ISRL: Intelligent Search by Reinforcement Learning in Unstructured Peer-to-Peer Networks

Xiuqi Li and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, Florida 33431
Shi Zhong
SDS-Data Mining Research
Yahoo! Inc.
Sunnyvale, California 94089

Abstract

Existing searches in unstructured peer-to-peer networks are either blind or informed based on simple heuristics. Blind schemes suffer from low query quality. Simple heuristics lack theoretical background to support the simulation results. In this paper, we propose an intelligent searching scheme, called ISRL (Intelligent Search by Reinforcement Learning), which systematically seeks the best route to desired files by reinforcement learning (RL). In artificial intelligence, RL has been proven to be able to learn the best sequence of actions to achieve a certain goal. To discover the best path to desired files, ISRL not only explores new paths by forwarding queries to randomly chosen neighbors, but also exploits the paths that have been discovered for reducing the cumulative query cost. We design three models of ISRL: a basic version for finding one desired file, MP-ISRL for finding at least k files, and C-ISRL for reducing maintenance overhead through clustering when there are many queries. ISRL outperforms existing searching approaches in unstructured peer-to-peer networks by achieving similar query quality with lower cumulative query cost. The experimental result confirms the performance improvement of ISRL.

This work was supported in part by NSF grants ANI 0073736, EIA 0130806, CCR 0329741, CNS 0422762, CNS 0434533, and CNS 0531410. Contacts: Xiuqi Li and Jie Wu: {xli, jie}@cse.fau.edu; Shi Zhong: frank.shizhong@gmail.com.

Index Terms– Hint-based search, peer-to-peer networks, reinforcement learning, unstructured peer-to-peer networks.

I. INTRODUCTION

File sharing peer-to-peer (P2P) networks have been very popular in recent years. They offer the benefit of harnessing the tremendous storage resources among computers on the Internet. In such networks, each node shares some of its local files with other nodes. All nodes play equal roles and act as servers for each other. P2P networks are overlay networks. Each overlay link is a virtual link mapped to a sequence of links in the underlying network. P2P networks are also dynamic. Nodes may join and leave freely. In an unstructured P2P, the overlay topology and data location do not follow restrictive rules, which makes this type of P2P the most widely used one on the Internet [1].

Searching is the basic operation in file sharing P2Ps. Many searching approaches in unstructured P2Ps have been proposed in the literature, and can be classified as *blind* or *informed*. In a blind search, no hints exist and queries are blindly forwarded, e.g. the random walk approach [2]. In an informed search, such as routing indices [3], nodes utilize hints to facilitate query forwarding. Informed schemes usually achieve a higher search quality than blind ones.

Existing informed approaches can be categorized into either *proactive* or *reactive*. Proactive approaches, such as routing indices [3], propagate hints before files are queried whereas reactive approaches, for example intelligent search [4], collect hints after query processing. Reactive approaches do not waste resource in propagating hints about unpopular files. Routing indices spread hints about document topics. Intelligent search collects past similar queries. All these informed schemes employ simple heuristics and lack theoretical support for their effectiveness. In addition, reactive approaches do not try to improve the quality of collected hints.

In this paper, we propose to systematically learn the best route to desired files through reinforcement learning [5]. RL addresses the general issue of how a learner that interacts with its environment can learn the optimal actions to achieve its goal. The most distinguishing features of RL are trial-and-error learning and the delayed reward mechanism. Each time an agent takes an action, it gets an immediate reward and the environment goes from one state to another. The learner is not instructed which actions to take. Instead it must discover which actions yield the

highest reward by trying them. In most cases, an action may affect both the immediate reward and all future rewards. Theoretical results prove that RL converges to the best sequence of actions with maximum cumulative reward.

When RL is applied to P2P searching, each node is a distributed learner and the P2P network is its environment. An action is a query forwarding. Each query path is a sequential process. The goal is to reach a node that hosts a desired file. All nodes (learners) work together to learn the best query path to hosting nodes. Each learner iteratively estimates which next-hop neighbor is the best one to forward a given query to by trying them. When a query is successfully resolved, rewards are propagated along the reverse query paths. After receiving rewards, each node updates its estimation to reinforce better paths. Each learner can also adapt itself to the network dynamics through this trial-and-error mechanism.

