To appear in Applied Intelligence, The Intl. Journal of Artificial Intelligence, Neural Networks, and Computer Problem-Solving Techniques, Springer Science+Business Media B.V., 2006.

Using Genetic Algorithms to Reorganize Superpeer Structure in Peer to Peer Networks

Jaymin Kessler
Artificial Intellignece Center
University of Georgia
Athens, Georgia 30602, USA

Khaled Rasheed
Computer Science Department
University of Georgia
Athens, Georgia 30602, USA
khaled@cs.uga.edu

I. Budak Arpinar
Large Scale Distributed Information Systems LAB (LSDIS)
Computer Science Department
University of Georgia
Athens, Georgia 30602, USA
budak@cs.uga.edu

August 11, 2005

Abstract

In this article, we describe a genetic algorithm for optimizing the superpeer structure of semantic peer to peer networks. Peer to peer, also called P2P, networks enable us to search for content or information in a distributed fashion across a large number of peers while providing a level of fault tolerance by preventing disconnecting peers from disrupting the network. We seek to maximize the number of queries answered while minimizing the time in which they are answered. It will be shown that the genetic algorithm (GA) dramatically improves network performance and consistently finds networks better than those found by random search and hill climbing. A comparison will also be made to networks found through exhaustive search, showing that the GA will, for smaller networks, converge on a globally optimal solution.

Keywords Peer to peer, Genetic algorithm, Clustering, Network.

1 Introduction

1.1 The Network Reorganization Problem

The problem of network reorganization varies greatly depending on the network architecture used and the objective function being optimized. For the purpose of this article, it should be assumed that we are using a superpeer network consisting of a large number of terminal peers, each of which is connected to one superpeer. We will refer to a superpeer and all its connected peers as a cluster[1]. These clusters can be connected to other clusters by connecting the superpeers.

When we reorganize a network, we are searching for a design that maximizes (or minimizes) an objective function by modifying certain discrete or continuous parameters. For example, an airplane can be designed

[1] In actuality, a peer may be a member of more than one superpeer cluster. However, we will assume this is not the case for simplicity.

to be more aerodynamic by modifying parameters such as wing curvature, wing length, ratio of wing length to fuselage length, and placement of jets. Once the problem is parameterized, reorganization becomes a simple search problem. In the case of superpeer networks, the number of parameters available to modify is considerably less than those used in many other design problems. We can't change the physical location of a node[2], the bandwidth allocated to that node, or the information it contains. Because superpeers are fixed, we can't change which nodes are superpeers. The only options available are to change the superpeer cluster that a peer belongs to, and to change which superpeers are connected to which other superpeers.

Dialogue is a peer to peer (P2P) network for knowledge discovery designed at the University of Georgia's computer science department. The purpose of the Dialogue network is to automatically discover new knowledge by combining information found on other nodes. Each node contains as its content in the form of html, pdf, ascii text, and other documents. Each node's content is also semantically annotated using recent semantic Web standards such as RDF or OWL. When a node wishes to generate new knowledge, it sends a query out over the network so that other nodes may fill in the missing knowledge with their own knowledge. Essentially, reorganizing the network involves finding a design which maximizes the number of queries successfully answered while minimizing the time required to answer them. This is done by modifying peer clustering and connections between superpeers.

1.1.1 Difficulty of Reorganization

P2P networks are dynamic in nature. Content, network structure, and the peers themselves are constantly changing. In fact, one of the key benefits of the architecture is that unlike the client server model, peers can leave and join without bringing down the network. This also means that our network structure must constantly be updated to reflect changes in the peers connected and user demand. For an algorithm to be effective, it has to take all this into account while still being fast enough to keep up.

Constantly searching for optimal network designs is made even more difficult by the size of the networks. The search space in P2P networks can become prohibitively large. Depending on the day of the week and time, some networks can grow to millions of connected peers. Even if we assume that a small subset of these will remain fixed as superpeers, the problem remains intractable. A relatively small superpeer network consisting of 200 peers and 25 superpeers gives us a search space of $25^{200} * 2^{(\frac{25^2-25}{2})}$ possible networks. The first term reflects the fact that each of the 200 peers is a member of 1 of 25 superpeer clusters. Therefore there are 25^{200} possible clustering arrangements. Multiplying by the second term takes into account the number of possible connections between the superpeers themselves. The best way to understand this is to think of the connections between the N superpeers as being represented by an NxN matrix of bits. Superpeers x and y are connected if matrix[x,y] == 1, and unconnected when matrix[x,y] == 0. It is assumed that if matrix[x,y] == 1, then matrix[y,x] must also be 1. Therefore we do not need the part of the matrix to the right of the diagonal. This means $\frac{(n+1)^2-n+1}{2}$ bits are required to express all possible connections. We can further reduce this number when we consider that the diagonal of the matrix (where $x == y$) is not needed as it only tells us if peer x is connected to itself. Without the matrix diagonal and everything to its right, the number of bits required to express all possible connections between 25 superpeers is $\frac{25^2-25}{2}$. Because each bit can either be 0 or 1, we have $2^{(\frac{25^2-25}{2})}$ different connections.

Furthermore, clustering peers based on similarity of content or queries alone is often insufficient for speeding up search. There is no guarantee that a peer is likely to search for something that is in the

[2]As used here, node is a host that is either a peer or superpeer.

same area of specialization as the knowledge that peer already contains. Clustering based on similarities in queries may work for networks such as Gnutella where the majority of users search for the same or similar things [13], but there is no guarantee that this will be the case in the Dialogue network.

1.2 Current Methods in Network Reorganization

The following sections outline both current methods in network reorganization and related uses of genetic algorithms (GAs). The reader who is unfamiliar with GAs or P2P networks can consult sections two and three respectively for an overview.

1.2.1 Neighbor Selection in Hybrid P2P Networks

Koo et al. [2] discuss applying genetic algorithms to the problem of neighbor selection in hybrid P2P networks. A hybrid P2P network is one that requires a central entity called a tracker to collect statistics and to select neighboring peers from which other peers download content. This is useful because the content distributed on these networks usually consists of very large movie files, which are broken into smaller blocks and downloaded concurrently from multiple peers. These blocks are reassembled by the downloading peer so there is no need to transmit the blocks in order. Once a peer has the entire file it becomes a seeder. Seeders do not always provide blocks to connected peers. They are there to provide blocks that no other currently connected peer has (for example, if all other peers disconnect from the network after downloading the full file). The most well known example of this type of network is Bittorrent.

Every few seconds, a peer will analyze the average rate of download experienced from its neighbors. These and other statistics can be sent back to the tracker and used to more intelligently select neighbors. They formulate the neighbor selection problem as

$$MAX_E \sum_{j=1}^{n} \{G_j(N) - p_j\}|C|$$

$$G_j \approx \frac{1}{|C|} |\bigcup_{i=0}^{N} (c_i \cap \epsilon_{ij}) \cup c_j|$$

subject to $\sum_{j=1}^{N} e_{ij} \leq d_i$ where N is the number of peers, p_j is the proportion of the total content peer j has, $|C|$ is the total number of pieces of content, and $G_j(N)$ can be thought of as the proportion of total content contained in the union of c_j and all c_i where neighbor i and j are connected. ϵ_{ij} is a set such that $\epsilon_{ij} = C$ if neighbors i and j are connected and $\epsilon_{ij} = \emptyset$ if they are not.

Their solution is a binary GA with chromosome length $\frac{N^2}{2} - N$. The bitstring corresponds to the e_{ij} values where 1 means peer i and j are connected and 0 means they are not. Standard one point crossover and bit flip mutation were used, with probabilities of 0.3 and 0.1 respectively. Solutions are evaluated through the above function by plugging the bits encoded by the chromosome into the ϵ_{ij} values. Infeasible solutions are given a fitness of zero.

One of the most noticeable differences between superpeer P2P networks and Bittorrent is that superpeer P2P networks must allow a peer to quickly search for content across large networks. With Bittorrent networks there is no searching for content or query routing because a central tracker makes all decisions as to which blocks are downloaded from which peers. Peers a and c may both be connected to peer b, but it is never the case that a query will be routed from a to c through d. Because of this, Bittorrent networks do not need to be reorganized to route or answer a large number of wildly varying queries, or to cluster

peers into groups. This is exactly what is necessary in the Dialogue network.

1.2.2 An Adaptive P2P Network for Distributed Caching of OLAP Results

Kalnisy et al. [3] discuss the PeerOLAP architecture for supporting On-Line Analytical Processing queries. In this architecture, a large number of clients are connected through an arbitrary P2P network. Each peer contains a cache storing the most useful results. If a query cannot be answered locally by the cache contents of the peer that initiated the query, it is propagated through the network until a peer that has cached the answer is found. An answer may also be constructed by partial results from many peers. The system is also able to reconfigure itself on-the-fly.

The PeerOLAP network can be thought of as a set of peers that access data warehouses and propagate queries. Each peer makes available its computational capabilities and the contents of its local cache. When a peer receives a query, it first tries to answer it locally. If it doesn't have the required data, it may propagate the query to its neighbors. In this way, the network behaves as a distributed virtual cache. The way the peers act together is designed to reduce the overall query cost.

The network structure is reorganized by creating virtual neighborhoods of peers with similar query patterns. If a peer's neighbors are intelligently selected, the peer should have more of its queries successfully answered without having to extend the search to other parts of the network. These neighbors are the only other peers that a peer may directly visit. It is also mentioned that the size of these neighborhoods should be limited in order to prevent flooding of the network and overloading peers with work.

