Peer-to-Peer Netw. Appl. (2015) 8:120–136
DOI 10.1007/s12083-013-0236-0

An efficient and distributed file search in unstructured peer-to-peer networks

Mohammad Shojafar · Jemal H. Abawajy · Zia Delkhah · Ali Ahmadi · Zahra Pooranian · Ajith Abraham

Received: 21 March 2013 / Accepted: 9 September 2013 / Published online: 29 September 2013

Abstract Peer-to-peer (P2P) networks are gaining increased attention from both the scientific community and the larger Internet user community. Data retrieval algorithms lie at the center of P2P networks, and this paper addresses the problem of efficiently searching for files in unstructured P2P systems. We propose an Improved Adaptive Probabilistic Search (IAPS) algorithm that is fully distributed and bandwidth efficient. IAPS uses ant-colony optimization and takes file types into consideration in order to search for file container nodes with a high probability of success. We have performed extensive simulations to study the performance of IAPS, and we compare it with the Random Walk and Adaptive Probabilistic Search algorithms. Our experimental results show that IAPS achieves high success rates, high response rates, and significant message reduction.

Keywords Unstructured peer-to-peer · Adaptive probabilistic search · Ant colony · Success rate · Information retrieval

M. Shojafar (✉)
Department of Information Engineering, Electronics and Telecommunication (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184 Rome, Italy
e-mail: Shojafar@diet.uniroma1.it

M. Shojafar
e-mail: m.shojafar@yahoo.com

J. H. Abawajy
Senior IEEE Members, School of Information Technology, Deakin University, Waurn Ponds, Australia

Z. Delkhah · A. Ahmadi
Department of Electrical and Computer Engineering, Qazvin Islamic Azad University, Qazvin, Iran

Z. Pooranian
Department of Computer Engineering, Islamic Azad University, Andimeshk Branch, Andimeshk, Iran

A. Abraham
Machine Intelligence Research Labs (MIR Labs), Auburn, WA, USA

A. Abraham
IT4Innovations - Center of excellence, VSB – Technical University of Ostrava, Ostrava – Poruba, Czech Republic

1 Introduction

A peer-to-peer (P2P) network is a distributed system in which computing nodes employ distributed resources to perform a critical function in a decentralized manner. Nodes in a P2P network normally play equal roles and are therefore called *peers*. Since P2P systems are capable of searching and communicating across the globe, they have become phenomenally popular in recent years [1]. Examples of P2P applications are distributed file-sharing systems, event notification services, and chat services [2–5].

P2P networks can generally be classified, based on the control over where data are located and the network topology, as structured or unstructured [6]. In an unstructured P2P network such as Gnutella http://www.gnutella.com, peers are typically connected to a random set of neighbors. There are no rules that strictly define where data is stored and which nodes are neighbors. Moreover, as there is no need to maintain particular network architecture, unstructured P2P systems are very efficient when nodes join or leave the network.

In contrast, structured P2P networks have well-defined neighbor links and can be further classified as loosely structured or highly structured. In highly structured P2P networks such as the network in Chord [7], the neighbors of nodes, the network architecture, and the locations for storing data are precisely specified. In loosely structured P2P networks such as Freenet [8] and Symphony [9], the overlay structure and data locations are not precisely determined. In Freenet [8], both the overlay topology and the data location are determined based on hints, and the network topology eventually evolves into some intended structure. In Symphony, the overlay topology is determined probabilistically but data locations are precisely defined.

The principal requirement in any peer-to-peer network is efficient searching for desired resources such as data or files. Heuristic and meta-heuristic methods have proven to be efficient for solving many hard network search problems. They are also demonstrating their usefulness in the P2P network domain, especially for unstructured P2P networks. Ant Colony Optimization (ACO) [10] is inspired by observations of the foraging behavior of real ant colonies. While moving from food sources to the nest and vice versa, ants deposit a substance called *pheromone* on the ground, forming a pheromone trail. When choosing their way, ants can smell pheromone and tend to choose paths marked by strongly probable pheromone concentrations. While an isolated ant moves practically at random, an ant encountering a previously laid trail can detect it and decide to follow it with high probability, thus reinforcing the trail with its own pheromone. The collective behavior that emerges is a form of autocatalytic process where the more the ants follow a trail, the more attractive that trail becomes for following. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that have previously chosen the same path. This method is one of the approaches that scholars have applied to unstructured P2P networks in the last decade, including AntNet [11], Anthill [12], AntSearch [13], SemAnt [14], ACO-based Search [15], and APS [16]. In the next section, the last three approaches are discussed in more detail.

In this paper, we present a novel Improved Adaptive Probabilistic Search (IAPS) algorithm for locating files in unstructured P2P networks. We focus on unstructured P2P networks because most popular P2P applications operate on unstructured networks. The highly transient node populations and ad hoc organization of unstructured P2P networks make it much more challenging to locate desired resources in these networks than in structured P2P networks. Conventional search methods for unstructured P2P networks use flooding and its variations or various types of indices that are quite expensive to maintain, may not scale well [17], and can incur heavy overhead [18]. In contrast, IAPS considers the file's type and a score based on previous searches for this file type to minimize the search space and, as a result, the search overhead. We performed extensive simulations to study the performance of IAPS, and we compared its performance with the popular Random Walk and Adaptive Probabilistic Search (APS) algorithms. Our experimental results show that IAPS achieves a better trade-off between search cost and performance than both Random Walk and APS for a wide range of P2P scenarios.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 provides the system model of our works. Section 4 presents the details of IAPS, our proposed search method. Section 5 analyzes the performance of IAPS and compares it with the Random Walk and APS algorithms. Finally, Section 6 presents our conclusions and future research directions.

2 Related work

Many search algorithms for unstructured P2P networks have been proposed in recent years. The algorithms can be classified as deterministic or probabilistic. In a deterministic approach [19, 20], query forwarding is deterministic, while in a probabilistic approach [21, 22], query forwarding is probabilistic, random, or based on resource ranking. Searching schemes in unstructured P2P systems can also be categorized as regular-grained or coarse-grained. In a regular-grained approach, all nodes participate in query forwarding. In a coarse-grained scheme, query forwarding is performed by only a subset of the nodes in the network. Existing search methods can also be classified into blind searches and informed searches [16, 22]. In blind searches, nodes do not store information about data locations, while in informed searches; nodes store some metadata that facilitates the search.

Flooding-based searching is the most commonly used method to locate specific data items http://www.gnutella.com. Flooding involves a breadth-first search (BFS) of the overlay network graph, with the querying node contacting nodes that are reachable within a specified time-to-live (TTL) number of hops. To perform this search, the querying node sends a query request to all of its neighbors. When a neighbor receives the query, it returns the result if it has the requested data. Otherwise, it forwards the query request to all of its neighbors except the querying node. This procedure continues until the depth limit time-to-live (TTL) is reached. Although flooding is a simple algorithm, it generates a large number of messages (many are duplicate messages) and does not scale well. Several approaches have been proposed to address this problem. These include variations on a depth-first search (DFS) that, for example, use routing indices [23], an attenuated bloom filter [19], and Successful Paths in Unstructured Networks or SPUN [24]. They also include BFS variations such as iterative deepening [20], a random k-walk [6], a modified random BFS [25], a two-level random k-walk [21], a directed BFS [20], an intelligent search [25], a search based on local indices [20], the Adaptive Probabilistic Search [22], and a search based on dominating sets [26]. In the iterative deepening and local index searches, queries are forwarded to all neighbors of the forwarding node, while in all other schemes, queries are forwarded only to a subset of those neighbors.

The Adaptive Randomized Search Protocol (ARSP) [27] is an extension of the Quorum-Based Protocol for Searching Objects in Peer-to-Peer Networks that minimizes the protocol's communication cost by disseminating objects to nodes based on popularity indices. Although the protocol automatically adapts to the dynamics of a network environment, it does not consider file types for performing the search. An entity retrieval model with a compressed index data structure is presented in [28]. A super-peer-based coordinated service provision framework (SCSP), consisting of a super-peer-based labor market (S-labor market) model, a recruiting protocol based on a weighting mechanism, and an optimal dispatch algorithm, is proposed in [29]. Although this framework has good scalability and robustness, it takes a long time for the algorithm to adapt to the dynamic environment of unstructured P2P systems. An approach that minimizes data processing time by choosing a set of peers based on their computation and communication capabilities is discussed in [30].

SPUN [24] is a controlled flooding algorithm in which each node probabilistically forwards queries to its neighbors. Each peer keeps a local index of its neighbors' relative probabilities of successfully locating data. Searching is based on the querying peer's simultaneous deployment of k walkers that are selected probabilistically, based on the querying peer's local index. Each walker travels along its own path in the network and returns a hit or a miss message to the originator along the reverse query path. A peer that finds the requested object in its local storage returns a hit and the query is terminated. Otherwise, the query is forwarded to the next neighbor. If a file is not found within the TTL period, the peer returns a miss. Each peer includes its profile in any message (query, hit, miss) that it sends, and each peer along the query path, excluding the one farthest from the requestor, sends its relative probability of success back to its previous neighbor in the query path. Each intermediate peer along the reverse query path uses this information to update its local index.