RL has been used in [6] to adapt P2P topologies to peer interests during searching. This paper applies RL to P2P searching without topology adaptation. We propose three models: basic ISRL, MP-ISRL, and C-ISRL. The basic ISRL (Intelligent Search by Reinforcement Learning) is targeted at locating one desired file efficiently. Each node learns the best path to one desired file. The learning process includes exploring new potential paths and exploiting the already discovered paths. To balance exploration and exploitation, each node adjusts its exploration coarsely or in a fine-grained manner based on the quality of the learned paths. MP-ISRL, where MP stands for Multiple Paths, aims at finding k files efficiently. Each node learns the top-k best paths to desired files. Paths are evaluated by costs or discounted rewards. C-ISRL, where C stands for Clustering, is designed to reduce the storage consumption when there are many semantic queries. The saving is achieved by clustering similar queries.

We make the following contributions in this paper. First, we propose a searching scheme, ISRL, that systematically learns the best path to a desired file by reinforcement learning and adapts itself to system dynamics. To the best of our knowledge, this is the first work that applies RL to P2P searching without topology adaptation. Second, we present three models of ISRL, the basic version for locating the best path, MP-ISRL for finding top k best paths, and C-ISRL for processing semantic queries more efficiently. Third, we discuss different trade-offs in designing these protocols among contradictory objectives, such as balancing exploration and exploitation. Fourth, we conduct extensive simulations and compare results with existing searching schemes.

The remainder of this paper is organized as follows. In Section 2, we review the related searching schemes in unstructured P2Ps and reinforcement learning. In Section 3, we introduce the basic version of ISRL. In Sections 4 and 5, we describe two extensions, MP-ISRL and C-ISRL respectively. In Section 6, we discuss other potential extensions of ISRL. Simulation results are presented in Section 7. Our work is summarized in Section 8.

II. Related work

In this section, we review searching schemes in unstructured P2Ps and reinforcement learning.

A. Blind search

Blind search includes flooding, random walk, and their variations. Flooding is a Breadth First Search (BFS) of the overlay graph with depth limit D. Gnutella [7] directly applies flooding. Its variation in [8], DiCAS [9] and local indices [10] employ different grouping schemes to restrict flooding among a subset of nodes in the P2P. The work in [11] [12] improves flooding by building P2P overlays that match the physical underlying networks. Iterative deepening [4] is a sequence of BFS searches with increasing depth limits.

Random walk [2] and its variations intend to reduce message redundancy in flooding. k-walker random walk [2] deploys k random walkers at the querying source and each walker is forwarded to a random neighbor thereafter. Modified random BFS [4] forwards a query to a random subset of neighbors. Two-level random walk [13] first deploys k_1 random walkers with the TTL being l_1. When the TTL expires, each walker forges k_2 random walkers with the TTL being l_2. Dominating-set-based search [14] applies random walk on a connected dominating set of nodes in the P2P overlay. The work in [15] runs random walk or flooding on the set of super-peers, which divide the entire overlay into disjoint clusters.

B. Informed search

Informed searches are classified into reactive and proactive. Directed BFS [10], intelligent search [4], CIS [16], adaptive probabilistic search (APS) [17], GIA [18], and the approaches in [19] and [20] are reactive approaches. Hybrid [21], routing indices [3], scalable query routing (SQR) [22], SETS [23], and ESS [24] are proactive approaches.

Directed BFS directs BFS searches based on simple heuristics such as the highest message count. Intelligent search directs a query to a subset of neighbors that answered similar queries previously. CIS sends queries similarly among the maximum independent set of the P2P overlay. APS forwards a single file lookup query probabilistically based on the past query executions and the guesses of query sources. APS search can be viewed as an ad-hoc application of reinforcement learning in specific single file lookup queries. Our ISRL approach is more general and theoretically sound.

GIA considers nodes with heterogeneous capacities. It uses capacity information to forward more queries to nodes with higher capacities. The approach in [19] tries to achieve load balance by utilizing the last node visited as a cursor pointing to where the next forwarding starts. The scheme in [20] considers applications where two peers can communicate with each other via two paths, the default path determined by IP routing, and an alternate overlay path through a relay node selected among other peers. AS-level path information is used to select the alternate path that is most disjoint from the default one.