They formulate the network reorganization as a caching problem. Each peer is allowed a limited number of available network resources, considered to be the equivalent of cache cells, and the objects that are cached are the direct connections to other peers. Each connection is assigned a benefit value and the most beneficial connections are selected to be the peer's neighbors. Caching is done through a simple LFU (Less Frequently Used) policy where the least frequently used connections are replaced. The benefit of connecting to a peer to get a chunk of a result is defined as

$$B(c,P) = \frac{T(c, Q \rightarrow P) + \alpha \cdot H(P \rightarrow Q)}{size(c)}$$

where $T()$ is the total cost of computing chunk c and transmitting it from peer Q to P, $H(P \rightarrow Q)$ is the number of hops from P to Q, and $size(c)$ is the size of the chunk (for normalizing). Reconfiguring the set of neighbors every time there is a change in the LFU cache is too expensive. Instead, the system selects the more beneficial connections as neighbors once after every k requests are served.

1.2.3 Reorganization in Network Regions for Optimality and Fairness

Robert Beverly [1] presents an improvement to unstructured overlay P2P networks, such as Gnutella. In unstructured overlays, nodes can directly connect to other willing nodes. There is no even or organized distribution of content, therefore queries are given a limited time to live and submitted to the network by flooding connections, similar to a breadth first search. In structured overlays, node and content identifiers are tightly constrained. These overlays can be thought of as distributed hash tables where content is evenly distributed across peers. A hash function determines which peers will store what content. Because of this, structured overlays are not very well equipped to deal with a highly transient set of peers. It becomes

harder to maintain the content structure required for efficient routing when nodes are joining and leaving at a high rate [11]. Beverly focuses on unstructured networks because they can form organically, without a forced structure, allowing nodes to selfishly connect and affect topology.

Beverly considers an optimal network to be one that is balanced between being fair and ideal. Fairness is defined as a network's ability to prevent nodes from gaining utility without contributing. For example, if a peer issues a large number of queries but does not provide any useful content, it is considered a freerider. It has been estimated that almost 70% of users on the Gnutella P2P network share no files, and nearly 50% of all responses are returned by the top 1% of sharing hosts [12]. These peers should be disconnected from the network. If possible, the network should try to ensure the query success of each node is proportional to the query success it provides to its neighbors. An ideal network is able to successfully answer any query, provided that the information needed to answer that query is available somewhere on the network. The most obvious example of such a network is one in which all peers are directly connected to each other. This network is also the worst you can do in terms of fairness (all peers directly connected even though a majority aren't contributing).

The actual reorganization of the superpeer network is simple. Peer i connects to a randomly chosen superpeer that is not currently a neighbor. After sending and receiving a number of queries, the utility of the connection to each neighbor is evaluated using a utility function. If the connection between neighbors i and j results in a negative utility value, the neighbor is dropped. The following utility function rewards ideal and fair networks while penalizing those with needless communication. The utility function tries to capture query load and search success rate when connected to peer i.

$$u_i(G, M_i, L_i) = \sqrt{M_i} - \alpha L_i, \forall i \in N$$

L_i is a measure of the aggregate load in queries per second received by node i. M_i is the total number of query matches obtained by node i from issuing queries.

1.2.4 Application of GAs to the Bin Packing Problem

The bin packing problem, while not strictly a network reorganization problem, provides a good example of using GAs in grouping. The bin packing problem (BPP) is defined as follows [15] : Given a finite set O of numbers (the object sizes) and two constants B (the bin size) and N (the number of bins), is it possible to 'pack' all the objects into N bins, i.e. does there exist a partition of O into N or less subsets, such that the sum of elements in any of the subsets doesn't exceed B? This NP-complete decision problem naturally gives rise to the following NP-hard optimization problem: what is the best packing, i.e. how many bins are necessary to pack all the objects into (what is the minimum number of subsets in the above mentioned partition)?

Emanuel Falkenauer [4] introduces the application of the Grouping GA (or GGA) to the bin packing problem. The GGA differs from classical GAs in two ways. First, specialized operators are employed that are well suited to the task of grouping. Second, a special encoding scheme is used in order to make the relevant structures of the grouping problem correspond to genes in chromosomes. For example, a standard GA solution to the bin packing problem might encode the following solution where one gene corresponds to one item to be packed:
ADBCEB
This is interpreted to mean the first item is placed in bin A, the second item in bin D, the third and sixth items in bin B, the fourth item in bin C and the fifth item in bin E. In the grouping GA, one gene always

corresponds to one group. From the previously mentioned solution, we know that the following items are placed into bins A through E:
A={1}, B={3,6}, C={4}, D={2} and E={5}
Therefore, our variable length group oriented chromosome may be written as {0}{2,5}{3}{1}{4}. The length of the chromosome is always equal to the number of groups. It should also be noted that by having chromosomes that are group oriented rather than item oriented, we are in a better position to optimize the cost function which heavily depends on the groups.

When optimizing, we want to find solutions that require the fewest bins. However, instead of only focusing on that one criterion, Falkenauer suggests maximizing the following cost function

$$f_{BPP} = \frac{\sum_{i=1..N}(F_i/C)^k}{N}$$

where N is the number of bins used in the solution, F_i the sum of sizes of all the items in bin i, C is the bin capacity and k a constant s.t. $k < 1$. This is a measure not only of the number of bins are required, but also how efficiently each one is being used. This function guides the GA toward a better solution by encouraging better use of the bins.

The operators used in the GGA are crossover, mutation, and inversion. The crossover operator is designed to transfer groups from parents to children. Since all grouping problems have different constraints and different definitions of an invalid population member, Falkenauer outlines the following pattern.

1. Select at random two crossing sites, delimiting the crossing section, in each of the two parents.
2. Inject the contents of the crossing section of the first parent at the first crossing site of the second parent. Recall that the crossover works with the group part of the chromosome, so this means injecting some of the groups from the first parent into the second.
3. Eliminate all items now occurring twice from the groups they were members of in the second parent, so that the old membership of these items gives way to the membership specified by the new injected groups. Consequently, some of the old groups coming from the second parent are altered: they do not contain all the same items anymore, since some of those items had to be eliminated.
4. If necessary, adapt the resulting groups, according to the hard constraints and the cost function to optimize. At this stage, local problem-dependent heuristics can be applied.
5. Apply the points 2. through 4. to the two parents with their roles permuted in order to generate the second child.

Mutation can be divided into three general strategies. Creating a new group, eliminating an existing group, and shuffling a small number of items among groups are all possibilities. Finally, the inversion operator takes good schemas and shortens them. This should help to increase the chances of them being transfered to offspring, ensuring an increased rate of sampling of the above-average ones. In inversion, promising genes (representing groups) are placed close together in a chromosome to increase the chances of them being transfered together to the next generation. Note that group membership does not change during this operation, rather only the order the groups are encoded in changes.

As was previously mentioned, bin packing is similar to the network reorganization problem in that they are both grouping problems. When superpeer networks are reorganized, peers are taken from one

superpeer group and placed in another. In some cases, this grouping is often subject to constraints such as requiring a peer to possess certain content or limiting superpeer clusters to a maximum number of peers. Furthermore, there exists a utility function that allows us to evaluate how good the new grouping arrangement is. In the bin packing problem, we are arranging items into bins in a way similar to arranging peers into clusters. Constraints are placed on the total size of the items placed in each bin. Finally, a number of objective functions have been proposed for evaluating how good of a solution each arrangement is. It seems reasonable that a solution to the bin packing problem may be adapted to solve a P2P network reorganization problem by replacing bins with superpeers, items with peers, changing the problem constraints, and replacing the objective function with one that measures something like query success rate.

1.3 The Dialogue network

Dialogue is a P2P network designed for systematic knowledge discovery, creation, and sharing. The network is arranged into clusters, where each cluster represents a particular area of expertise. When a peer joins the network, the meta-data in its documents (Web pages, PDFs, text files, and others) are extracted to OWL or RDF making them easier to process. RDF, which stands for Resource Description Framework, is a Web standard written in XML. It provides a way to describe content using resources, properties, and objects. OWL, which stands for Web Ontology Language, is considered to be a stronger language than RDF with greater machine interpretability. OWL also has a larger vocabulary and stronger syntax than RDF [8]. Next, representative terms and relationships of the peer's content will be identified. Semantic similarity metrics are then used to find the cluster whose content is most similar to that of the peer wishing to join the network.

Dialogue discovers new knowledge by combining information from connected peers. When we want to find information, a query is sent out over the network. Queries consist of combinations of entities and relations such that either can be a missing piece to be filled in by a peer possessing the proper information. Note that Dialogue system is still under construction.

The following four requirements were defined for the Dialogue network

- Dialogue needs to facilitate a query interface and algebra to interact with, process and seek new knowledge. This algebra facilitates composition as well as querying the knowledge in RDF(S) or OWL.
- Dialogue enables classification of peers according to their local information maps and sends queries only to "knowledgeable" peers for a given query. Thus we use peers to find the right information and their information to find the right peers. This can be accomplished by initially clustering the peers by checking how similar their content is.
- Dialogue employs a systematic set of propagation (i.e., collaboration) techniques among peers to find the right knowledge in a short amount of time. Note that different collaboration schemes can yield or miss the valuable information. Similar to human interaction, new knowledge can be created if the right collaborators contribute the right knowledge at the right time. This capability requires developing some heuristic algorithms similar to social interaction.
- Finally, it should be possible to detect and eliminate conflicts as different peers can contribute contradictory information on the same issue.

