the score vector

Table 15. Class Summary of Score

Constructor or method	**Parameter → return**	**Description**
Score	int, float → void	Constructor to set the score of the peer
compareTo	Object → int	Method to compare the two Score objects
getID	void → int	Method to get the ID of the peer
getScore	void → float	Method to get the score of the peer