Hybrid probes and forwards queries based on simple heuristics such as the number of files. In routing indices, each node proactively maintains information about document topics that may be found through each neighbor. A query is forwarded to the neighbor with the highest probability that is computed based on the current keyword query and the document information kept for each neighbor. SQR uses a data structure called exponential decay bloom filter (EDBF) to proactively propagate information about files. During the propagation, the amount of information decays exponentially with increasing distance from the hosting node. A query is dispatched to the best neighbor that has the maximum amount of information according to EDBFs.

SETS builds an additional semantic overlay on top of the P2P overlay. The semantic overlay is a small-world graph. ESS adjusts the unstructured P2P overlay to a semantic overlay by proactively seeking nodes with similar contents and nodes with dissimilar contents.

C. Reinforcement learning

There are many forms of reinforcement learning. Q-learning [5] is the most popular one because of two factors. In Q-learning, a learner does not need a model for either learning or action selection. The convergence of the Q-learning algorithm is not affected by the details of the

exploration strategy. Therefore, we chose Q-learning in this paper. In Q-learning, a Q function is defined for each state-action pair. The value $Q(s, a)$ for state s and action a is the expected discounted cumulative reward obtained by taking action a at state s and following an optimal policy thereafter. A policy is a mapping from states to actions or to probability distributions over actions. When the learning terminates, the optimal action from any state is the one with the highest Q value. A learner estimates $Q(s, a)$ values iteratively based on experiences as follows. At current state s, select an action a and execute it. After receiving immediate reward r and observing the new state $s^{'}$, the learner updates $Q(s, a)$ based on this experience according to the following formula:

$$Q(s, a) = (1 - \eta)Q(s, a) + \eta(r + \gamma max_b Q(s^{'}, b)),$$

where η is the learning rate and γ is the discount factor. Both are in the range $[0, 1]$. $Q(s, a)$ can be implemented as a simple table or a trainable parameterized function.

Reinforcement learning has been applied in P2Ps to solve problems other than efficient searching. In [25], RL was used to design a payment method for motivating peer cooperation.

III. BASIC ISRL

The basic ISRL search is designed to find one desired file. The goal of basic ISRL is to deliver a query through the best path to a node hosting the desired file. The best path is the one with the lowest cost in terms of the overlay hops. To reach its goal, with probability p_q, basic ISRL forwards query q to a random neighbor for the purpose of exploring a potential better path. With probability $1 - p_q$, a query is sent through the best path currently known. The newly explored better path replaces the existing path during path updates. We consider semantic queries for files with similar semantic content. It can also be applied to single file replica lookup by ID without much modification. In this section, it is assumed that all peers have the same service capacities. Section VI will discuss how to handle peers with heterogenous capacities.

A. Semantic content representation

We adopt the vector space IR model to represent the semantic content of documents and queries. In this model, each document is represented by a semantic vector of term weights. Each dimension

of the vector corresponds to a term that appears in a document. The weight of a term indicates its importance in describing the semantic content of a document. If a term does not appear in a document, the weight is 0. The number of dimensions in a semantic vector corresponds to the size of the vocabulary for the document collection. To reduce dimensions, stop words are excluded from the vocabulary. All words that have the same root word are condensed to their root word (stemming). A user can issue a semantic query described in a natural language. The system extracts a semantic vector for such a query just like extracting a semantic vector for a document [26]. Each query vector or document vector is normalized such that its Euclidean vector norm is 1 before similarity computation.

Many term weighting schemes have been proposed in the IR literature [26]. We calculate the weight of a term t in the semantic vector of a document d, denoted by w_{td}, according to the following formula.

$$w_{td} = 1 + \log(f_{td}),$$

where f_{td} refers to the number of occurrences of t in d. This scheme does not require the global knowledge of the document collection in a P2P network. Moreover, this weighting approach has been demonstrated to be effective in document clustering [27].

B. Path entries

To facilitate learning the best path during query processing, each node, say x, keeps one path entry for each query vector that was resolved successfully through x. Table I lists the items in an entry. q represents the query vector. z is the best neighbor for finding files similar to q that x has discovered so far. Q_{xq} refers to the cost of the best-so-far path for resolving q, which corresponds to the Q value in Q-learning but minimized. p_q represents the probability of exploring new paths for q. $cntU_q$ counts the consecutive number of updates to Q_{xq} that are smaller than a threshold. These path entries are created when desired files are found the first time. They are continuously monitored during path exploration and update process, which will be discussed later in this section.