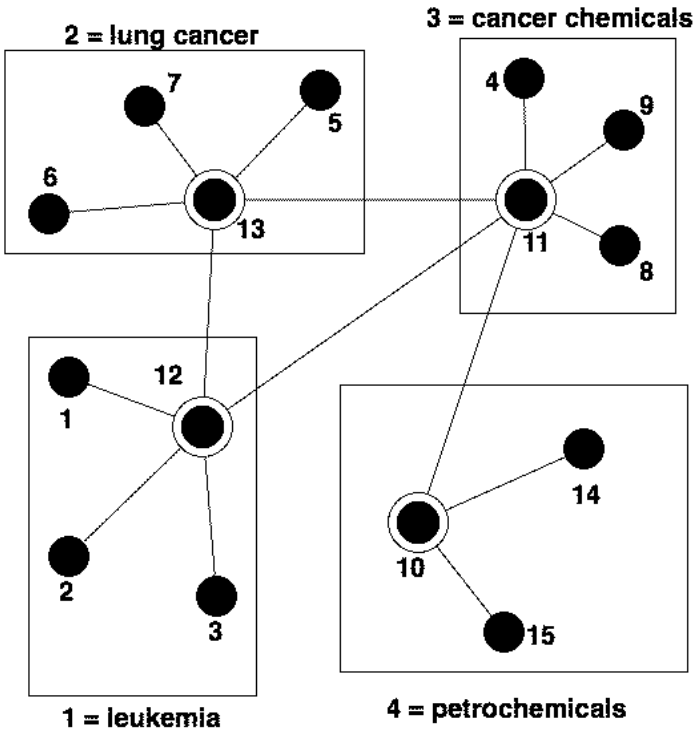

Figure 1: Solving the cause of cancer query

1.4 Knowledge Discovery Example

During the course of a lawsuit investigation, scientists investigated possible causes of the client's cancer. They speculated that the cancer may have been caused by the client's exposure to cancer causing chemicals while at school. At the time Beverly Hills High School, the client's school, had an active oil rig nearby that produced 450 barrels of oil and 400,000 cubic feet of natural gas per day. An investigation discovered high levels of benzene, a chemical known to cause leukemia.

Dialogue would link the client's cancer to the benzene from the oil rig in the following way (see Figure 1). Lets assume that there are four clusters of peers on the network, specializing in leukemia (cluster 1), lung cancer (cluster 2), cancer-causing chemicals (cluster 3), and petrochemicals (cluster 4). There are connections between clusters 3 and 4, clusters 2 and 3, clusters 1 and 3, and clusters 1 and 2. Connections between these clusters often represent some semantic correlations between areas of specialty but this need not always be the case. In this case cancer causing chemicals and petrochemicals are related, lung cancer and cancer causing chemicals are related, leukemia and cancer causing chemicals are related, and leukemia and lung cancer are related.

The query could originate from anywhere in the network but for this example we will assume it starts in the leukemia specialist cluster. We initiate a query, which consists of entities and relations[3], about benzene which is forwarded to other peers in the leukemia cluster. Because none of these peers can successfully answer by filling in any missing entities or relations, the query is forwarded to all the connected clusters (lung cancer and cancer causing chemicals). No peers in the lung cancer cluster can successfully answer so

[3]Example entities could include *benzene* and *cancer*, and the relation could be *causes*.

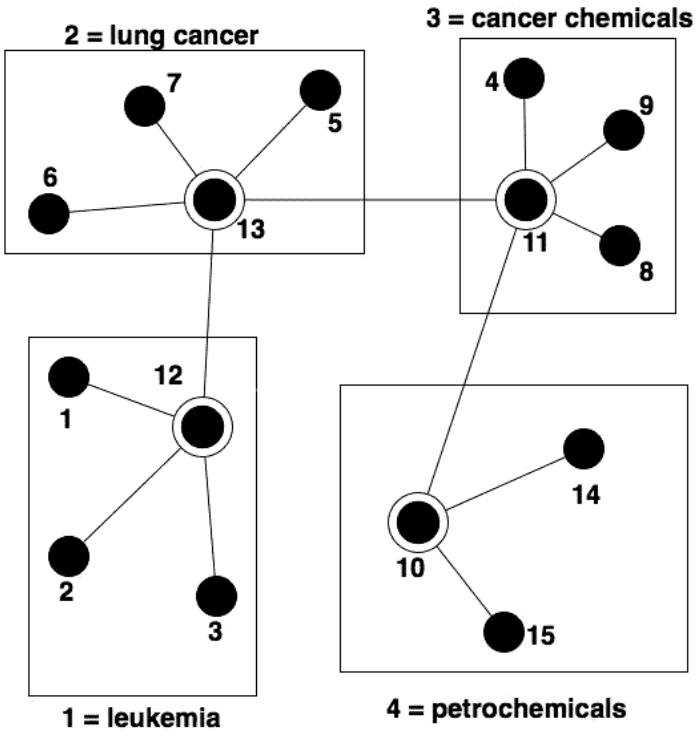

Figure 2: A slower way of solving the cause of cancer query

the query is forwarded to the cancer causing chemicals. If the peers in the cancer causing chemicals cluster can't successfully fill in the missing information in the query, it is forwarded to the petrochemicals cluster where it is successfully answered. The success and efficiency of answering this query depends heavily on the network structure. Therefore, optimal dynamic restructuring of the network is very important. Note that the expected query result in this example should contain newly discovered entities and relations illustrating the cause of the cancer.

In the network shown in Figure 1, the benzene query is answered in a minimum of 2 hops[4] between superpeers. First, the leukemia cluster sends it to the cancer chemicals cluster, and then it is forwarded to the petrochemicals cluster. To show the importance of network reordering, contrast this with the network shown in Figure 2. The same benzene query initiating in the leukemia cluster is now answered in a minimum of 3 hops, because there is no longer a direct link between the leukemia and cancer chemicals clusters. While the difference between 2 and 3 hops does not seem that great, consider what would happen when the number of superpeers gets larger. In a network of 50 superpeers, adjusting one connection can mean the difference between a query being answered in one hop or 49 hops in the worst case.[5]

1.5 Proposed Approach

In this article, we apply a simple steady state GA to the problem of network reorganization. In P2P networks, the connections between peers directly determine what content or information is available to which

[4]Here, a hop refers to sending a query to another superpeer cluster.

[5]Imagine the 50 superpeers are connected such that superpeer 1 is only connected to 2, superpeer 2 is only connected to 3, and so on. A query that originates in superpeer 1 and can only be answered by superpeer 50 is answered in 49 hops. Connecting superpeers 1 and 50 causes the query to be answered in one superpeer hop.

peers. Increasing the number of connections makes more content available but it may flood the system with queries, slowing down the network by increasing traffic and the workload of connected machines [6]. Rather than blindly increasing the number of random connections, our GA should intelligently find an optimal network structure by clustering peers into superpeer groups and connecting superpeers in a way that maximizes the overall query success rate while minimizing the overall time required to answer queries. Each individual population member will use integers to encode information on both peer clustering around a superpeer, and connections between these superpeer clusters. Note that this scheme is used for the sake of simplicity in the experiments using simulation. The objective fitness function will seek to maximize the number of queries answered while minimizing the time in which they are answered.

The rest of this article is organized as follows. Section 2 presents a description of network architecture. We focus on superpeer networks, specifically the Dialogue network. Knowledge encoding, query routing, and the dynamic reordering of network structure is explained. Section 3 begins with an introduction to genetic algorithms. We present a GA for solving the network reorganization problem, including representation, operators, and an objective fitness function. Section 4 details three experiments testing the effectiveness of our GA solution. The GA is compared to random search, a variation of hill climbing, and exhaustive search for networks of various size. Finally, section 5 gives a summary, a review of our contribution, and possible directions for future work.

2 Network Architecture

This section provides an introduction to network architecture, specifically focusing on the superpeer architecture used in the Dialogue network. The Alliance for Telecommunications Industry Solutions [9] gives two definitions of network architecture.

- The design principles, physical configuration, functional organization, operational procedures, and data formats used as the basis for the design, construction, modification, and operation of a communications network.
- The structure of an existing communications network, including the physical configuration, facilities, operational structure, operational procedures, and the data formats in use.

This is to be differentiated from the underlying protocol which specifies the details of how data passes between two communicating applications[6]. As of 2004 one of the most commonly used architectures is client-server, partially because the client-server architecture is the basis of the World Wide Web. Client-server can be described as follows [10]. The central entity (server) is a program running on a host at a specific IP address and port and offers a service such as telnet, ftp, or serving Web pages (http). The client initiates a connection request, negotiates with the server, and eventually connects. The server is usually able to handle multiple clients being connected at once. The servers that are connected to the clients are not connected to each other, and do not even have any knowledge of each other. This provides a layer of security. However, since the server is the central entity, if the server dies the whole network disappears. Also, these networks are not very scalable as a sufficiently large number of clients connecting to a server will usually make the server unavailable.

The rest of this section will focus on a radically different network architecture called P2P which tries to address some scalability and fault tolerance problems that client-server has.

[6]These protocols include TCP/IP and UDP, to name some commonly used ones.