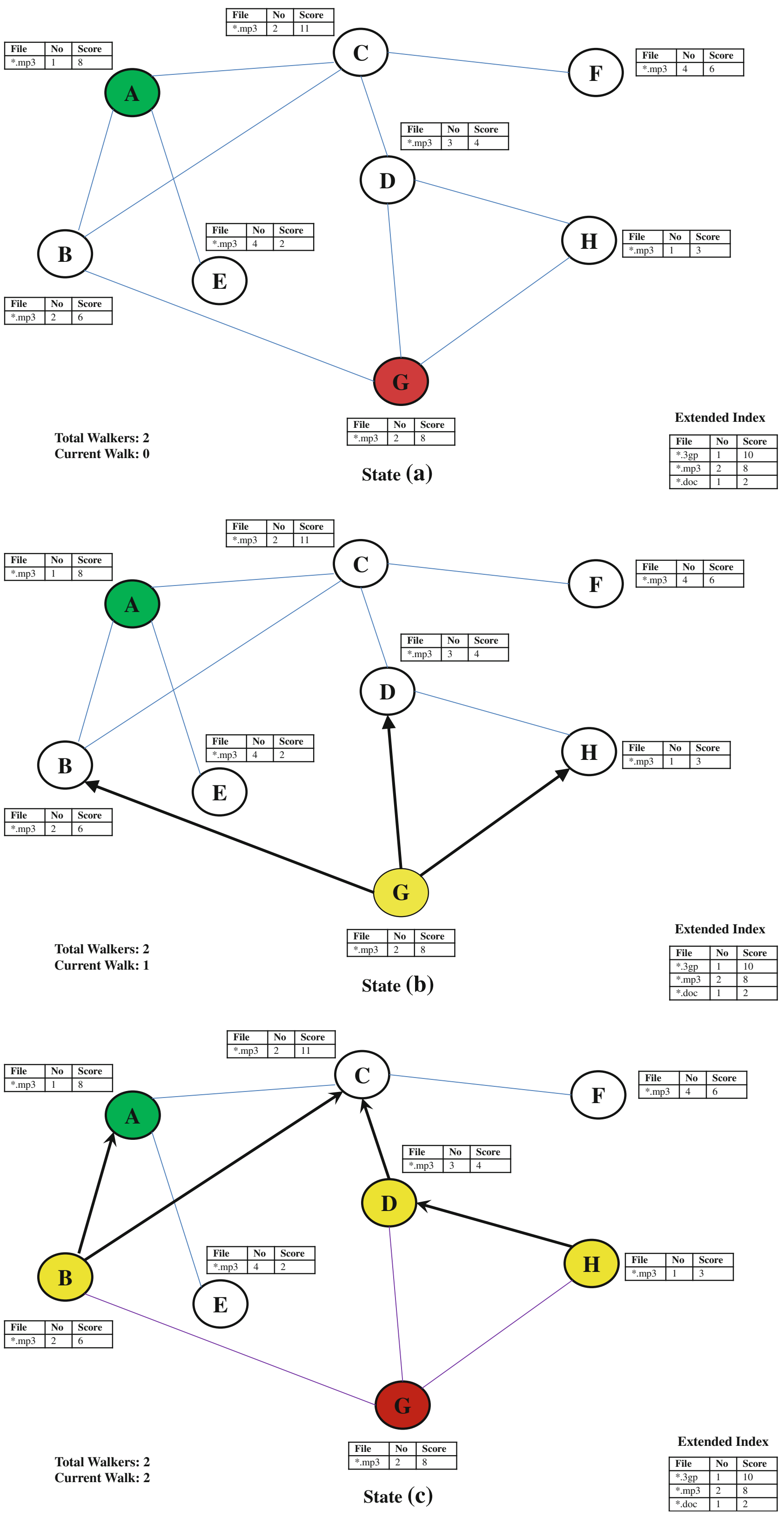
An alternative approach to flooding is the Random Walk algorithm. In the standard Random Walk algorithm, the querying node forwards the query message to one randomly selected neighbor. This neighbor randomly chooses one of its neighbors and forwards the query message to that neighbor. This procedure continues until the data is found. The standard Random Walk algorithm can greatly reduce the message overhead but incurs a longer search delay. The k-walker Random Walk algorithm [6] attempts to reduce the search delay by having the querying node forward k copies of the query message to k randomly selected neighbors. Each query message takes its own random walk. Each walker periodically communicates with the querying node to decide whether that walker should terminate. Nodes can also use soft states to forward different walkers for the same query to different neighbors. On the average, the total number of nodes reached by k random walkers in H hops is the same as the number of nodes reached by one walker in kH hops, so the routing delay is expected to be k times smaller when k walkers are used rather than a single walker.

In [15], the authors have presented a new ACO-based search in AntNet [11], a method that uses two tables (i.e., local information and routing tables). This method is similar to our approach, which searches the local database before transmitting a query request to available neighbors. Both methods use Internet Control Message Protocol (ICMP) packets to avoid network congestion or oversubscription, as well as avoiding requested services not being available and a hosts, router, and neighbor nodes being unreachable, and also to relay query messages. Both methods use forward and backward ants with similar details, preventing loops and cycles to compel the forward ant to come back to the nest after finding an available destination peer for the keyword (i.e., our method uses the entire TTL deadline for the forward ant and indicates the TTL in searching queries, but with different policies). The pheromone trail updating policy in our method is completely different from the policy in [15], but both update the trail by the backward ant for each query request. Our model is based on Gentulla, while the basic model of [15] comes from AntNet [11]. The method in [15] does not control bandwidth and latency, nor does our approach.

In another work [14], the authors presented the SemAnt method for distributed query routing by the ACO metaheuristic algorithm in fixed and ad hoc networks with churn peers. Both our method and the SemAnt method use reputation learning, manage document repositories, and lead their data to the other peers in searching queries, and both perform keyword-based searches on their meta-data. Both methods use independent forward and backward ants to reward the nodes that take part in query occurrences, and both methods use forward and backward ants to search the query request from visited peers and intermediate peers. The method of scoring in SemAnt is based on regularly updating pheromone and evaporating pheromone and is different from our proposed method. Our method uses a number scoring system for the intermediate and destination peers, so that it can perform a better search for file types for similar queries in the next phase, because we pay attention to the types of files requested by queries. A score corresponds to each type, to help new queries for different types search more easily than SemAnt, which searches each node based on the current table's probability for nodes after evaporation. We note that we also use an evaporation rule, but not as a probability: our reduction is based on intervals, as discussed in the next section. Neither method is able to consider the impact of bandwidth and latency in the selection phase for finding the best visited paths. Both methods use the concept of random leaf selection. The authors in [14] modeled

Table 1 Query table status indexed for each node

File type	File format	Number of copies files	Score of files
Video	*.3gp	500	80
Audio	*.mp3	300	200
Document	*.doc	200	1300



State (a)

A:

File	No	Score
*.mp3	1	8

B:

File	No	Score
*.mp3	2	6

C:

File	No	Score
*.mp3	2	11

D:

File	No	Score
*.mp3	3	4

E:

File	No	Score
*.mp3	4	2

F:

File	No	Score
*.mp3	4	6

G:

File	No	Score
*.mp3	2	8

H:

File	No	Score
*.mp3	1	3

Extended Index

File	No	Score
*.3gp	1	10
*.mp3	2	8
*.doc	1	2

State (b)

A:

File	No	Score
*.mp3	1	8

B:

File	No	Score
*.mp3	2	6

C:

File	No	Score
*.mp3	2	11

D:

File	No	Score
*.mp3	3	4

E:

File	No	Score
*.mp3	4	2

F:

File	No	Score
*.mp3	4	6

G:

File	No	Score
*.mp3	2	8

H:

File	No	Score
*.mp3	1	3

Extended Index

File	No	Score
*.3gp	1	10
*.mp3	2	8
*.doc	1	2

State (c)

A:

File	No	Score
*.mp3	1	8

B:

File	No	Score
*.mp3	2	6

C:

File	No	Score
*.mp3	2	11

D:

File	No	Score
*.mp3	3	4

E:

File	No	Score
*.mp3	4	2

F:

File	No	Score
*.mp3	4	6

G:

File	No	Score
*.mp3	2	8

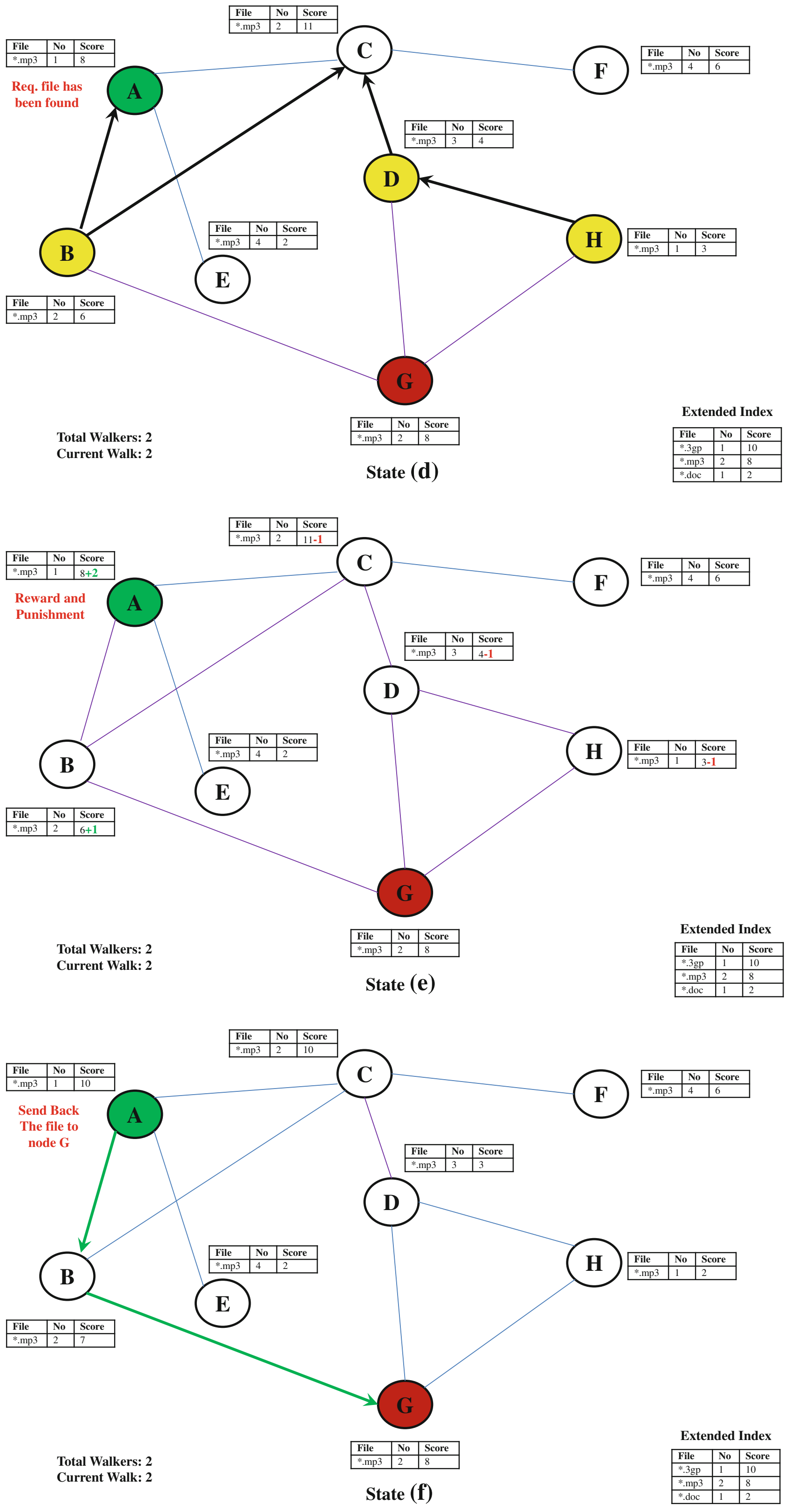
H:

File	No	Score
*.mp3	1	3

Extended Index

File	No	Score
*.3gp	1	10
*.mp3	2	8
*.doc	1	2

Fig. 1 General scoring procedure in IAPS



State (d)

	File	No	Score
A	*.mp3	1	8
B	*.mp3	2	6
C	*.mp3	2	11
D	*.mp3	3	4
E	*.mp3	4	2
F	*.mp3	4	6
G	*.mp3	2	8
H	*.mp3	1	3

Extended Index

File	No	Score
*.3gp	1	10
*.mp3	2	8
*.doc	1	2

State (e)

	File	No	Score
A	*.mp3	1	8+2
B	*.mp3	2	6+1
C	*.mp3	2	11-1
D	*.mp3	3	4-1
E	*.mp3	4	2
F	*.mp3	4	6
G	*.mp3	2	8
H	*.mp3	1	3-1

Extended Index

File	No	Score
*.3gp	1	10
*.mp3	2	8
*.doc	1	2

State (f)

	File	No	Score
A	*.mp3	1	10
B	*.mp3	2	7
C	*.mp3	2	10
D	*.mp3	3	3
E	*.mp3	4	2
F	*.mp3	4	6
G	*.mp3	2	8
H	*.mp3	1	2

Extended Index

File	No	Score
*.3gp	1	10
*.mp3	2	8
*.doc	1	2

Fig. 1 (continued)

their content for each peer based on ACM computing classification systems [31], but we use Gnutella network features [32]. The results we achieved are more realistic as compared to the approach in [14].

The Adaptive Probabilistic Search (APS) algorithm [16] utilizes feedback from previous searches to make future searches more efficient. In APS, each node maintains a local index with one entry for each object it has requested or for which it has forwarded a neighbor's request. The value of the entry reflects the relative probability of this node's neighbor being chosen as the next hop in a future request for the object. The search uses k walkers and probabilistic forwarding. The requester chooses k out of its N neighboring nodes (while the query is still being sent to neighbors) to forward the request to. Each of these neighbor nodes compares the query with its local repository, and if a hit occurs, the walker terminates successfully. If a miss occurs, the query is forwarded to one of the node's neighbors. This procedure continues until all k walkers have terminated with either a success or a failure. Thus, while the requesting node forwards the query to k neighbors, all of the other nodes forward it to only one. In the forwarding process, a node does not choose its next-hop neighbor(s) randomly, but rather based on the probabilities in its local index. At each forwarding step, nodes append their identifiers to the search message and maintain a soft state about the search they have processed. If two walkers from the same request cross paths (i.e., a node receives a duplicate message due to a cycle), the second walker is assumed to have terminated with a failure and the duplicate message is discarded. Efficient (delay tolerant and intolerant) data sharing mechanisms in P2P networks and current video coding trends are elaborated in detail in [33], where it is shown that APS has the drawback of depending on the number of walkers and the broadband width. Our proposed IAPS algorithm eliminates these disadvantages. In addition, IAPS has a high success rate, less overhead (because duplicate messages are eliminated), and a high hit per query rate.

3 System model

In the Improved Adaptive Probabilistic Search (IAPS) algorithm, each peer maintains information about files and shares this information with its single-hop neighbors. Table 1 (the Query Table) shows the information that a node maintains about each file. Recall that, in the search algorithm for a P2P network, the destination node is unknown and the contents of search packets are variable, so the content of a query, which is expressed by a keyword and syntax, should be considered in the design of the routing or query table. The Query Table has four columns representing the file type, the file format, the number of copies of the file, and the scores of the file. Initially, all tables' entries are commenced with the same amount of file-types files. Before a node leaves the network, it must distribute all of the entries in its index table to its neighbors in a uniform and balanced style. This ensures that the attenuated bloom filters (data structures that reside at each node in the system) [19] in its index table are still available while the node is offline. When a node rejoins the network, it asks its neighbors to distribute some entries in their index tables to fill in its own index table. A query (request) Q consists of one or more keywords $k_1,\ldots,k_n$, which are connected by the boolean operator OR or AND. At startup, all table entries for each node (peer) are initialized with the same small value for each file type's scores.

IAPS maintains a score for each file based on the results of previous searches for the file; the score is an indication of how many times the file has been referenced. For example, the

Table 2 File-type probability table τ for each node

File type	File format	File scores	Probability
Video	*.3gp	80	80/(80+200+1200)
Audio	*.mp3	200	200/(80+200+1200)
Document	*.doc	1200	1200/(80+200+1200)

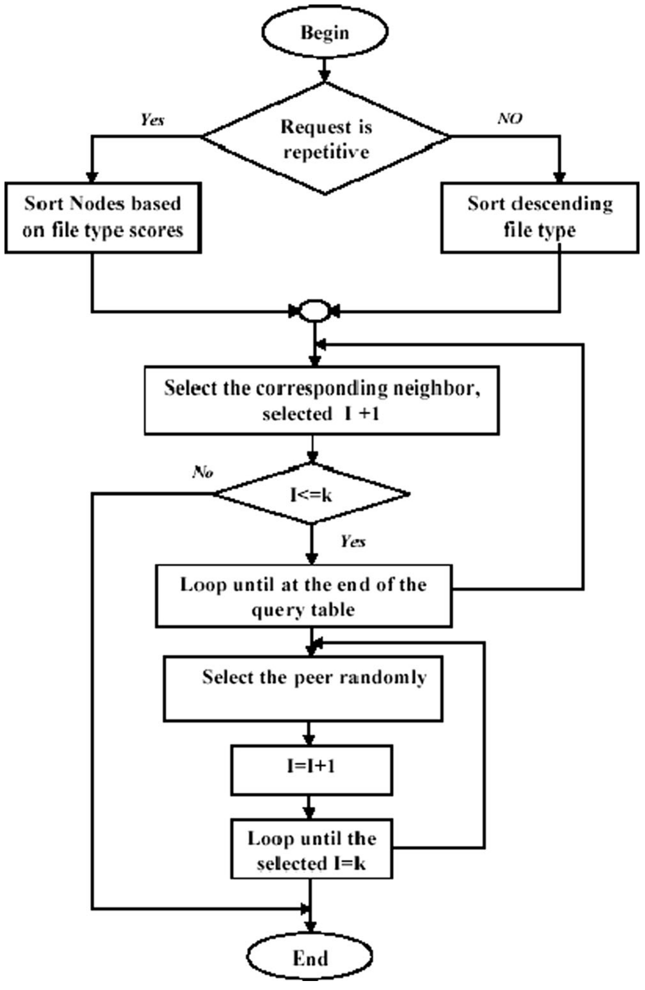

Begin
Request is repetitive
Yes
NO
Sort Nodes based on file type scores
Sort descending file type
Select the corresponding neighbor, selected I +1
No
I<=k
Yes
Loop until at the end of the query table
Select the peer randomly
I=I+1
Loop until the selected I=k
End