Notation	Meaning
q	The query semantic vector
z	The neighbor on the best-so-far path for q
Q_{xq}	The cost of the best-so-far path for q at node x
p_{xq}	The path exploration probability for q
$cntU_q$	The consecutive number of minor updates to Q_{xq}

TABLE I

THE QUERY-PATH ENTRY FOR QUERY VECTOR q AT NODE x IN THE BASIC ISRL.

C. 1-thread semantic search

All nodes handle queries similarly. When node x receives from node y a Query message for query vector q initiated at node s, it takes the following actions.

1) If a local file f is semantically similar to q, x replies to y a QueryResponse message that includes the query, the detailed description about f, and the cost $Q_{xq} = 0$. The query will not be forwarded further. If no desired file exists on x, go to step 2.
2) If x has a path entry that contains vector q, then with probability p_q, the query is forwarded to a randomly chosen neighbor other than z (exploration). With probability $(1 - p_q)$, the query is sent to the best neighbor z discovered so far (exploitation).
3) If x does not have an entry for q, it then directs the query to a neighbor chosen randomly. The query forwarding stops when TTL expires. In the dynamic overlay scenario, x also generates a query response message at TTL expiration (query failure).

A query response message is sent along the reverse query path and terminates at the querying source. p_q is an important design parameter. It will be discussed in detail in the next subsection. To avoid searching loops (duplicate queries), each Query message carries all node IDs on the query path so far.

The cosine similarity model is selected for evaluating the similarity between a query vector and a document vector or between two query vectors. This model is widely used in the IR community. Given two m-dimensional semantic vectors, $a = (a_1, a_2, ..., a_m)^T$, and $b = (b_1, b_2, ..., b_m)^T$, their semantic similarity, $Sim(a, b)$, is the cosine of the angle between them

$$Sim(a, b) = \frac{\sum_{i=1}^{m} a_i b_i}{\sqrt{\sum_{i=1}^{m} a_i^2} \sqrt{\sum_{i=1}^{m} b_i^2}}.$$

```
Path update at node x when getting qr_qs from node y:
    // Basic ISRL;
    // qr_qs: QueryResponse message for Query qs.
    // qs: Query message for vector q and source node s.
    // Q_xq/Q_yq: the cost of the best path for q currently known to x/y.
 1. if (the first path for q)
 2.       Q_xq = Q_yq + 1;
 3.       z = y;
 4. else if (Q_yq + 1 < Q_xq)
 5.       ΔQ_xq = Q_xq − (Q_yq + 1);
 6.       Q_xq = Q_yq + 1;
 7.       z = y;
 8. else
 9.       ΔQ_xq = 0.
10. if (ΔQ_xq < ε_1)
11.       ++ cntU_q;
12. else
13.       cntU_q = 0;
14. Adjust p_q based on cntU_q;
15. Include (q, Q_xq) in the new QueryResponse message;
```

Fig. 1. Path update algorithm at node x when receiving QueryResponse message qr_{qs} from node y in basic ISRL.

Because we normalize a query vector and a document vector before the similarity computation, the denominator in the formula is 1. The larger $Sim(a, b)$, the semantically closer the two vectors a and b. We use a threshold to determine whether two semantic vectors are similar.

D. Path update

In the static overlay scenario, path entries are updated only when queried files are found. The new path information is carried in the QueryResponse message and transferred along the reverse query path. All nodes on the reverse query path update their related entries accordingly. Figure 1 shows how node x updates its path entries when receiving from y a message qr_{qs} in response to query qs. If the discovered path via y is the first path for q, x adds this new path. If the cost of the new path via y, denoted by $Q_{yq} + 1$, is lower than that via the currently known best neighbor z, denoted by Q_{xq}, then replace z by y. Otherwise, keep z and reset the update to Q_{xq}, denoted by ΔQ_{xq}, to zero. If the update to Q_{xq} is trivial, then increase the counter $cntU_q$. Otherwise, reset $cntU_q$. x always propagates the updated Q_{xq} using a new QueryResponse message.