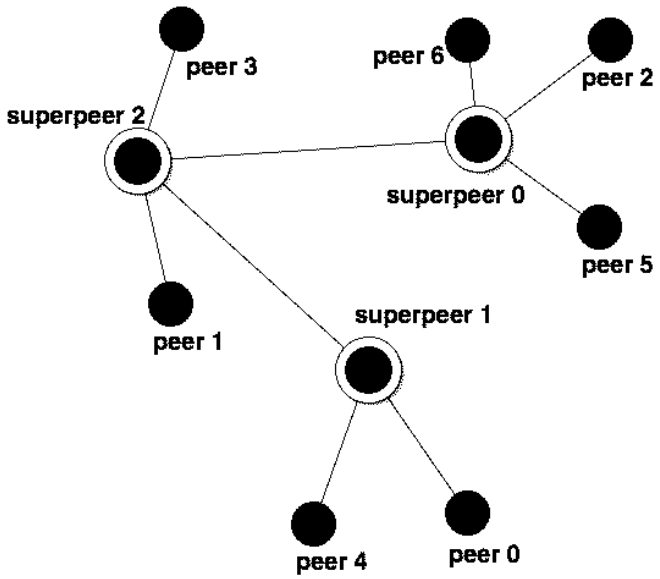

Figure 3: A sample superpeer network with three superpeers.

2.1 Superpeer Architecture

P2P is a network architecture in which there is no centralized coordination. Peers are fully autonomous and able to join and leave the network at will, with little effect on performance. These peers act as both client and server simultaneously while providing content to the rest of the network. P2P networks provide a number of improvements over the traditional client-server model, but also introduce a number of challenges. Having clients directly exchange files, information or knowledge rather than go through a centralized server obviously increases efficiency, but a level of security is sacrificed as peers must now directly connect to each other. Having peers act as both client and server at the same time is a good way to share computing resources although load balancing becomes a problem. Having the peers provide content themselves provides a tremendous amount of available information, although it becomes difficult to search as the size of the network grows. Finally, the network exhibits a level of fault tolerance coming from the fact that much of the content on the network is redundant [5]. Because of this, there is a smaller chance of a particular piece of content becoming completely unavailable when a peer leaves the network. However, there are no guarantees that content available one minute will be available the next.

One of the most serious problems with P2P networks occurs when the network traffic saturates slower links, such as peers connected via dial-up modem. Routing queries through these peers can slow network operation to a crawl. One commonly used improvement intended to fix this problem is the superpeer architecture. In a superpeer network, more powerful machines with faster connections are chosen to be superpeers. These superpeers receive (or generate) queries, try to answer them, and then forward them to all connected peers (see Figure 3). The peers try to answer queries but never forward them to other hosts.

2.2 Queries and Knowledge Encoding

The GA described in this article was initially designed as part of the Dialogue semantic P2P network. Given knowledge of what queries were asked, which queries were answered, and in how much time, the GA should find an optimal superpeer structure that maximizes the number of queries asked while minimizing

the time they are answered in. However, at the time this was written functionality supporting the asking and answering of queries was not complete. A more simple scheme had to be constructed in order to create the simulation and test the GA.

In the actual Dialogue network, clients contain information in the form of papers, Web pages, and other documents. This information is then stored in a form that can be understood by the Dialogue client, such as RDF or OWL, and used to answer incoming queries. The queries themselves take the form of combinations of entities and relations. Specific clients also have specific areas of expertise that may be used in the initial grouping of peers into superpeer clusters. To summarize, any system adequately simulating the knowledge system used in the Dialogue network must have the following

- a way of encoding queries that doesn't depend on currently incomplete semantic processing functionality.
- a way of determining if a host can answer a query.
- a way of expressing related pieces of knowledge belonging to the same area of expertise.
- a way of generating related pieces of knowledge in a specific domain without having to code all the connections between them.

To simplify the simulation, both pieces of knowledge and queries are represented as integers. Specific areas of expertise are represented as groups of ten integers, 0 — 9, 10 — 19, and 20 — 29 for example. When an instance of a peer is allocated, the knowledge manager class randomly chooses an area of expertise, lets say 20 —29, and then generates ten integers in that range representing pieces of knowledge. When a new query arrives, the knowledge manager tries to answer it by matching it to a piece of knowledge. If the host can't answer the query, NO_ANSWER is returned.

2.3 Query Routing

Routing begins when a host receives a QUERY_SUBMIT event. Because each host's area of expertise is randomly generated, it is possible for the query originator to answer the query itself. However, for this simulation we assume that in a real situation there would be no reason to propagate queries that the originating host can answer and therefore we always assume that the originating host cannot answer the query itself.

The host first forwards the query to the superpeer it is connected to which then attempts to answer the query. If the superpeer is unable to answer, it is then broadcast to all peers that are currently connected to that superpeer. The superpeer then waits for all hosts to respond, either with a successful answer or NO_ANSWER. If one or more successful answers are received, the first is returned to the query originator and the query is considered answered. The simulation only considers the first successful answer in determining the time in which a query is answered.

If no connected hosts return a successful answer, the superpeer will broadcast the query to all connected superpeers and the process continues. Those superpeers will then forward the query to all peers connected to them, wait for all peers to answer, and then broadcast to other connected superpeers if the query has not already been answered.

There are three conditions that can end query propagation. First, queries have a time to live field that is decremented every time it is forwarded to another superpeer. This is a system parameter and can be set to any number of hops. When the time to live becomes zero, the query dies and it is no longer forwarded. Second, a query contains a record of all the superpeers that it has been previously sent to. If a superpeer receives no successful answers from connected peers and is not connected to any other superpeers that haven't previously seen the query, the query dies. Finally, a successful answer to a query causes all other responses to be ignored. Notice that a peer in one cluster answering a query does not guarantee that query propagation will end in other clusters. The query is allowed to run its course through other clusters and all returned answers are ignored.

Because broadcasting and answering queries is a potential bottleneck in P2P networks, techniques to reduce this bottleneck can be employed. Kunwadee Sripanidkulchai [13] shows that caching query results, similar to the way that Web pages can be cached for faster fetching, helps to reduce both the time in which queries are answered and also the amount of traffic on the network. Because peers will always greatly outnumber superpeers, limiting the number of peers that queries are broadcast to will also considerably improve network efficiency. In the Gnutella network, this is achieved through the use of bloom filters. A bloom filter is a data structure for succinctly representing a set of filenames. The list is hashed, compressed, and sent to the superpeer. An interesting property of bloom filters is that when they are used to test if a particular peer contains a file, they give a very small number of false positives and absolutely no false negatives [14]. Although peers connected to the Dialogue network will be discovering knowledge rather than searching file names, the network can still greatly benefit from peers periodically sending a summary of the knowledge they contain to their superpeers for indexing.

2.4 Initial Clustering and Nightly Reorganization

New hosts wishing to join the network will initially be clustered based on the semantic content of their public documents. They will be assigned a superpeer cluster to join based on how similar their information is to that of the other peers in that superpeer cluster. The GA is in no way involved in this initial clustering. For example, if a user is an expert in GAs, her documents are likely to contain a large amount of information in that area. Therefore the user would most likely be placed into a superpeer cluster with other GA experts. The simulation described in this article randomly places all peers into initial superpeer clusters rather than clustering based on semantic content.

The GA will most likely be run nightly, during off peak hours when network usage is at a minimum. Depending on daily query volume, either all queries or ones that were not satisfactorily answered are logged during the day. At night, the GA reconfigures the network structure based on a population of designs and runs all logged queries again. It then finds the network structure that maximizes the number of queries successfully answered while minimizing the time taken to answer them.

3 Genetic Algorithm

3.1 Overview of Genetic Algorithms

In this section we describe a GA to solve the network reorganization problem. GAs are evolutionary search techniques based on principles of biological evolution. An initial population of randomly generated solutions to some problem is maintained, each with a scalar fitness value. Fitness is a value that is computed for each population member and used to evaluate how good a solution that member is. In the example of the

Dialogue network, we could use the number of queries answered along with other information to compute the fitness for each population member (network)[7]. From this population, two members are selected to be parents, with higher fitness population members having a greater chance of being selected. These parents then produce one or more offspring by means of various operators such as crossover and mutation. In generational GAs this process is repeated until a new generation is created. This new generation is then considered to be the current generation and the old generation is discarded. See Figure 4 for a common algorithm for generational GAs. In steady state GAs, like the one described in this article, the new individuals are directly inserted into the current population replacing weaker members, rather than being placed in a new population. This process is outlined in Figure 5. The population's average fitness tends to increase over time leading to a higher average fitness and populations of better solutions. The GA is allowed to run until some terminal condition becomes true, such as the loss of diversity.

The remainder of this section describes the general approach the GA takes in terms of its use in the Dialogue network. As mentioned above, we begin with a randomly generated population. Since each individual population member encodes a network design, all initial clusters and connections are random. We then evaluate each population member and assess its fitness. To evaluate a population member, we configure the actual Dialogue network connections and clusters according to the design encoded by that member. We then test the network by submitting queries. The population member's fitness is directly related to how successful the particular configuration is in answering those queries. The specific function used to evaluate fitness will be given in a later section. For now, just note that it takes into account the ratio of queries answered to queries asked, and the total time taken to answer those queries. After all population members have been assigned fitness values, two are chosen to be parents. The higher the fitness, the more likely a population member is to be chosen as a parent. Note that low fitness members are not excluded from being parents, but the probability of them being chosen is low. The two parents then produce a child through operators such as crossover and mutation, which are described in a later section. We want that child to be added into the population, but the population size must remain fixed. This means a current population member must be removed to make room for the new child. We randomly choose a member from the worst n% of the population, remove it, and replace it with the new child. We then choose another two parents and the process is repeated until a terminal condition is reached such as the loss of diversity.