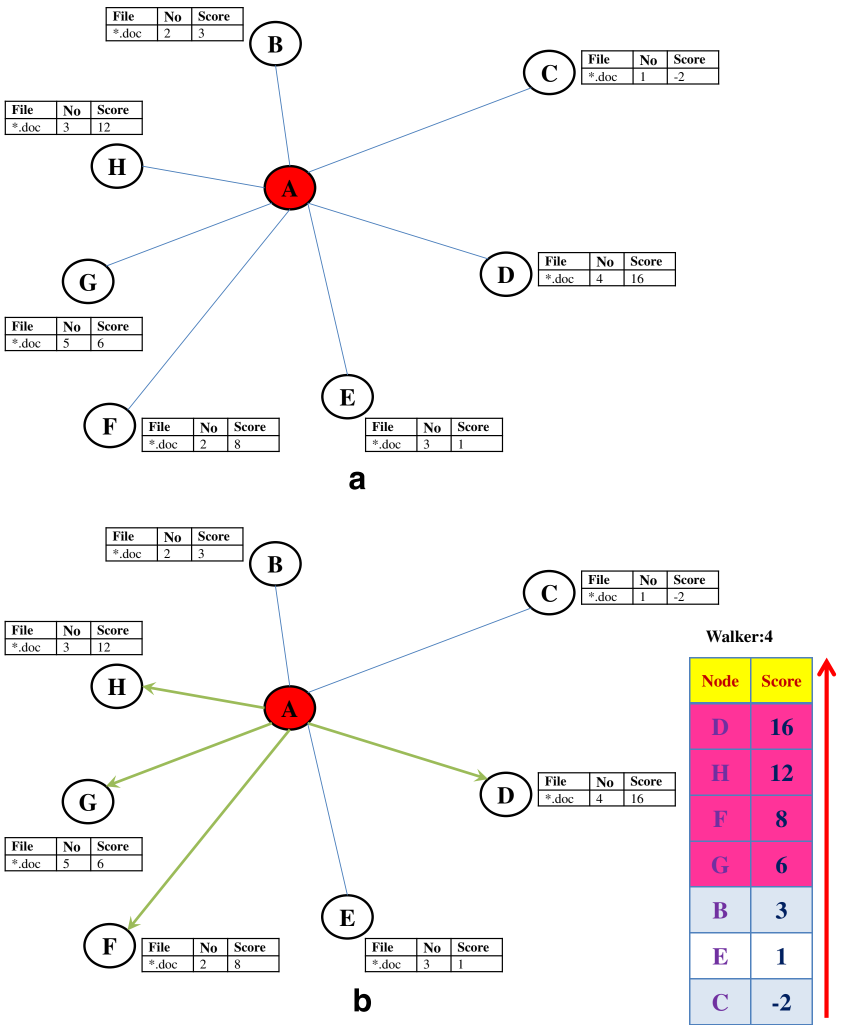
Fig. 2 Flowchart for the IAPS neighbor selection phase

score in Table 1 for a file with *a .mp3* extension indicates that there have been 200 references to that file. The file format score maintained in each node is increased or decreased based on the search request and the placement of the node along the path traveled by the file request. The type of request depends on the awareness of the data structure by the user and on the user expertise. We use requests like [34]. If a node has the requested file, the node increases its score for the file by two points (we have selected this amount to show how the pheromone of the forward ant can calculate the nodes that will cause the query to reach the destination node), and all other nodes along the route to the destination node, excepting the node that originated the request, increase their file score by one point. If there is more than one path from the destination node to the requesting node, the path with fewer hops is selected, and the nodes in the other paths are rewarded the same points as the nodes of the selected path. However, if the requested file was not found or the route comes back to the requesting node, one point is deducted from the score of each of the nodes that assessed the request.

Figure 1 illustrates how the score for the file is computed. In this figure, we consider two walkers, and node G is the originating node for a request for *a .mp3* file that is hosted by node A. The figure also displays next to each node its index table with its current state (state a).

Node G sends a request message to all of its neighbors (broadcasting request message [34]) (state b). We search the index tables of the *unvisited* neighbors of each node simultaneously. Then nodes B, D, and H search their databases concurrently. If a node finds the requested file, it sends a response to node G. Otherwise, the node sends the request to its *unvisited* neighbors. For example, node D is not visited because of the path $D \rightarrow G$ (state c).

In state (d), node B sends a request message to all of its neighbors that have not already received the request. Node A searches its database and finds the file. Now, the score phase to update the score for all nodes engaged in the search begins (state e). The nodes along the path $A \rightarrow B \rightarrow G$, except the requesting node, are rewarded with additional points, while all other nodes that engaged in the search process but were not



File	No	Score
*.doc	2	3

File	No	Score
*.doc	1	-2

File	No	Score
*.doc	3	12

File	No	Score
*.doc	4	16

File	No	Score
*.doc	5	6

File	No	Score
*.doc	2	8

File	No	Score
*.doc	3	1

Node	Score
D	16
H	12
F	8
G	6
B	3
E	1
C	-2

Fig. 3 The IAPS selection phase

successful in locating the file in their database have one point deducted. Thus, node A is rewarded twice compared to the other nodes that receive additional points. Finally, node A sends the requested file back to node G along the selected path (state f).

In this example, we used two walkers. If, instead, we consider three walkers, the nodes that are rewarded will change. This is because node C will send a request to node A and receive its response. The reward for node A will then be four points, while the rewards for each of nodes B, C, and D will be one point. Also, we will have two alternative paths ($A \rightarrow B \rightarrow G$ and $A \rightarrow C \rightarrow D \rightarrow G$) for sending the response back to node G. We do not consider shared file types in this paper, because there is no supplementary data about shared file types among nodes. In the following section, we discuss IAPS in greater detail.

4 Improved adaptive probabilistic search (IAPS)

Our proposed Improved Adaptive Probabilistic Search (IAPS) method consists of four phases: (1) the search phase, (2) the neighbor selection phase, (3) the scoring phase, and (4) the flow control phase.

4.1 Search phase

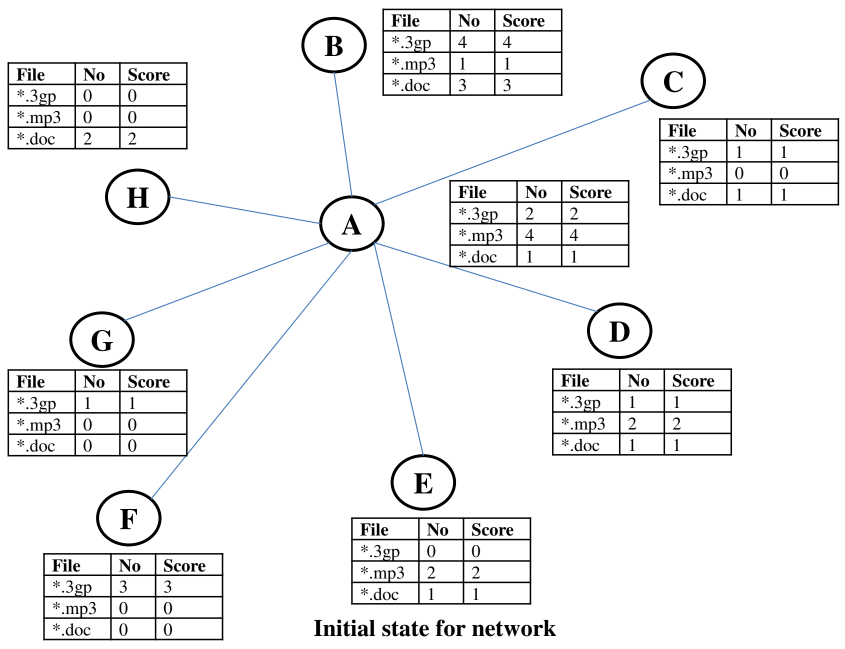
The search actions are as follows. There are two states in which a search can be conducted for a given file:

- **First Search**: If this is the first request for the file, our method is different from the APS method, because it emphasizes the specific file type when randomly selecting the nodes to which the query will be sent.
- **Next Search**: If this is not the first request for the file, its path has been indexed in the index table.

If a node receives a request message for a file, it searches in its local database (this is similar to the local information table in [15]) for the requested file. If the file exists, the node sends a response message by generating a forward ant at starting time T_{start} to the querying node, along the reverse of the path that the file request traveled by generating the backward ant. The node considers the TTL parameter T_{final} and also the number of walkers, as in [21], in searching the network, starting by searching its available neighbors. T_{final} and the number of walkers serve to prevent ants from infinitely running forward in the network. Our method uses ICMP packets to avoid network congestion and prevent the possibility that a requested service is not available or that a host or neighbor peers cannot be reached, so that only available peers are visited in our query search. In the sequel, if the file does not exist, the node forwards the request to its neighbors by its forward ant. We use a common policy for preventing the forward ant from engaging in a cycle or loop when searching: If a forward ant detects a cycle occurring (i.e., it is about to search the nest peers), the forward ant is forced to return to an already visited node, the cycle's nodes are popped from the ant's stack, all memory about them is destroyed, and the forward ant continues searching. But if the cycle lasts longer than the T_{final} value of the forward ant, it is destroyed as in [10, 15].

4.2 Neighbor selection phase

Neighbor selection for request forwarding is based on information in the nodes' index tables (illustrated in Section 3),