In the dynamic overlay scenario, discovered paths may be invalidated due to node leaves or failures. We remove these false paths in two ways. The first method is to set a maximum life time for a discovered path. Any existing path is purged automatically when it reaches its maximum life time. The second method is to delete invalid paths on the reverse query path at query failure. When a query q fails at TTL expiration, the last node on the query path generates a query

response message to be delivered back to the query source. If a node has a cache entry for q via its upstream neighbor on the reverse query path, the entry is removed immediately.

In the dynamic overlay scenario, path update also occurs on query success. But the path replacement policy considers both path quality and path freshness, which can be estimated using the following formula respectively:

$$P_{quality} = 1 - \frac{Cost}{MaxTTL + 1}.$$

$$P_{freshness} = 1 - \frac{Age}{MaxAge}.$$

$Cost$ refers to the Q value of a path. We use $MaxTTL + 1$ instead of $MaxTTL$ to avoid 0 path quality. A path with the maximum length is not useless. Age refers to the time period that a path entry has stayed in the cache without being refreshed or replaced. When an existing path is refreshed, its age is reset to 0. $MaxAge$ denotes the maximum time period that an existing path can be cached before it is considered expired and is purged. The total goodness score of a discovered path is

$$P_{goodness} = \begin{cases} P_{quality} + P_{freshness} & \text{if } Age < MaxAge \\ 0 & \text{otherwise}, \end{cases}$$

A new path for q replaces an existing one if the new one has a higher goodness score. This path replacement policy is designed to implement the following rules. A newly discovered path for q replaces the existing best-so-far path for q if one of these conditions is satisfied.

- The existing best-so-far path expires ($Age = MaxAge$).
- The new path has a smaller cost than the existing one.
- The existing path is very old (but not expired yet) and its path cost is not significantly smaller than the new one.

E. Path exploration

Path exploration is controlled by the system parameter p_q, which determines which neighbor to send query q to. The neighbors attempted by a node x serve as the training samples for x

to learn the best path for q. x faces a tradeoff between exploration and exploitation. Large p_q favors gathering new information by exploring unknown paths. Small p_q prefers utilizing already discovered good paths so as to reduce the total path cost. Typically, exploration is favored initially and exploitation is preferred later. In this paper, we consider two design options for setting p_q.

p_q **design 1: coarse adaptation.** In this option, we use two constants: one large value α_{high} for the initial progressive exploration; one small value α_{low} ($< \alpha_{high}$) for later lazy exploration. Each node, say x, initializes p_q to α_{high} to gather new information aggressively. x changes p_q to α_{low} when the best path discovered so far is close to the actual best path. This can be estimated by a large ($\geq$ a threshold w_1) number of consecutive minor updates to path cost Q_{xq}. $cntU_q$ is used to count such updates. The detailed function is shown below.

$$p_q = \begin{cases} \alpha_{high} & \text{if } cntU_q < w_1 \\ \alpha_{low} & \text{otherwise.} \end{cases}$$

p_q **design 2: fine tuning.** Adjust p gradually. Each node, say x, initializes p_q to a large value. Each time a certain number of consecutive minor updates is observed at x, reduce p_q by a constant amount μ. When forwarding a query for q next time, x explores less. p_q decreases as the best-so-far path approaches the actual best path.

$$p_q = p_q - \mu, \; if \; cntU_q > w_2.$$

If a cached path for q is purged due to topology change, the exploration probability for q is reset according to coarse adaptation or fine tuning.

F. Illustration

Figure 2 shows an example of the basic ISRL search. The query source is A, denoted by an empty square. The desired file is only hosted in node E, represented by a solid circle. The arrows indicate query forwarding directions. The number next to each arrowed line refers to the experiment sequence. A - F are node IDs. The number within curly brackets next to each node ID is the Q value for this query. The first trial via B is successful, which causes nodes D, C, B, and A to add new entries for this query with Q values as shown in the figure. The second and third trials fail. No update to Q values occurs on the reverse query path. The fourth experiment succeeds and causes node F to add a new entry with cost 1 for the query. A replaces the entry