The hope is that two parents who are high in fitness can produce a child that is of equal or higher fitness. To illustrate this process, assume the following bitstrings are parents that encode information about connections between superpeers[8].
11100000
00001111
The more 1's in the string, the more superpeers are connected. Through the use of genetic operators, these parents could produce an offspring such as 11101111 which is a design connecting more superpeers than either parents. This increase in superpeer connections could lead to an increase in the query success rate. Two high fitness parents do not always produce a high fitness child. Parents such as
11110000
00001111
could easily produce a child like 00000000. However, overall the high fitness children produced replace the lower fitness members and the population's average fitness generally tends to increase over time.

[7] It would be overly simplistic to use only the number of queries answered as it ignores the number of queries asked and the time it took to answer those queries.

[8] The actual encoding of superpeer connections used in the GA is a little more complex, but this simplified encoding will work for the example.

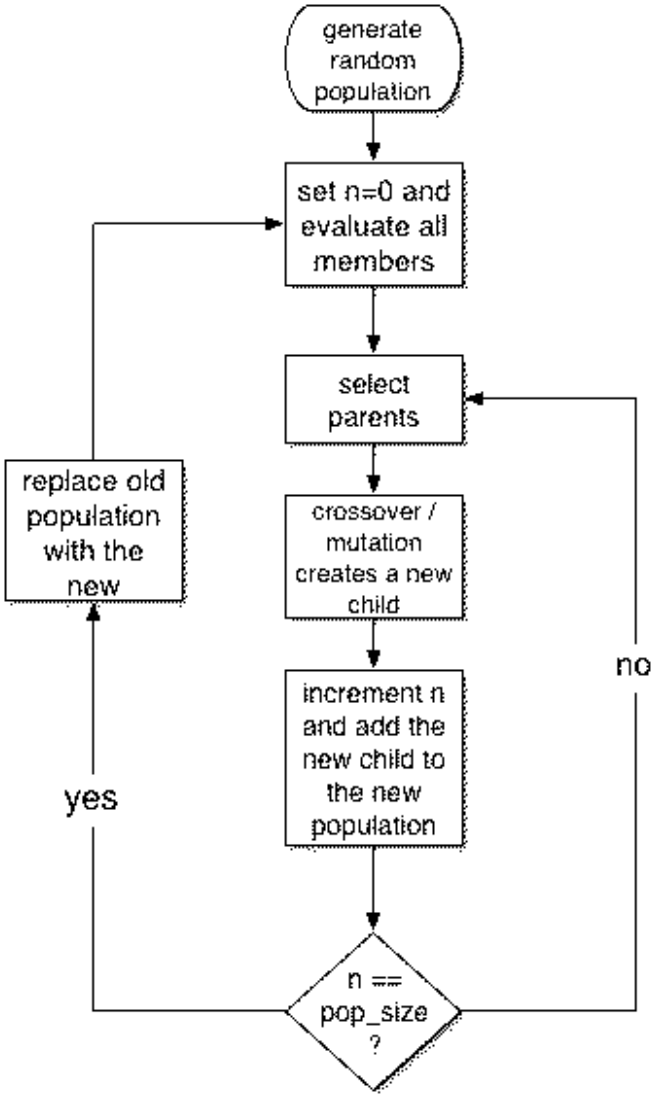

Figure 4: A common algorithm for generational GAs.

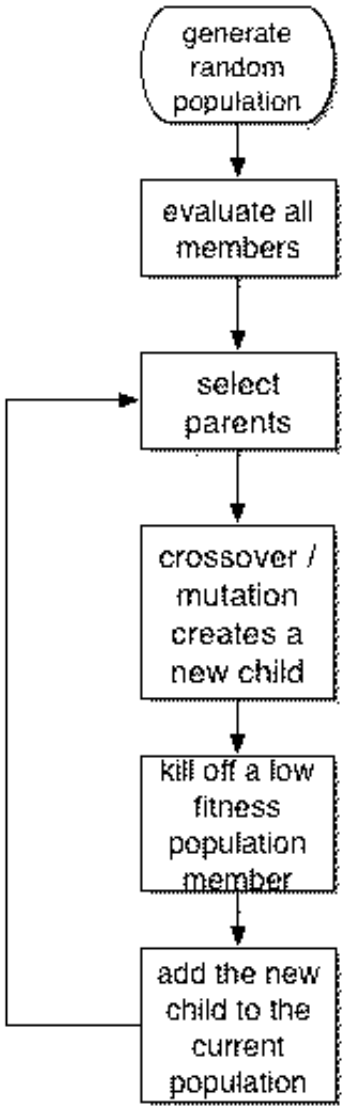

Figure 5: A common algorithm for steady state GAs.

Figure 6: Connections between hosts 0 and 2, 1 and 3, 2 and 4. **x** indicates an unused cell.

3.2 Representation

Each population member is represented by two separate vectors of integers. The first vector describes which peers belong to which superpeer clusters and the second describes the connections between the superpeers. Obviously, the total length of each population member directly depends on the number of peers and superpeers currently connected to the network. Chromosome length is defined as $h + \frac{n^2-n}{2}$ where h is the number of connected hosts that are not superpeers and n is the number of superpeers. Each integer in the vector represents an ordinal index into a superpeer lookup table and can range from 0 to $MAX\_SUPERPEERS - 1$.

3.2.1 Clusters

The first MAX_PEERS positions in the chromosome represent clustering information. Each of the MAX_PEERS positions represent a non-superpeer host currently connected to the network. The integer stored in that position represents the superpeer that the non-superpeer host is connected to, or rather the table index of the superpeer. For example if we have five peers and two superpeers, we could theoretically have the following
(1,1,0,1,0)
meaning that host 0 is connected to superpeer 1, host 1 is connected to superpeer 1, host 2 is connected to superpeer 0, host 3 is connected to superpeer 1, and host 4 is connected to superpeer 0. This essentially groups the peers into clusters by what superpeer they are connected to.

3.2.2 Inter-superpeer Connections

The next $\frac{n^2-n}{2}$ genes represent connections between the superpeers themselves where n is the number of superpeers. If you think of this part of the GA as an NxN matrix such as the one in Figure 6, a connection between superpeers 3 and 1 can be established by setting matrix[3][1] = 1. Likewise, matrix[3][1] = 0 indicates that no connection is present between superpeers 3 and 1. You will notice an unnecessary amount of redundancy in this encoding scheme. Because all connections are bidirectional, a connection between superpeers 3 and 1 must surely imply there is also one between 1 and 3. If you look back at the matrix, you will see that we only really need the part of the matrix to the left of the diagonal as the rest expresses redundant or unnecessary information. The diagonal itself expresses hosts connected to themselves while everything to the right just repeats what is to the left of the diagonal (with x and y values swapped). This is why instead of an NxN matrix, we only need $\frac{n^2-n}{2}$ bits to represent connections. This $\frac{n^2-n}{2}$ bit string allows us to encode $2^{\frac{n^2-n}{2}}$ different sets of superpeer connections.

3.3 Operators

The basic operators used in this GA are selection, crossover, mutation, and replacement. Selection is the process of choosing population members to be parents. Crossover, a recombination operator, is responsible for creating a new individual from two parents. Mutation introduces further variation in search directions by randomly incrementing or decrementing the value of individual genes. Replacement takes the newly formed population member and reinserts it back into the population by killing off a weaker existing member. No diversity maintenance operators were used in our GA. A diversity maintenance operator tries to limit population members from becoming too similar. In steady state GAs we can choose a population member to kill off and replace not only by fitness, but also by how similar it is to the rest of the population. In generational GAs we can weight a population member's fitness value with a measure of how similar it is to the rest of the population.

3.3.1 Selection

Two selection methods were used in choosing parents. Earlier experiments used rank based selection, a method in which population members are assigned a rank based on their fitness. Members with a better rank have a greater chance of being selected. For example if population member 1 has a fitness of 13, member 2 has a fitness of 128, and member 3 has a fitness of 42, ranks 1, 2, and 3 will be awarded to population members 2, 3, and 1 respectively. One way to compute the probability of selection is with the function $prob(rank) = q - (rank - 1)r$ where $r = q/(pop\_size - 1)$. The parameter q controls selection pressure[9] and ranges between $[\frac{1}{pop\_size}, \frac{2}{pop\_size}]$. Selection by rank can in some cases be an improvement over roulette wheel selection. In roulette wheel selection a population member's probability of being selected is the ratio of its fitness to the total population fitness. For example, if the total population fitness is 50.0 then a population member with a fitness of 25.0 has a probability of 0.5 of being selected. As you can see, rank based selection can keep excessively high fitness members from completely dominating selection and limiting diversity.

Table 1: Comparing probability of selection in rank based and roulette wheel methods with $q = 0.66$ and $r = 0.33$.