File	No	Score
*.3gp	0	0
*.mp3	0	0
*.doc	2	2

File	No	Score
*.3gp	4	4
*.mp3	1	1
*.doc	3	3

File	No	Score
*.3gp	1	1
*.mp3	0	0
*.doc	1	1

File	No	Score
*.3gp	2	2
*.mp3	4	4
*.doc	1	1

File	No	Score
*.3gp	1	1
*.mp3	0	0
*.doc	0	0

File	No	Score
*.3gp	1	1
*.mp3	2	2
*.doc	1	1

File	No	Score
*.3gp	0	0
*.mp3	2	2
*.doc	1	1

File	No	Score
*.3gp	3	3
*.mp3	0	0
*.doc	0	0

Fig. 4 Initial score distribution for nodes

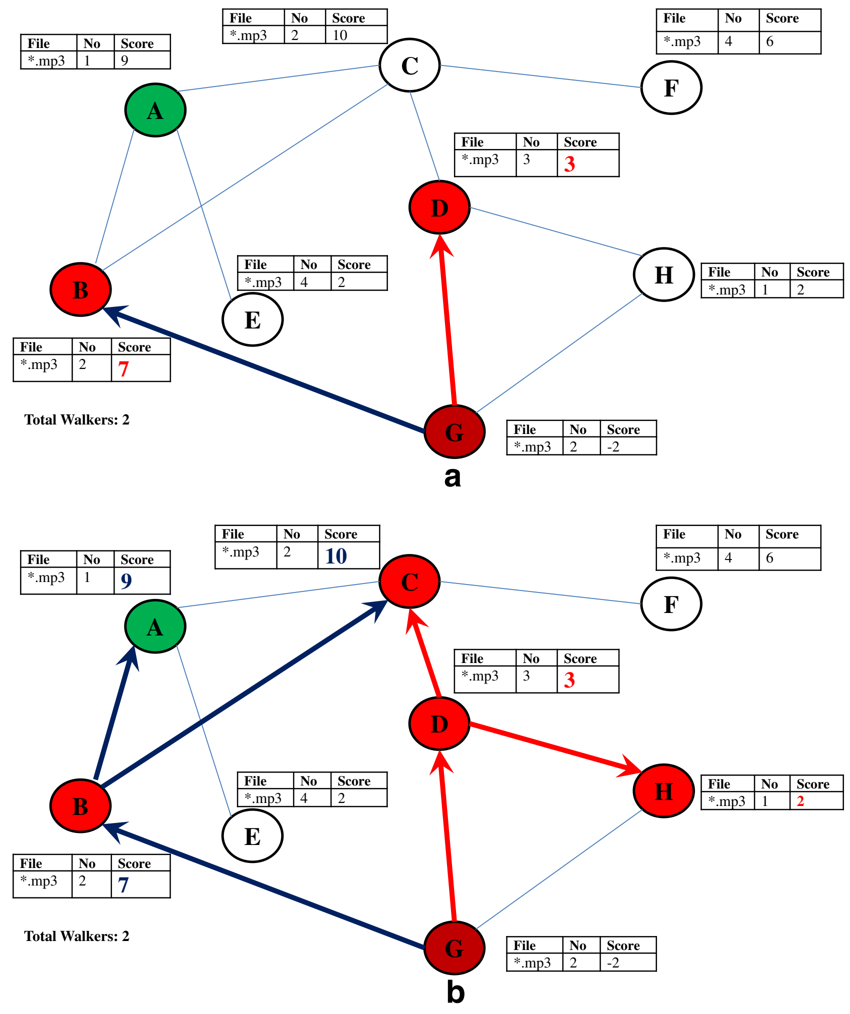


Fig. 5 Selecting routes in IAPS

such as the number of successful requests (file scores), file types, and the file-type probability table τ. The file-type probability table τ stores the probability rate for each peer, a value between 0 and 1, based on the file scores that it has. Table 2 shows the probability information that a node maintains about each file type.

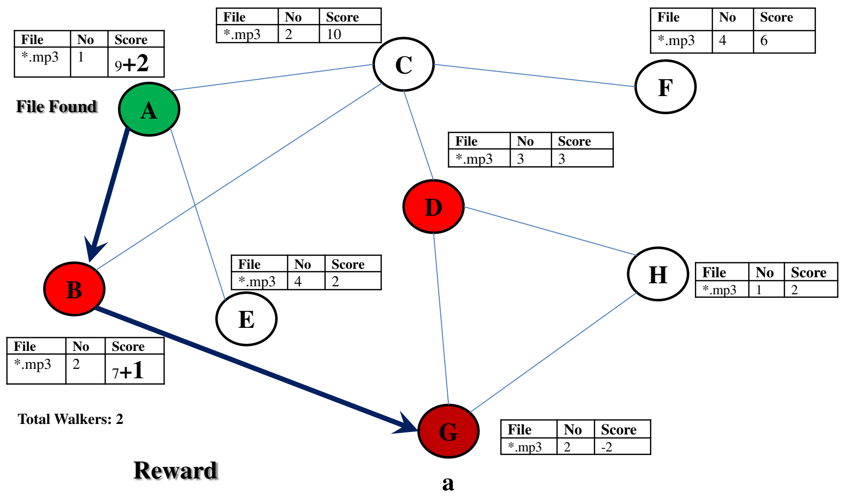


Fig. 6 Rewards and punishments in IAPS

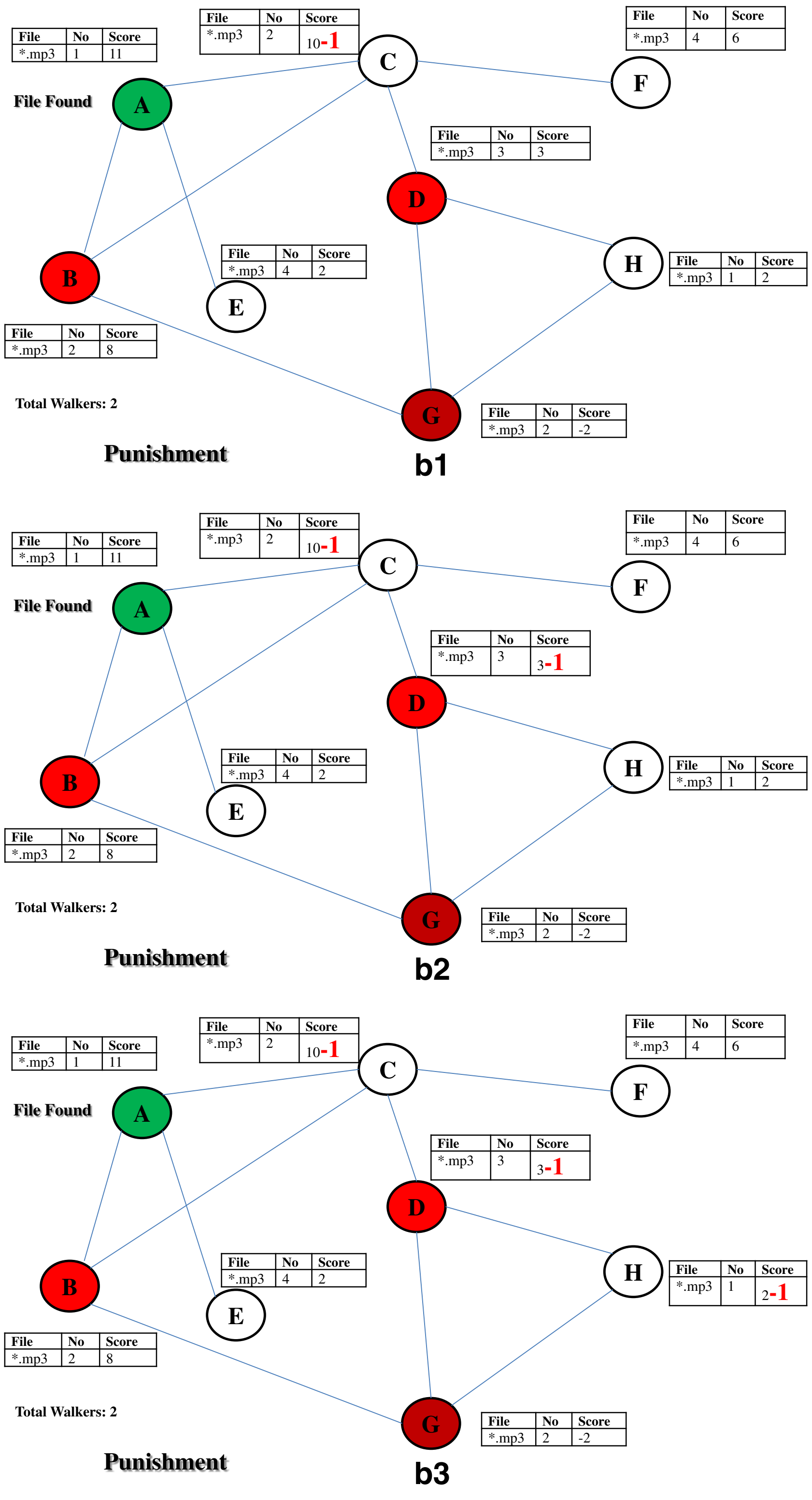


Fig. 6 (continued)

The querying node selects the neighbors that have the highest scores for the requested file type. Figure 2 describes the IAPS selection phase in a flowchart.

For example, if we want to find "mike.mp3," we identify the nodes that have high scores for files with .mp3 extensions. When a request is issued, we sort the nodes in descending order of the scores, using the Counting Sort algorithm [35]. We then select k nodes with the highest scores from the sorted neighbor list and send the request message to these nodes. This process is repeated for the unvisited and available neighbors of those nodes, and so on, until the requested file is found or all the walkers cross paths and the corresponding file is not found.

Figure 3 illustrates the process for five walkers. In Fig. 3a, node A wants to find "a.doc" file in a P2P network. First Node A searches its database. If it doesn't find the file, it will initiate a search through its neighbors (Fig. 3a). It selects the four nodes whose indexes have the highest scores for files with .doc extensions and sends them a request (Fig. 3b). In this case, node A sends a request to nodes D, F, G, and H (Fig. 3b).

4.3 Scoring phase

At the outset, each node is given a score of one point for each file of a given type that it contains in is database and zero points for file types that are absent. Figure 4 illustrates the initial scores for nodes before searching commences. Nodes F, G, and H have scores of zero for the file types that are not contained in their database.

When a node in P2P network sends a file request to its neighbors, the neighbors search their databases for the requested file. If a neighbor node finds the file in its database, it sends a reply message to the requesting node by generating a backward ant and increases its score for the file type by three points. Otherwise, the node sends a request for the file to k neighbors in T_{final}, where k is the number of walkers. If one of the neighbors finds the file in its database (index tables), the neighbor node will be rewarded with two points for the file type and updates Tables 1 and 2. In addition, the scores of the nodes along the path from the requesting node and the responding node are increased by one point excluding the requesting node (i.e., the forward ant has a stack into which it pushes visited peers; Reward). If the neighbor node does not have the requested file in its database, it will similarly send a request message to *its* unvisited neighbors by its forward ant. In the worst-case situation, if none of the nodes in the network have the requested file or the TTL (T_{final}) of all of the walkers reaches zero, the scores of all visited notes are decreased by one point (i.e., the backward ant performs a pop and punishes that visited peer; Punishment). On the other hand, if the search is successful, then the scores of all visited nodes that are not on the successful search path are also decreased by one point.