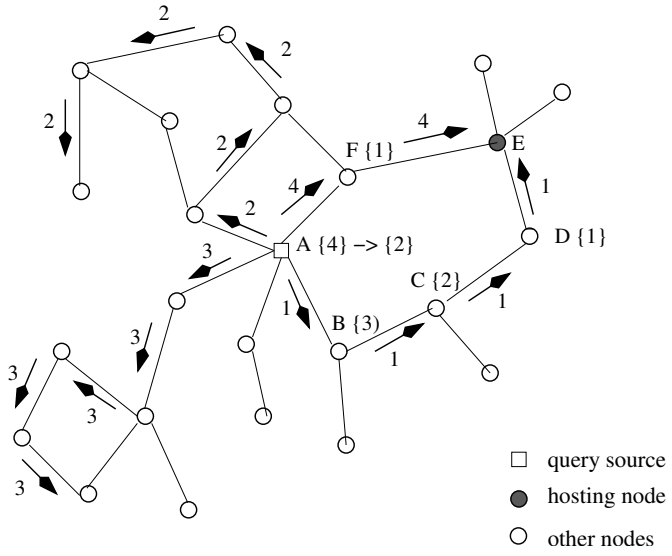


Fig. 2. An example of the basic ISRL search.

to B by the new entry to F with lower cost 2. Therefore A learns the best path to hosting node E through trials.

The basic ISRL corresponds to Q-learning with the following settings: $r = 1$, $\gamma = 1$, and $\eta = 1$. Thus we have

$$Q(s, a) = 1 + min_b Q(s^{'}, b)).$$

Because $Q(s, a)$ represents the path cost, it is to be minimized. r is set to 1 considering that the path cost is computed in terms of overlay hops. γ is set to 1 because $Q(s, a)$ represents path costs and discounts are unnecessary. η is set to 1 because we can adjust learning through the exploration probability.

IV. MP-ISRL (MULTI-PATH ISRL)

The basic ISRL keeps only one path for each successful query vector. This may not suffice when we try to find more than one desired file. In this extended version, MP-ISRL, each node keeps multiple (k) paths for each query vector. Both exploration and exploitation are implemented as k-thread forwarding. An exploring message for vector q is always sent to k neighbors chosen randomly. An exploiting message always utilizes k discovered paths for q. If a node does not have k known paths, it replaces unknown paths by random forwarding. The exploration probability for q is similar to basic ISRL. A newly discovered path is always added before k paths have been found, and replaces the worst path thereafter if it is better. Path evaluation can be based on cost

Notation	Meaning
q	The query vector
z_j	The neighbor on the j^{th} path for q, $j \in [1, k]$
Q_{xqj}	The evaluation score of the j^{th} path for q, $j \in [1, k]$
p_q	The exploration probability for q
$cntU_q$	The consecutive number of minor updates to any Q_{xqj}, $j \in [1, k]$

TABLE II

THE PATH ENTRY FOR VECTOR q AT NODE x IN MP-ISRL.

alone or discounted reward considering both cost and similarity scores of desired files. In this section, we consider P2P networks where all peers have the same service capacities. Section VI will discuss how to handle peers with heterogenous capacities.

A. Path entries

Each node keeps the top k best-so-far paths for each successful query vector. Table II lists items in the path entry for query vector q kept at node x. (z_j, Q_{xqj}) represents the j^{th} path. z_j is the next-hop neighbor on this path and Q_{xqj} is the path evaluation score. p_q has the same meaning as that in basic ISRL. To determine that the top k best paths have been found, $cntU_q$ counts the consecutive number of minor updates to any path for q.

B. k-thread semantic search

When a desired file is found, a node x in MP-ISRL acts similarly to nodes in basic ISRL. MP-ISRL differs from basic ISRL in query forwarding. When node x receives a semantic Query message from node y that contains the query vector q initiated at query source s, x acts as follows:

1) If x has one or more paths for q, then with probability p_q, q is forwarded to k randomly chosen neighbors other than those in the known paths (exploration). With probability $(1-p_q)$, x dispatches the query along k already discovered paths for q.
2) If no existing path for q is kept at x, then send the query to k neighbors chosen uniformly at random.