Fitness	Rank	P_rank	P_roulette
10	3	0	0.009
1000	1	0.66	0.9
100	2	0.33	0.09

An alternative approach, known as tournament selection, seemed to help the GA to converge more quickly. In tournament selection, n population members are randomly selected to compete in a tournament. Of those n members, the one with the best fitness is chosen to be a parent. This process is repeated one more time to find the second parent. Tournament selection seems to provide two main benefits over rank based selection. First, similar to simulated annealing, the boltzmann selection distribution can be used to occasionally select a tournament winner with a lesser fitness. Over time, the probability of a lesser fitness individual winning a tournament can be decreased as we begin to converge on a global maximum (if we are maximizing fitness). Second, tournament selection allows us to better control selection pressure by regulating the tournament size. For example, if the tournament size is two, a population member only

[9]Selection pressure can be thought of as the ratio of the best individual's selection probability to the average selection probability of all individuals in the population.

needs to have a fitness greater than the other competitor in the tournament. By increasing the tournament size to four, a population member must now have a fitness that is better than the other three competitors to win.

3.3.2 Crossover

For the given problem, the most useful crossover operators seem to be one-point and uniform. In one-point crossover, a random locus is chosen between 0 and the length of the chromosome. The first child is created by concatenating the first part of the first parent and the second part of the second parent. Similarly, the second child is the concatenation of the first part of parent two and the second part of parent one. This operator seems to perform well because rather than mathematically calculating a new gene based on two good genes, some sequences present in the parents are preserved in the children. Uniform crossover is a generalization of one-point crossover. In uniform crossover, you step through each gene. With a probability of P, where P is usually 0.5, either the gene from parent 1 or parent 2 is passed on to the new child. This operator tends to break up strings of genes (called building blocks) that may have contributed to a good solution. However, the ability to combine features from both parents regardless of their relative locations often outweighs this disadvantage [7].

3.3.3 Mutation

The most useful mutation operator for this problem seems to be uniform mutation. In uniform mutation, a gene's current value is replaced with a random value in the gene's range. It may sound counterintuitive that a random search would consistently outperform the more intelligent mutation methods such as non-uniform mutation but this is only because uniform mutation is the only one that doesn't place unnecessary and undesirable restrictions on the search space at any time during the search.

3.3.4 Replacement

In generational GAs, two parents are selected from the current generation to create a child. This process is repeated n times (where n is the population size) until the children form a whole new generation. Only after this new generation is completely created are new parents selected from it to continue the process. Notice that the newly created child is placed in the next generation, rather than in the current one. This is not the case with steady state GAs, like the one described in this article. In a steady state GA, a newly created child is inserted immediately into the current population. If we wish to keep the population size constant, this means we must choose a population member to replace with the new child. This, for obvious reasons, is called replacement. There are many methods of replacement ranging from replacing the worst population member, to replacing a member randomly chosen from the worst $n\%$ of population members, to replacing the worst population member that is the most similar to the child we want to insert. The last method mentioned is intended to maintain diversity by preventing the population from containing too many similar members. This GA uses the second method, randomly choosing a current member to replace from the worst 10%.

3.4 Fitness

Essentially, network reorganization is a multiobjective problem. We want to maximize the number of queries answered while minimizing the time taken to answer them. Many solutions have been proposed to help with the problem of multiobjective optimization, one of the most commonly used being finding the Pareto front points. A point is on the Pareto front if there are no points that are better than it in *all*

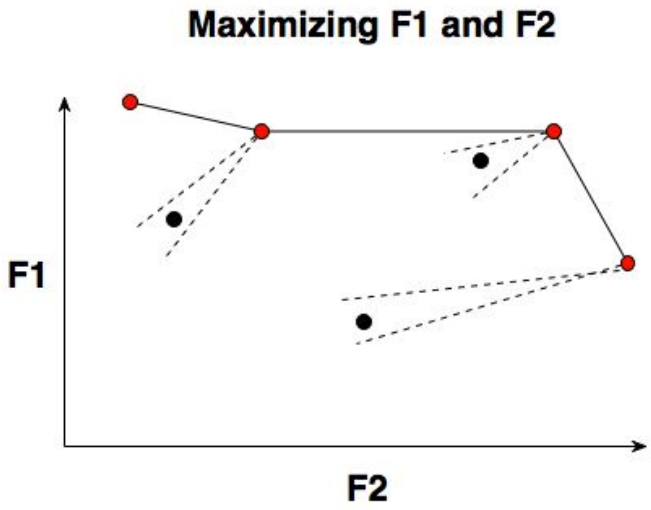

Figure 7: Pareto front points are in red and dominated points are between dotted lines.

objectives. To clarify, a point may still be on the Pareto front if another point is better than it in some objectives. But that point is no longer in the Pareto front if another point is found that is better than it in all objectives. Figure 7 demonstrates a sample Pareto front. Pareto points are in red and dotted lines demonstrate some points that are dominated by Pareto points. To simplify things, a 2 dimensional example will be used where the first parameter f1 is represented by the y axis and the second parameter f2 is represented by the x axis. To find the Pareto front, we first must decide if we want to minimize or maximize each of the parameters f1 and f2. Then we have to find the points that are not dominated by any others. If we want to maximize f1 along the y axis, it dominates all points below it. If we want to minimize f1 along the y axis, it dominates all points above it. If we want to maximize f2 along the x axis, it dominates all points to the left of it. If we want to minimize f2 along the x axis, it dominates all points to the right of it. So putting it all together, if we want to maximize f1 and f2, a point p is on the Pareto front if it is not dominated by any other points (there exist no points that the Pareto point is to the left of and below).

A fundamental drawback of the Pareto front method is that an end user is usually required to interpret the graph and evaluate which solution is the best. Whether the Pareto front is analyzed by humans or computer, it is a very time consuming operation. To speed up the GA, a weighted sum approach can be taken where the two criteria for evaluation are combined in one objective function. This is a classical approach to multiobjective optimization and can greatly simplify the evaluation of points. Specifically, the following formula was used, taking into account the number of queries asked, the number of queries successfully answered, the time they were answered in, and the total time the simulation ran for.

$$\frac{total\_query\_time}{queries\_answered + 0.001} + (1000.0 * \frac{queries\_asked}{queries\_answered + 0.001})$$

Notice that we are trying to minimize the fitness, rather than maximize. If we ignore the first part, we are left with the ratio of queries asked to queries answered, multiplied by a factor of 1000, with 0.001 added to the denominator to prevent division by zero. As the number of queries answered approaches zero, this term becomes excessively large. If two networks have a similar percent of queries successfully answered, we need to favor the network that answered the queries the fastest. The first term simply calculates the average time each query was answered in.

To appear in Applied Intelligence, The Intl. Journal of Artificial Intelligence, Neural Networks, and Computer Problem-Solving Techniques, Springer Science+Business Media B.V., 2006.

4 Experiments and Results

Here we present three experiments to evaluate the effectiveness of the GA when applied to the network reorganization problem. The networks found by the GA are compared to networks found by random search and hill climbing, and for smaller networks the GA is also compared to exhaustive search.

4.1 Experiment 1

The first experiment compares the results found by running the GA on a small network of 12 peers and 3 superpeers to those found by random search, a variation on hill climbing, and exhaustive search. We hope to show that the GA is doing more than randomly visiting points in the search space, and can outperform a heuristic method such as hill climbing. Hill climbing begins with a randomly generated network. At each iteration we either change a peer's superpeer cluster or add/remove a connection between superpeers. If the resulting network has a higher fitness than the previous one, it becomes the current network. Random search simply generates one new network consisting of random clusters and superpeer connections at each iteration and stores the best network seen so far.

All networks were tested on 120 randomly generated queries, which were evenly distributed across all peers for submission to the network. In all experiments, a set of random costs representing the time needed to transmit a query between two hosts is generated. The queries were generated at the beginning of the simulation and used to test all network configurations throughout the experiment. The GA, consisting of 50 population members, was allowed to evolve over 5000 iterations. To keep the comparison between the GA and other methods fair, a new random network is generated for hill climbing and random search every time we create a new GA population member. Tournament selection was used with a tournament size of 2. The entire experiment was repeated 30 times.

4.1.1 Results

Table 2 compares the fitnesses of the networks found by the GA, random search, and hill climbing in 10 out of the 30 runs. The GA was the most consistent, producing networks with the highest fitness in every run. Hill climbing was able to produce 7 networks with the same fitness found by the GA, while the other 23 runs produced comparatively low fitness networks. As expected, random search performed the worst, with no runs producing networks with a fitness as high as those found by the GA.

Figures 8 and 9 represent a sample network before and after evolution by the GA. The large circles represent superpeers 0, 1, and 2. The smaller circles labeled P0 through P11 represent peers. Near each peer is a number representing a peer's knowledge specialty. For example, peers 0, 5 and 11 all specialize in knowledge area 2. These specialties do not change as the network changes. The edges connecting peers to their superpeer clusters represent connections, and are assigned a number based on the cost of sending information between a peer and superpeer. For example, in Figure 8 there is a cost of 40 time units when sending data between superpeer 0 and peer 0. These time units do not affect any real world quantity such as bandwidth or physical distance. They are only used to simulate the cost of sending data to different peers.