The scoring process is depicted below, in Figs. 5 and 6, for two walkers. In both figures, Node G requests "a.mp3" file that is has failed to find in its database. In Fig. 5a, Node G sorts its neighbors and selects two of these: nodes B and D. It sends request message to these nodes. Based on Fig. 5b, in the next walk both node B and node D select node C (because of its high score) and send C a file request. Node C sends the request to nodes F and A (i.e. Node C does not send the request message to Node D, because, it saves the previous request message already came from Node D in its database, so, no need to send request to Node D), and node A has the file. Figure 6 shows the steps involved in rewarding and deducting score points. All the nodes participating in the path connecting A to G are rewarded according to the policies we have presented (Fig. 6a). In contrast, all visited nodes (light red color nodes) *except* the nodes that were rewarded are punished under the scoring policy discussed above (Fig. 6b). Thus, a reward fortifies the location of the file extension in the network and enhances the probability of selecting that node for the next search (Fig. 6a). We use an ant colony to simulate the scoring phase in our programming, with a pheromone parameter that rewards a node based on the number of times per minute a file type is found on that node. Pheromone parameter is updated according to the information gathered by the forward ant by altering the routing table of each visited node similar to [36]. If a file type is requested more than 10 times per minute and is repeatedly found on one node, then the node is rewarded with three points the 11th time the file type is found there, and every node along the path from that node to the requesting node is rewarded two points (except of the requesting node-here node A) (Fig. 6a). For example (Figs. 5 and 6), if a file is searched for more than 10 times in a minute with the same results, node A is rewarded three points so that its new score will be 11+3=14 points, and other nodes along the successful route, such as node B in this example, are rewarded two points, so that node B's new score will be 7+2=9 (i.e. this figure is not shown here). Figure 6b-1 to b-3 show ants share the negative data. It gives negative points to the nodes that were participated as intermediate nodes to find the corresponding file until reach to the requested node.

Each peer applies the evaporation rule shown in (1) in a predefined interval T_e for each link to a neighbor peer, where the amount of evaporating pheromone is controlled by the parameter $\rho \in [0,1]$:

$$\tau_i = (1-\rho)\tau_i, \qquad (1)$$

Table 3 Simulation parameters and their default values

Simulation parameters	Default value
Number of nodes	1000
P2P model	Pure
Graph model	Random
Average node degree (AND)	8
Walkers deployed (k)	15
TTL	6
Number of objects (files)	150
Replication distribution	Zipf($a=0.82$)
Query distribution	Zipf($a=0.9$)
Number of Req. nodes	1000
Number of queries per request node	3162

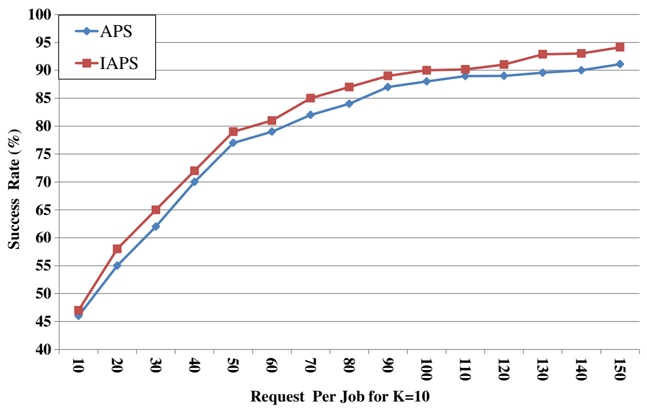


Fig. 7 Success rates for K=10, TTL=6, Object=150

where τ_i is the file-type probability table for node i.

4.4 Flow control phase

The flow control phase is used to control and limit the number of messages received at each time step based on the nodes' abilities and capabilities. We use a least-recently-used (LRU) scheme to control the flow; this involves throwing away some unimportant messages by ICMP controlling packets and thus increasing the network's capacity to serve important new messages in the near future.

5 Performance evaluation

In this section, we compare IAPS with the standard APS and Random Walk algorithms. We use random graph models to simulate the network. Before each simulation, object replication and query distributions and query tables are set. We used PeerSim software http://peersim.sourceforge.net/ for the experiment. PeerSim is written in JAVA and runs on the Linux OS. It is composed of two simulation engines, a simplified (cycle-based) one and event driven one. The engines are supported by many simple, extendable, and pluggable components, with a flexible configuration mechanism http://peersim.sourceforge.net/. It is suitable for evaluating P2P protocols based on discrete event simulation. PeerSim includes the following parameters:

- *Node*: this specifies the peer nodes that can be geographically identified;
- *CDprotocol*: this specifies the protocol used in the simulation;
- *Linkable* :this specifies the connectivity between nodes;
- *Control*: this specifies various simulation parameters.

To evaluate a protocol's performance, this algorithm is designed to work in a specific environment. In our simulation, we set the number of peers to 1000. The nodes' end-to-end delay averages 200 msec [37]. Each node generates a search request for the location of a random key with an average search time of 3000 msec, and each node contacts an average of 8 nodes. The maximum number of walkers is 15 (we have tested all the methods mentioned for 1 to 15 walkers), the TTL (T_{final}) is set to 6, and control parameter ρ is 0.1. Moreover, various parameters are altered in the simulations so that they can be evaluated. We considered the following metrics in the simulation:

- *Success rate*: the ratio of the number of hits to the total number of requests;
- *Hit per query*: the average number of discovered objects per query;
- *Response Distribution*: the response distribution with the number of walker steps for requests.
- *Duplicate Message*: the number of duplicate messages generated while searching for the file.

Table 3 summarizes our simulation parameters and their default values. We used 150 objects in most simulations for simplicity and speed; increasing that number does not affect

Fig. 8 Success rates

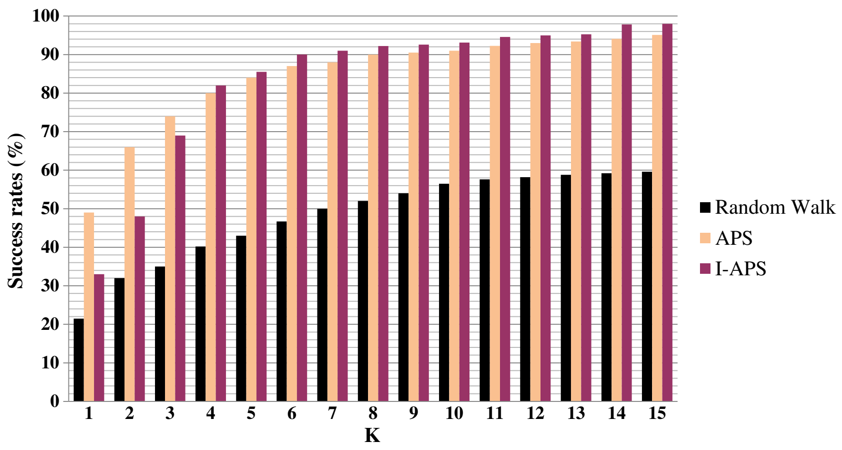
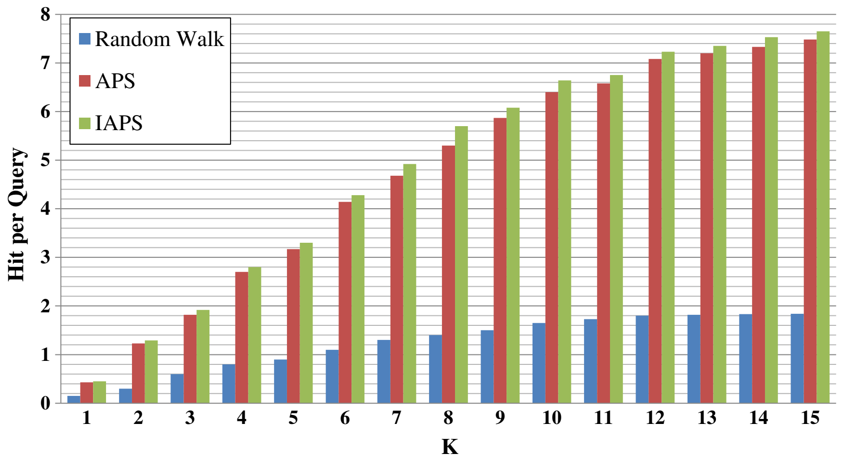


Fig. 9 Hit per query



the quality of the results. Our query and replication strategies generally follow the observations in [32], but we preferred a less skewed distribution, where the objects in the 90th percentile of the rankings constitute approximately 30 % of the total number of stored objects and receive about 30 % of the total requests. For the 150 objects in our model this was a more realistic choice and resulted in requests for all of the 150 objects. We used three file extensions: .3GP, .mp3, and .doc, with file size generally less than 5 MB. For each file extension, we included 1000 files distributed in a uniform manner among the nodes in the network.

5.1 Success rates

Figure 7 displays the number of requests affects accuracy between APS and IAPS. We have tested our works in k=10 and TTL=6, we varied the number of requests per object using a uniform distribution for both storage and requests on the default graph. As we can see, accuracy improves with only a small increase in requests, even though only about 150 copies of each object exist. The average success rate for APS is 78.5 % but IAPS is approximately 81 %.

Figure 8 presents the success rates for the three algorithms as a function of the number of walkers deployed (k). When $k<4$, the success rate for IAPS is lower than for APS, mainly because IAPS gathers information from all nodes in this case. However, as K increases, the success rate for IAPS surpasses of the success rate for APS. In Fig. 8, we can see that the success rate of IAPS increases with k and reaches close to 97 %, outperforming both APS and the Random Walk methods.

As the numbers of requests increase, APS's success rate also increases relative to the Random Walk method. IAPS is smarter than the other methods because it acquires information based on previous requests and uses this information to transmit requests to the nodes that have a higher probability of successfully responding to the requests. As a result, IAPS's success rate is better than those of the other two methods.