C. Path exploration and update

Like basic ISRL, MP-ISRL sets the exploration probability p_q for query vector q using a similar formula according to the quality of already discovered paths. Unlike basic ISRL, MP-ISRL gathers

```
Path update at node x when getting qr_qs from node y:
    // MP-ISRL; path cost as evaluation score.
    // qr_qs: QueryResponse message for Query qs
    // qs: Query message for vector q and source node s
    // k_c: the number of currently known paths for q
    // j: the index of the last known path for q
 1. if (y == z_i in the i^th path for q) //existing next-hop
 2.     if (Q_yq + 1 < Q_xqi) // a better one
 3.         Q_xqi = Q_yq + 1;
 4.         ΔQ_xq = Q_xqi − (Q_yq + 1);
 5.     else
 6.         ΔQ_xq = 0;
 7. else if (k_c < k) //a new next-hop; not found k paths yet
 8.     Q_xqj = Q_yq + 1;
 9.     ΔQ_xq = Q_xqj;
10.     z_j = y; j = j + 1; k_c = k_c + 1;
11. else //found k paths already
12.     Find the worst existing path, path w: (z_w, Q_xqw);
13.     if (Q_yq + 1 < Q_xqw)
14.         ΔQ_xq = Q_xqw − (Q_yq + 1);
15.         Q_xqw = Q_yq + 1;
16.         z_w = y;
17.     else
18.         ΔQ_xq = 0.
19. if (ΔQ_xq < ε_2)
20.     ++ cntU_q;
21. else
22.     cntU_q = 0;
23. Adjust p_q based on cntU_q;
```

Fig. 3. Path update at node x when receiving from node y QueryResponse message qr_{qs} in MP-ISRL; path cost as evaluation score.

new information more aggressively by using k-thread explorations because MP-ISRL aims to locate the top k best paths. When node x decides to explore new paths for any query vector q, it dispatches q to k randomly chosen neighbors that do not appear in known paths.

As for path updates in MP-ISRL, like the basic version, path replacements occur only when a query result is found. New path information is propagated along the reverse query path. We evaluate paths in two ways. The first one is by path cost like the basic version. The second one is a new model, called discounted reward, which is designed for the scenario when we care about the similarity scores of desired files with respect to the same query.

Path cost as evaluation score. Figure 3 shows the path update in this scenario. When a node x receives a QueryResponse message qr_{qs} for query vector q and source s from node y, if y is the next-hop neighbor on the existing i^{th} path for q, x records the new cost Q_{xqi} if y brings a smaller cost. If y is a new next-hop neighbor and x hasn't discovered all k paths yet, add y as a new path for q. If there are k existing paths for q, replace the worst existing path if the new path via y is better. If the update to any existing path of q is trivial ($< \epsilon_2$), the counter $cntU_q$ is

increased. Otherwise, $cntU_q$ is reset.

Discounted reward as evaluation score. When evaluating paths using the path cost alone, we do not know how similar the desired files found through different paths are to the current query. If we want to get as similar files as possible, we can use both the path length and the similarity score between a desired file and the current query to evaluate discovered paths. The reward of a path for query vector q is the similarity score between q and the most similar desired file on the hosting node. We still prefer desired files that are close to the query source. Therefore, the reward is discounted exponentially with increasing distance from the neighbor of the hosting node on the reverse query path.

Path update in this scenario is different from using path cost alone mainly in computing and comparing Q values for paths. With Q values being discounted reward, paths with higher Q values are better. Q values are to be maximized. When a desired file with vector f for query vector q is found on a target node v, the similarity score, $Sim(q, f)$ is returned to the upstream neighbor y on the query path. y does not discount the reward and sets the Q value for this path, Q_{yqv}, to be $Sim(q, f)$. y then informs its upstream neighbor x of Q_{yqv}. x assigns the Q value for this path according to the following formula.

$$Q_{xqy} = \gamma * Q_{yqv},$$

where γ is the system parameter to control the amount of discount. It is in the range $[0, 1]$. The larger the γ, the less the discount, and the less the impact of path cost. The rest of the nodes on this new query path calculate the Q value for the path similarly to x.

In the dynamic network scenario, path removal is handled similarly to the basic ISRL. Path replacement considers both path quality and path freshness. If path cost is used in path evaluation, the path quality is computed in the same way as in the basic ISRL. If discounted reward is used, the path quality is the same as Q value.

In summary, both basic ISRL and MP-ISRL apply Q-learning to P2P searching. Each node learns the best next-hop neighbor to forward a given query in order to follow the best path with the lowest cumulative cost or the highest cumulative reward. The learning process iterates through continuous exploration and exploitation. The learning rate is adjusted via the exploration probability. Basic ISRL keeps the best path while MP-ISRL maintains the top k best paths.