The number of factors affecting network performance make it difficult to understand how the network in Figure 9 is any better than 8. To fully understand, you would have to take into account every query submitted, where that query originated from, the cost of searching each peer, and each peer's collection of information. From looking at the figures, it may seem like there is no logical reason for one network

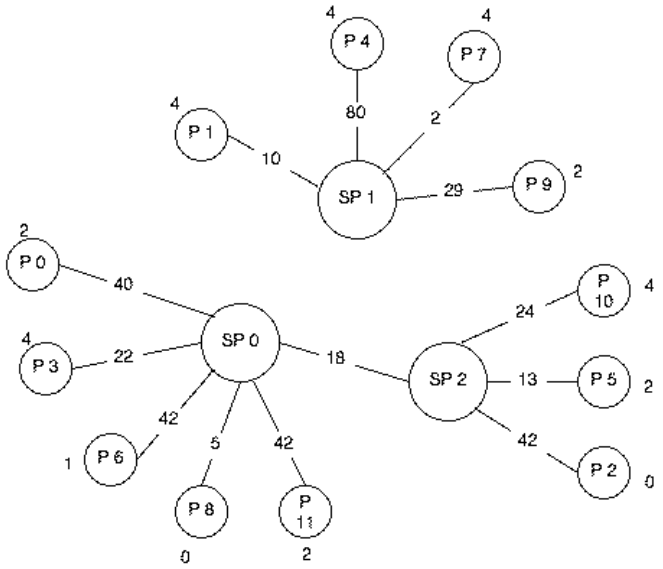

Figure 8: The highest fitness network in the population before evolution.

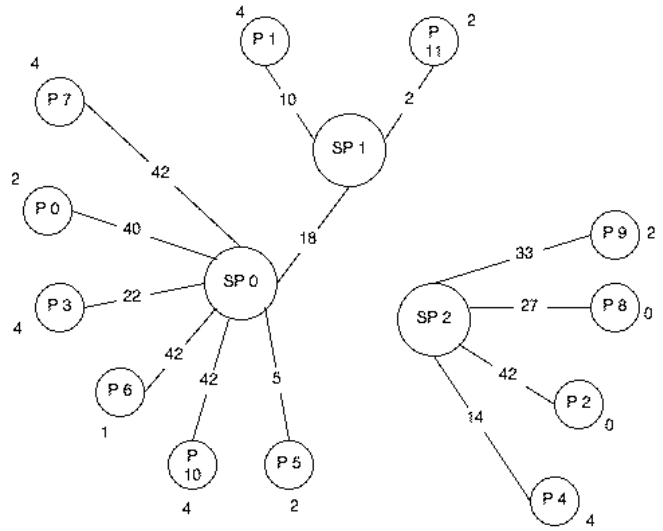

Figure 9: The highest fitness network found by the GA after evolution.

performing better than the other. Much in the way that it is difficult to visualize exactly what was learned by a large neural network, it is difficult to rationalize why ceertain peers belong to certain clusters. These P2P networks found by the GA should be thought of as "black boxes" to which queries are input and a fitness is output. In contrast, Figure 1 which solves the cancer cause query is easy to follow. This is because it was a carefully crafted example intended to answer one query and show how the GA can evolve a network that answers the query better. It is easy to see how this one query moves through clusters as missing pieces of information are filled in. If the network were larger and more queries were submitted, it would become quite unclear how one network is better than another without a considerable amount of work.

Table 2: 10 sample runs (out of 30) comparing GA, random search and hill climbing (lower is better).

Run	GA Fitness	Random search fitness	hill climbing fitness
0	3177.053711	3871.968506	3342.723145
1	3177.053711	3754.563477	3342.723145
2	3177.053711	3654.455566	3342.723145
3	3177.053711	3447.515381	3177.053711
4	3177.053711	3654.455566	3342.723145
5	3177.053711	3540.059814	3342.723145
6	3177.053711	3756.250977	3177.053711
7	3177.053711	3342.723145	3342.723145
8	3177.053711	3871.968506	3342.723145
9	3177.053711	3763.250977	3259.163086

However, verifying that the GA converges to a network with the same fitness in every run and consistently finds better solutions than random search and hill climbing is not enough to prove that it is converging on a globally optimal network. The best (and most obvious) way to assure the GA is finding a globally optimal network is to compare the solution found to one found through exhaustive search. More accurately we will be comparing the fitnesses of the networks found by both methods rather than the networks themselves, as many different networks can have the same fitness and the GA is only able to find one of these optimal solutions in every run. This phenomenon is called genetic drift because the population will drift towards some particular high fitness solution as the GA converges. After running the above experiment 30 times, an exhaustive search of network space[10] was run to verify GA convergence. The solution found by exhaustive search had a fitness of 3177.053711, proving that the GA had consistently converged on a globally optimal network.

While exhaustive search is guaranteed to always find a globally optimal network, the obvious drawback of this method is that the problem becomes intractable for networks even slightly larger than the ones in the above experiments. Even with only 3 superpeers,12 peers, and 120 queries submitted, exhaustive search took approximately 3 hours and 29 minutes to complete. The GA found an equivalent solution in around 15 seconds, making it much more practical for evolving large networks.

4.2 Experiment 2

Experiment 2 tests how the GA performs on a substantially larger network of 50 superpeers and 500 peers. Since the Dialogue network will certainly consist of more than 12 peers and 3 superpeers, it is important to

[10]Search space consisting of every possible network.

observe the GA's performance when the network size is considerably larger. Given the number of possible networks ($50^{500} * 2^{(\frac{50^2-50}{2})}$), comparing the GA results to exhaustive search (as was done in experiment 1) would be impossible to do in a reasonable amount of time. What we hope to show is not that the GA is finding a globally optimal network, but rather that it dramatically improves network performance and consistently finds networks with a much higher fitness than those found by random search or hill climbing.

Because the larger search space leads to a problem of greater complexity, the population size was increased to 200 and the network was allowed to evolve for 25,000 iterations. This time, each peer submitted 2 randomly generated queries in its area of expertise rather than 10.

4.2.1 Results

As in experiment 1, the GA produced the best networks followed by hill climbing, with the worst networks always produced by random search. However, this time hill climbing was unable to find any networks with a fitness comparable to those found by the GA.

Table 3: 10 sample runs comparing GA, random search and hill climbing on larger networks (lower is better).

Run	GA Fitness	Random search fitness	hill climbing fitness
0	2464.855957	4329.772461	2707.284180
1	2474.807617	5315.000977	2706.151367
2	2504.295898	5207.087402	2821.211914
3	2524.281982	4876.630371	2664.714111
4	2516.926758	3982.040527	2821.688965
5	2415.625977	5116.051758	2782.782959
6	2519.579590	3982.040527	2803.927734
7	2501.279541	3825.925049	3229.171387
8	2519.579590	4878.194824	2784.510010
9	2660.644531	4342.915039	2842.250977

To demonstrate the dramatic increase in network performance, we must compare the best network at the beginning of the GA's run to the best found when the GA terminates. In the fourth run the initial population had a highest fitness of 3454.398926. If you look at table 3, the GA found a network with a fitness of 2524.281982, a difference of approximately 930.12. Other runs gave a change in fitness from best initial population member to final solution found of 500 – 900. Even in the worst case, we are still dramatically improving network fitness.

4.3 Experiment 3

Experiment 3 tracks the progress of the network over 30 simulated days. We want to see how the solution found by the GA one day performs when tested on queries submitted the next day. This involves initially configuring the simulated Dialogue network randomly, submitting and answering queries during the day, performing reorganization by running the GA on sample queries at night, and using the best network found by the GA to configure the Dialogue network for the next day. A network of 25 superpeers and 250 peers was used. These 250 peers are allowed to submit 10 queries each over the network per day, and the worst

25 answered queries are used that night to test each GA population member. A population size of 150 was used and the GA was allowed to evolve for 15,000 iterations. The following chart outlines the general algorithm used for this experiment:

1. Create a single random solution to be used in configuring the Dialogue network for the first day.
2. Allow peers connected to the Dialogue network to submit and answer queries all day.
3. At the end of the day, choose 25 previously submitted queries to be used in testing each GA population member.
4. Generate 149 other population members. This will be the population the GA evolves.
5. For every population member, configure the Dialogue network, submit the 25 sample queries, and record the population member's fitness. This will fill in the fitness values for the initial population.
6. Run the GA for 15,000 iterations, after which the GA is stopped and we note the population member with the best fitness.
7. Configure the Dialogue network based on the best solution found by the GA.
8. Goto step 2

To further explain step 4, new populations were formed as follows: the first member of the new population is always the best network from the previous day. This ensures that the previous day's best network has an influence on the current population. We then loop through the rest of the population, creating new members by copying the previous day's best network, and then stepping through each cluster and connection gene and mutating with a probability of 0.5.

It should also be noted that peers will usually generate queries in their own area of expertise. As described in section 2, each peer specializes in an area of knowledge. This is represented as a range of integers such as 0–9, 10–19, 20–29, and so on. Because peers usually ask queries in their own area of expertise, the network found by nightly reorganization one night should still perform well on the next day's queries.

4.3.1 Results

Table 4 shows the results of the last 10 days of the thirty day simulation. Each row contains the following information: The fitness of the previous day's network when tested on the current day's queries, the best and worst fitness when the initial GA population is generated and nightly reorganization begins, and the best fitness found by the GA after evolution. By comparing the last column of row n and the first column of row $n+1$, we can compare how well the best network found on day n generalizes to answer the queries on day $n+1$. While the previous day's network does not always provide the optimal network for the current day's queries, it does always provide a very good solution.

Table 5 shows sample times required to complete nightly reorganization for networks of various sizes. For a network consisting of 10 superpeers and 100 peers, 248.84 seconds is required for the GA to complete 15,000 iterations. A network of 50 superpeers and 500 peers requires a little more than 5 hours for the GA to complete nightly reorganization. For large networks, the time required to complete 15,000 GA iterations quickly becomes too large. The obvious solutions are to test networks using fewer queries, and most

Table 4: 10 sample days comparing the best network found the previous day, the current best (not including the previous day's network) and worst before reorganization, and best found after reorganization (lower is better). All networks were tested on the current day's queries.