5.2 Average number of discovered objects (Hit per query)

Figure 9 shows the average number of discovered objects per query. APS puts the walkers to much better use, achieving greater than 65 % walker accuracy (the ratio of successful walkers to all walkers deployed), and it discovers approximately four times more objects than the Random Walk method. This is an immediate side effect of its high success rate and very few walker collisions. Since walkers are directed to (possibly) different parts of the network where different copies of an object may exist, the successful walker percentage increases. This is extremely important for currently popular P2P applications, giving the user a much broader choice for downloading files. IAPS produces marginally better results than APS.

5.3 Response distribution (Distance rate per steps)

Figure 10 compares the response distribution with the number of walker steps for requests. The random walk

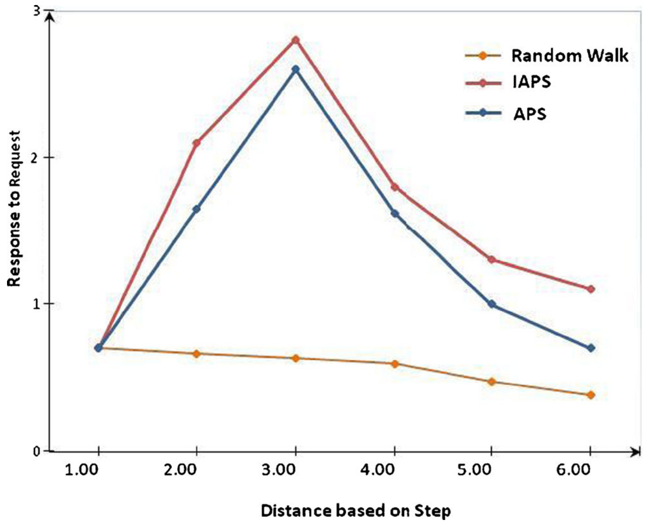


Fig. 10 Distance based on walker steps

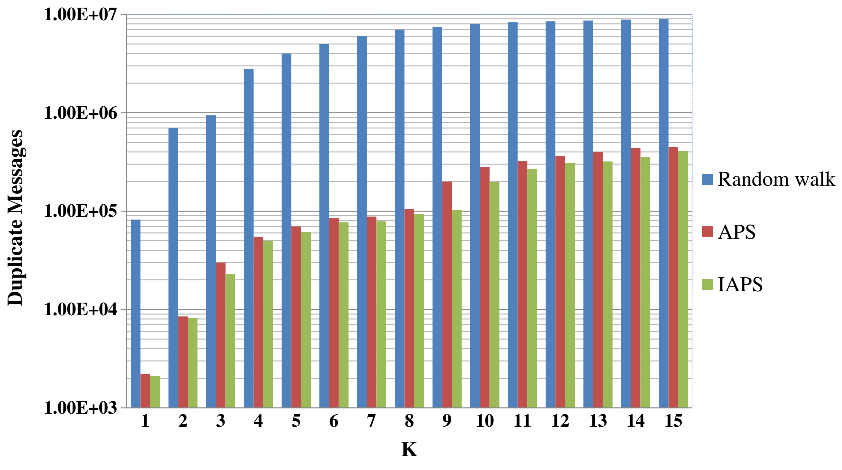


Fig. 11 Duplicate messages

has an approximately smooth and straight line that shows discovered goals between one and six steps, equal to the TTL value. APS increases the discovered goals to half of the TTL. In contrast, IAPS finds the goals in shorter distances than both of the other methods. As a result, the number of requests that need to be sent for a discovery is less in IAPS. Both APS and IAPS are useful only for up to three steps, after which their efficiency quickly decreases, with APS's efficiency decreasing more rapidly than IAPS's.

5.4 Duplicate messages

Figure 11 displays the experimental results for the duplicate message metric. We can see a vast reduction of duplicate messages in IAPS, which reduces wasted bandwidth. Duplicate messages are considered to be failure states for our walkers, and hence the learning process makes adjustments in order to minimize walker collisions. The APS and IAPS methods produce several messages in the update processing, in contrast with the Random Walk method. Most (70 %) of the request messages in the Random Walk method are failed messages, and a few of them continue their walk until their TTL reaches zero. This imposes a large traffic load on the network. In APS, there are approximately 4 distinct messages for each 15 that are circulating, while IAPS has a better strategy for update processing. APS consistently outperforms the Random Walk, producing one to two orders of magnitude fewer duplicate messages. This is also important because it increases each peer's useful processing time. As we can see in Fig. 11, IAPS produces close to 18 % fewer duplicate messages than APS.

At last, In Table 4 we conclude our results on pure P2P networks. We compare IAPS with Random Walks, and APS over three different topologies: The default one (Zipf), Uniform graph, and a 5000-node random graph with average degree 8. We monitor the success rate, the average number of hits, and duplicate messages per query. Apart from the default setup, we also test the algorithms using the uniform distribution for both requests and storage. This may be a more suitable model for other kinds of P2P applications, for example sharing of sensor data between wireless ad-hoc peers. The replication ratio is set to 1 % and each object is requested 30 times. We clearly notice that IAPS can greatly benefit from such a setup, delivering over 94 % in success rate and discovering 4 times more results than Random Walks. Our simulations on the 5000-node random graph justify our prediction that the graph size cannot influence the performance of IAPS and APS. The results were a little worse from the ones in the original graph (Zipf); because the quality of

Table 4 Results for three network graphs: Zipf, Uniform, and 5000-node random graph

Methods	IAPS			APS			Random Walks		
	Succ. Rate (%)	Average Hits	Dupl. Msg	Succ. Rate (%)	Average Hits	Dupl. Msg	Succ. Rate (%)	Average Hits	Dupl. Msg
Zipf	93	6.8	0.06	88	6.1	0.04	55.5	1.35	1.6
Uniform	94.8	7.15	0.07	90	6.8	0.06	40	0.6	2.3
5000-node random graph	88.3	5.8	0.01	85	5.1	0.01	58	1.3	1.4

the new graph was worse (many more disconnected components were present). Still, our method is over 30 % more accurate and delivers 4 times as many results as the random method. In all these simulations, we notice that IAPS produces almost the same number of messages with Random Walks, wasting at most 4 extra messages per search.

6 Conclusions and future research directions

In an unstructured P2P network such as Gnutella, there is no rule that strictly defines where data are stored. To search for a specific item, flooding and its variations are frequently used. In this paper, we have proposed a new approach, the Improved Adaptive Probabilistic Search (IAPS). We can *summarize the benefits* of IAPS as follows:

- **Dynamic Topology**: IAPS is based on multi-agent systems, and the walkers are independent. This feature provides consistency for current network topologies.
- **Perception**: In IAPS, information is exchanged between different neighbor nodes, so that nodes that have a higher probability of successfully responding to a request can be chosen.
- **Link Quality**: Nodes can easily be rewarded or punished based on their responses to requests.
- **High Efficiency**: Whenever a node leaves the network under IAPS, the node is inactivated and the paths that include this node are eliminated from searching. In contrast, whenever a new node is added to the network, it is added to some paths and its database is used for information retrieval.
- **Score Balancing**: One node might precipitate successful responses to some requests for a period but fail to response to requests made after this period. IAPS can decrease its scores gradually. Our simulation results show that IAPS outperforms the original Adaptive Probabilistic Searching (APS) algorithm with respect to most evaluation metrics, especially the query success rate and hits per query.

However, unlike a programming system which is closed, an unstructured P2P network's topology structure is in practice fairly ad hoc. Peers join and leave the system frequently. This will significantly impact the score system, which partially relies on the P2P network structure. The study in this paper can be improved by taking this behavior into account. Therefore, in the future, we plan to make some structural changes to IAPS and test its usefulness in dynamic systems and also in structured P2P networks.

References

1. Iskandar I, Naomie S (2009) Selective flooding based on relevant nearest-neighbor using query feedback and similarity across unstructured peer-to-peer networks. Journal of Computer Science 3(5):184–190, ISSN 1549–3636
2. Balakrishnan H, Kaashoek MF, Karger D, Morris R, Stoica I (2003) Looking up data in P2P systems. Communications of ACM 46(2): 43–48
3. Barkai D (2002) Technologies for sharing and collaborating on the net. *Proceeding of the 1st International Workshop on Peer-to-Peer Computing* (*IPTPS'02*), ISBN: 0-7695-1503-7, 13–28. doi: 10.1109/P2P.2001.990419
4. Daswani N, Garcia-molina H, Yang B (2003) Open problems in data-sharing peer-to-peer systems. *Proc. of the 9th International Conference on Database Theory* (*ICDT'03*) 1–15
5. Milojicic DS, Kalogeraki V, Lukose R, Nagaraja K, Pruyne J, RichardB, Rollins M, Xu Z (2002) Peer-to-peer computing, HP Lab technical report, HPL-2002-57 www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf
6. Lv Q, Cao P, Cohen E, Li K, Shenker S (2002) Search and replication in unstructured peer to-peer network's. *Proceeding of the 16th ACM International Conference on Supercomputing* (*ACM ICS'02*) 258–259. doi: 10.1145/514191.514206
7. Stoica I, Morris R, Karger D, Frans M, Kaashoek, Balakrishnan H (2001) Chord: A scalable peer-to-peer lookup service for internet applications. *Proceeding of the 2001 ACM Annual Conference of the Special Interest Group on Data Communication* (*ACM SIGCOMM'01*) 149–160. http://pdos.csail.mit.edu/6.824/papers/stoica-chord.pdf
8. Clarke I, Sandberg O, Theodore BW, Hong W (2001) Free net: A distributed anonymous information storage and retrieval system. *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability* 46–66. www.cs.cornell.edu/people/egs/615/freenet.pdf
9. Manku GS, Bawa M, Raghavan P (2003) Verity Inc, Symphony: Distributed hashing in a small world. *Proceeding of 4th USENIX Symposium on Internet Technology and Systems* (*USITS'03*) 127–140. www.infolab.stanford.edu/~bawa/Pub/symphony.ps
10. Dorigo M, Gambardella LM (1997) Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1):53–66
11. Caro GD, Dorigo M (1998) AntNet: Distributed stigmergy control for communications networks. Journal of Artificial Intelligence Research 9:317–365
12. Babaoglu O, Meling H, Montresor A (2002) Anthill: A framework for the development of agent-based peer-to-peer systems. *In Proceedings of the 22nd International Conference on Distributed Computing Systems*
13. Wu C, Yang K, Ho J (2006) AntSearch: An ant search algorithm in unstructured peer-to-peer networks. *In Proceedings of the 11th IEEE Symposium on Computers and Communications*
14. Michlmayr E (2006) Ant algorithms for search in unstructured peer-to-peer networks, Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW '06). IEEE Computer Society, Washington, pp 142–146
15. Tang D, Lu X, Yang L (2011) ACO-based search algorithm in unstructured P2P Network. In Proceedings of the 2011 International Conference of Information Technology, Computer Engineering and Management Sciences, 1 (ICM '11), IEEE Computer Society, Washington, 143–146
16. Tsoumakos D, Roussopoulos N (2003) Adaptive probabilistic search in peer-to-peer networks. Technical Report, CS-TR-4451
17. Tsoumakos D, Roussopoulos N (2003) Adaptive probabilistic search for peer-to-peer networks. *Proceedings of the 3rd International Conference on Peer-to-Peer Computing* (*P2P 2003*) 102–109