Day	previous day	current best	current worst	best network found
21	—	3915.872803	6906.364746	2211.458008
22	3074.716324	3156.817383	7571.601074	1896.735962
23	2340.185731	2321.773682	6906.364746	2336.970703
24	2710.011743	3915.872803	8358.556641	1946.076050
25	2321.773682	2321.773682	7593.101074	2321.773682
26	2898.143896	2729.072510	7568.300781	2610.543457
27	2529.007421	2738.643799	6593.522461	2474.678467
28	3265.657480	3020.647949	7556.951172	2760.764648
29	2554.114238	2915.424072	7226.524902	2563.967773
30	2661.772110	2661.104492	7582.400879	2373.247070

importantly to reduce the number of iterations the GA is allowed to run for. By limiting the simulation to 8 hours instead of a fixed number of iterations, we can still find a good design and the network will perform well on the next day's queries.

Table 5: Comparing network size to time required for nightly reorganization.

superpeers	peers	max eval time (in seconds)
3	12	23.37s
10	100	248.84s
50	500	21409.97s
100	1000	15228221.7s[11]

5 Conclusion

5.1 Summary

We have outlined a GA intended to be applied to the network reorganization problem. It employs commonly used operators to achieve high performance when compared to other methods such as random search and hill climbing.

Each population member consists of two separate substrings and represents a parametric description of a superpeer network design. The first substring encodes which superpeer cluster each peer belongs to and is made up of n integers, where n is the number of peers in the network. Each of these n genes can be an integer from 0 to m, m being the number of superpeers in the networks. The second substring is composed of $\frac{m^2-m}{2}$ genes, the exact number of distinct connections possible between m superpeers. These genes are boolean, taking on a value of 1 or 0 depending on if a connection exists between two given superpeers. These two strings, although contained in the same population member, are evolved and operated on separately. The information needed to compute a population member's fitness, specifically how many queries were successfully answered and the total time taken to answer them, is obtained by reconfiguring

the network based on that population member and then resubmitting a select subset of previously submitted queries. A steady state model was used in which common operators are applied to two parents, chosen by tournament selection with a tournament size relative to the size of the population. These two parents are chosen to produce one offspring by one-point and uniform crossover, and uniform mutation. This newly created population member is then reinserted into the current population replacing either the current worst member or a member randomly chosen from the worst 10% of the population. The process is continued for a fixed number of iterations depending on the number of nodes currently in the network and the population size.

5.2 Review of Contributions

This article presented a novel use for steady state GAs by applying them to the problem of optimizing a superpeer network for information discovery. In section 4, the GA method was compared to random search, hill climbing, and for smaller networks, exhaustive search. Both the GA, random search, and hill climbing were allowed to run for 5000 iterations.

All networks found by random search exhibited poor fitnesses, usually considerably worse than those networks found by hill climbing. Unlike random search, hill climbing produced no networks that were unable to successfully answer any queries. However, hill climbing was able to converge on networks as fit as those found by the GA only 7 out of 30 times, with the rest of the networks found being of relatively poor fitness. The most consistent method was the GA, which was able to find the highest fitness networks in every run. Furthermore, it was proven that for small networks, not only was the GA able to find the highest fitness solutions but it also consistently found globally optimal networks. This was verified by comparing the fitnesses of networks found by the GA to those found by an exhaustive search of network space. On average, the GA found these globally optimal solutions in approximately 15 seconds. Comparing this to exhaustive search, which took 3.5 hours to complete, shows that the GA was able to find optimal networks in a very reasonable amount of time.

Experiment 2 tested the GA's ability to reorganize considerably larger networks. Because it was not possible to run an exhaustive search, the GA was compared only to hill climbing and random search. As in experiment 1, the GA produced the best networks followed by hill climbing, with the worst networks always produced by random search. However, this time hill climbing was unable to find any networks with a fitness comparable to those found by the GA. While it cannot be verified that the GA is finding a globally optimal solution, it is important to note the dramatic difference in fitness between the initial GA population and the GA population after 25,000 iterations. Across all runs, the change in fitness from best initial population member to final solution found was always between $500 - 900$. This is a dramatic improvement that no other method was able to match.

Experiment 3 simulated 30 days of network activity, including nightly reorganization, for a network of 25 superpeers and 250 peers. We began with a single randomly generated network that was used to answer the first day's queries. Every day, each of the 250 peers are allowed to submit 10 queries each to the network. During nightly reorganization, a random population of 150 networks was created and the GA was allowed to evolve for 15,000 iterations. Twenty five sample queries from that day were used to test each population member and compute its fitness. The best network found by the nightly reorganization was used to answer the next day's queries. While the network found by the GA the previous night is not always the best network for answering the next day's queries, it does always provide a very good solution. Furthermore, networks of various sizes were tested and the amount of time required to run nightly

reorganization listed in Table 5. For large networks, the time required to complete 15,000 GA iterations quickly becomes prohibitively large. Possible solutions include testing networks using fewer queries and reducing the number of iterations the GA is allowed to run for.

5.3 Future Work

Although it was not explicitly part of the objective function outlined in this article, the number of inter-superpeer connections should be taken more into consideration. A larger number of these connections will cause queries to be propagated to more clusters, giving them a greater chance of being successfully answered. However, this also would lead to bandwidth saturation and degrade overall network performance. It is worth noting that the current objective function depends largely on the overall time taken to answer queries. Because of this, network designs that connect a large number of superpeers, therefore leading to network slowdowns, may have lower fitnesses. This will not always be the case, depending on what subset of daily queries are submitted to the GA during nightly reorganizing. We may also want to use repair operators to ensure that every superpeer cluster is connected to at least one other superpeer, leaving no cluster as an island.

The GA may benefit, in the case of larger networks, from a more intelligent replacement strategy. Currently, the two replacement strategies being used are replacing the worst population member and replacing a random population member from the worst 10% of the population. Perhaps a better strategy would be to employ a niching method such as crowding. In crowding, which is commonly used in steady state GAs, only the worst 10% of the population is considered for replacement. Unlike the replacement strategy currently used, we choose the population member that is most similar to the newborn for replacement, rather than choosing a random population member. This should help the GA to maintain diversity longer, giving us a better chance of finding better networks.

Finally, a means of better determining when the GA should be stopped may be helpful. This may help with diversity but more importantly, it will allow us in some cases to stop the GA early. Given the large number of nodes that may be present and the time required to reconfigure the network and resubmit queries, the most scarce resource is time. When testing population members, we can't increase the speed with which queries are answered, so reducing the time the GA takes to run is imperative.

References

[1] Robert E. Beverly IV. Reorganization in Network Regions for Optimality and Fairness. *MSc Thesis*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, August, 2004.

[2] Simon G. M. Koo, C. S. George Lee, and Karthik Kannan. A Genetic-Algorithm-Based Neighbor-Selection Strategy for Hybrid Peer-to-Peer Networks. In *Proceedings of the 13th IEEE International Conference on Computer Communications and Networks (ICCCN'04)*, Chicago, IL, October 2004.

[3] Panos Kalnis, Wee Siong Ng, Beng Chin Ooi, Dimitris Papadias, and Kian Lee Tan. An Adaptive Peer to Peer Network for Distributed Caching of OLAP Results. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, 2002, pp. 25-36.

[4] Emanuel Falkenauer. A Hybrid Grouping Genetic Algorithm for Bin Packing. In *Journal of Heuristics, Vol. 2, No. 1*, 1996, pp. 5-30.

[5] Brian F. Cooper, and Hector Garcia-Molina. "Peer-to-Peer Resource Trading in a Reliable Distributed System." In *Electronic Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[6] Stephanos Androutsellis-Theotokis. A Survey of Peer-to-Peer File Sharing Technologies." *White paper*, Electronic Trading Research Unit (ELTRUN), Athens University of Economics and Business, 2002.

[7] Zbigniew Michaelwicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, 3rd, rev. and extended ed. 1996. Corr. 2nd printing, 1998.

[8] Resource Description Framework (RDF) Model and Syntax Specification, *W3C Recommendation*, February 1999 (http://www.w3.org/TR/REC-rdf-syntax/).

[9] ATIS Telecom Glossary, The Alliance for Telecommunications Industry Solutions, February 2001 (http://www.atis.org/tg2k/_network_architecture.html).

[10] Douglas E. Comer, and David Stevens. 1996. Internetworking With TCP/IP, Volume III. Prentice Hall, May 1997, 513 pages.

[11] Qin Lv, Sylvia Ratnasamy, and Scott Shenker. Can Heterogeneity Make Gnutella Scalable. In *Electronic Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[12] Eytan Adar, and Bernardo Huberman. Free Riding on Gnutella. First Monday, Volume 5, Number 10 October 2nd, 2000.

[13] Kunwadee Sripanidkulchai. The Popularity of Gnutella Queries and its Implications on Scalability. featured on O'Reilly's www.openp2p.com website, February 2001.

[14] Michael Mitzenmacher. Compressed Bloom Filters. In *Proc. of the 20th Annual ACM Symposium on Principles of Distributed Computing*, 2001, pp. 144-150.

[15] Michael R. Garey, and David S. Johnson. Computers and Intractability–A Guide to the Theory of NP-Completeness. W. H. Freeman And Company, New York, 1979, 340 pages.