18. Huo Q, Chen J, Xu X, Zhou Y, Liu X (2011) A location-aware efficient content-based searching over unstructured P2P network. International Conference on Network Computing and Information Security (NCIS) 2:183–187. doi:10.1109/NCIS.2011.135
19. Rhea SC, Kubiatowicz J (2002) Probabilistic location and routing. Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02) 3:1248–1257. doi:10.1109/INFCOM.2002.1019375
20. Yang B, Garcia-Molina H (2002) Improving search in peer-to-peer network's. *Proceeding of the 22nd IEEE International Conference on Distributed Computing* (*IEEE ICDCS'02*) 5–14, doi: 10.1109/ICDCS.2002.1022237
21. Jawhar I, Wu J (2004) A two-level random walk search protocol for peer-to-peer networks. *Proceeding of the 8th World Multi-Conference on Systemic*, *Cybernetics and Informatics* 1–5. doi: www.faculty.uaeu.ac.ae/ijawhar/publications/randw_S942DB.pdf
22. Tigelaar AS, Hiemstra D, Trieschnigg D (2012) Peer-to-peer information retrieval: An overview. ACM Transactions on Information Systems (TOIS) 30(2):1–34. doi:10.1145/2180868.2180871
23. Crespo A, Garcia-Molina H (2002) Routing indices for peer-to-peer systems. *Proceeding of the 22nd International Conference on Distributed Computing Systems* (*IEEE ICDCS'02*) 23–32. doi: 10.1109/ICDCS.2002.1022239
24. Himali DMR, Prasad SK (2011) SPUN: A P2P Probabilistic search algorithm based on successful paths in unstructured networks. *IEEE International Symposium on Parallel and Distributed Processing Workshops* 1610–1617. doi: 10.1109/IPDPS.2011.316
25. Kalogeraki V, Gunopulos D, Zeinalipour-Yazti D (2002) A local search mechanism for peer-to-peer networks. *Proceedings of the 11th ACM Conference on Information and Knowledge Management* (*ACM CIKM'02*) 300–307. doi: 10.1145/584792.584842
26. Yang C, Li X (2005) Dominating-set-based searching in peer-to-peer networks. International Journal of High Performance Computing and Networking 3(4):205–210. doi:10.1504/IJHPCN.2005.008562
27. Wu Y, Izumi T, Ooshita F, Kakugawa H, Masuzawa T (2007) An adaptive randomized search protocol in peer-to-peer systems, Proceedings of the 2007 ACM symposium on Applied computing 533–537. doi: 10.1145/1244002.1244126
28. Delbru R, Campinas S, Tummarello G (2012) Searching web data: An entity retrieval and high-performance indexing model. Elsevier Web Semantics: Science, Services and Agents on the World Wide Web archive 10:33–58. doi:10.1016/j.websem.2011.04.004
29. Liu M, Koslela T, Ou Z, Zhou J, Riekki J, Ylianttila M (2011) Super-peer-based coordinated service provision. Journal of Network and Computer Applications archive 34(4):1210–1224. doi:10.1016/j.jnca.2011.01.007
30. Lee P, Jayasumana AP, DilumBandara HMN, Lim S, Chandrasekar V (2012) A peer-to-peer collaboration framework for multi-sensor data fusion. Journal of Network and Computer Applications archive 35(3):1052–1066. doi:10.1016/j.jnca.2011.12.005
31. Association for Computing Machinery (1998) ACM computing classification system (ACM CCS)
32. Ripeanu M, Foster I, Iamnitchi A (2002) Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal* 6
33. Ramzan N, Park H, Izquierdo B (2012) Video streaming over P2P networks: Challenges and opportunities. Image Communication Journal 27(5):401–411. doi:10.1016/j.image.2012.02.004
34. Androutsellis-Theotokis S, Spinellis D (2004) A survey of peer-to-peer content distribution technologies. ACM Computing Surveys (CSUR) Journal 26(4):335–371
35. Cormen TH, Leiserson ChE, Rivest RL, Stein C (2009) Introduction to algorithms, Second edition, MIT Press, ISBN 0-262-03384-4 168–171
36. Michlmayr E (2006) Ant algorithms for search in unstructured peer-to-peer networks, *Proceedings of ICDE*
37. Marti S, Ganesan P, Molina HG (2004) DHT routing using social links. *Proceeding of the 3rdInternational Workshop on Peer-to-Peer Systems* (*IPTPS'04*)

Mohammad Shojafar received his B.S. in Computer Engineering-Software major at Iran University Science and Technology, Tehran, Iran (2001-2006) and M.sc. at Qazvin Islamic Azad University, Qazvin, Iran (2007-2010). He is currently a Ph.D. student in Information and Communication Engineering at DIET Dept. of the "La Sapienza" University of Rome. He is an author/co-author of 25+ peer reviewed publications in Springer, Elsevier and IOS press Publishers. Also, Mohammad was a Programmer and Analyzer in Exploration Directorate Section at N.I.O.C in Iran from 2012-2013. His current research focuses on wireless communications, distributed computing and optimization. Email: shojafar@diet.uniroma1.it, WWW.mshojafar.com

Jemal H. Abawajy is a faculty member in the School of Information Technology, Deakin University, Australia. He is actively involved in funded research in robust, secure and reliable resource management for pervasive computing (mobile, clusters, enterprise/data grids, web services) and networks (wireless and sensors) and has published more than 150 research articles in refereed international conferences and journals and books. He is currently the principal supervisor of 13 PhD students and co-supervising 3 PhD students. Prof. Abawajy is on the editorial board of several international journals. Prof. Abawajy has been a member of the organizing committee for over 100 international conferences serving in various capacity including chair, general co-chair, vice-chair, best paper award chair, publication chair, session chair and program committee. Email: jemal.abawajy@deakin.edu.au

Zia Delkhah Received His Bsc in Computer Engineering-Software major Sari Islamic Azad University, Sari, Iran (2001–2006) and Msc in Information Technology major at Qazvin Islamic Azad University, Qazvin, Iran (2007–2010). His Speciality is Peer to Peer systems and distributed systems. He has published two papers in IEEE conferences till now. He is a instructor of Sari Islamic Azad University now. Email: ziadelkhah@gmail.com

Ali Ahmadi Received His Bsc in Computer Engineering-Software major Sari Islamic Azad University, Sari, Iran (2001–2006) and Msc in Information Technology major at Qazvin Islamic Azad University, Qazvin, Iran (2007–2011). His Speciality is Wireless sensor network and distributed systems. He has published two papers in IEEE conferences till now. He is an instructor of Sari Islamic Azad University now. Email: ahmadiali83@gmail.com

Zahra Pooranian received her Msc in Computer Architecture degree as honor student in Dezful Islamic Azad University since 2011. She is an instructor in Sama University in Dezful and Ahvaz since 2009. Her research interest in Grid computing specially in resource allocation and scheduling. She has worked on several papers in decreasing time and makespan in grid computing by using several AI methods such as GA, GELS, PSO, and ICA. She has published more than 8 papers, 2 Springer journals in grid scheduling and resource allocation, and various conferences, such as ICCCIT'11, ICEEE'11 and TSP'13. Email: Zahra.pooranian@gmail.com

Ajith Abraham's received the Ph.D. degree in Computer Science from Monash University, Melbourne, Australia. He is currently the Director of Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, USA, which has members from more than 100 countries. He has a worldwide academic and industrial experience of over 23 years. He works in a multi disciplinary environment involving machine intelligence, network security, and various aspects of networks, e-commerce, Web intelligence, Web services, computational grids, data mining, and their applications to various real-world problems. He is an author/co-author of 900+ peer reviewed publications, h-index 54 and has over 12,000+ citations. He has also given more than 60 plenary lectures and conference tutorials in these areas. Since 2008, he is the Chair of IEEE Systems Man and Cybernetics Society Technical Committee on Soft Computing and a Distinguished Lecturer of IEEE Computer Society representing Europe (since 2011). Dr. Abraham is a Senior Member of the IEEE, the Institution of Engineering and Technology (UK) and the Institution of Engineers Australia (Australia), etc. He is the founder of several IEEE sponsored annual conferences, which are now annual events. More information at: http://www.softcomputing.net and Email: Ajith.Abraham@ieee.org