V. C-ISRL (CLUSTERED ISRL)

In this section, we describe another extended version, C-ISRL, which tries to reduce the storage overhead in the basic ISRL through clustering. We will first introduce the basic idea of C-ISRL, then discuss how to process queries and how to control path exploration and path update. The discussion in this section assumes that the service capacities of all peers are the same. The next section will describe how to handle peers with various capacities.

A. Protocol overview

In the basic ISRL, each node keeps one path for each distinct query vector. When the query set contains a large number of different query vectors, each node has to devote a significant amount of memory for path maintenance. C-ISRL aims to reduce the storage overhead by keeping one path for each cluster of similar query vectors.

Given a query qs for query vector q issued by query source s, with probability $1 - p_q$, node x exploits the existing path for the most similar query vector or probabilistically chooses one among the paths for all similar query vectors. With probability p_q, node x gathers new information by sending q to a random neighbor other than that in the paths for similar query vectors.

The goal of path update in C-ISRL is to seek the best path for each distinct query cluster, each of which is represented by a successful query vector. The representative query vectors at any node are kept dissimilar to each other so as to maximize distances between clusters. When a query vector q is resolved, the new path information is spread in reverse. If q is similar to a set of existing query vectors SV, a system policy determines either to add q and remove all vectors in SV or to discard q and keep SV. The policy will be discussed later in this section. If no existing query vector is similar to q, then add q and its path.

In C-ISRL, we make the following assumptions. Documents in a P2P network are localized. If a file f is found on a node x, files semantically similar to f are most likely to exist in x's neighborhood. If the initial document distribution does not satisfy this requirement, we can use the method in [28] to change the document distribution and then run C-ISRL. In addition, because different users may have similar interests, queries are assumed to be clustered. Many similar queries that are targeted for semantically similar files may be issued from different nodes.

Notation	Meaning
q_i	Representative query vector of cluster i
z	Neighbor on the best-so-far path for i
Q_{xi}	Evaluation of the best-so-far path for i
p_i	Exploration probability for cluster i
$cntU_i$	Number of minor updates to Q_{xi}

TABLE III
THE PATH ENTRY FOR CLUSTER i AT NODE x: C-ISRL.

B. Path entries

To ease searching, each node keeps track of the best-so-far paths for a certain number of query vectors, each of which serves as the representative vector of a distinct group of similar queries. Table III shows the structure of a path entry for cluster i kept at node x. Past queries in i were forwarded by x to some neighbor z. q_i refers to the representative query vector for cluster i. It is one of the past query vectors in the cluster. z is the next-hop neighbor on the best-so-far path for forwarding q_i and similar queries. Q_{xi} refers to the evaluation score of the best-so-far path for queries in cluster i. The evaluation score can be the path cost or discounted reward. p_i is the probability of gathering new information for queries in cluster i. $cntU_i$ denotes the consecutive number of minor updates to the path cost for cluster i, Q_{xi}.

C. Query processing

C-ISRL is different from the basic ISRL in query forwarding. When node x receives from node y a semantic Query message qs that contains the query vector q initiated at querying source s, x directs qs as follows.

1) If x has one or more path entries that contain query vectors similar to q, then with probability p_q, q is forwarded to a randomly chosen neighbor other than that in the associated path entries (exploration). With probability $(1 - p_q)$, x dispatches q along one of the already discovered paths for similar queries.
2) If no path for past similar queries is kept at x, then it sends qs to a neighbor selected uniformly at random.

Path selection policies during path exploitation can be deterministic or probabilistic. We can always choose the path for the most similar query vector or probabilistically select one according

```
Path update at node x when getting qr_qs from node y:
    // C-ISRL: path cost as evaluation score.
    // qr_qs: QueryResponse message for qs.
    // qs: Query message for vector q and querying source s.
    // Q_yq: the cost of the best path known to y for q and similar vectors.
    // p_l, p_j, p_m: the exploration probabilities for clusters l, j, m.
1.  if (q is not similar to any existing query vector)
     //Assume that q represents cluster l.
2.       Add new entry (q, y, Q_yq + 1, p_l, 0); Return;
3.  if (q is similar to only one existing vector representing cluster j)
4.       if (Q_yq + 1 < Q_xj) // a better path
      //q becomes the new representative for cluster j
5.            q_j = q; z = y;
6.            ΔQ_xj = Q_xj − (Q_yq + 1);
7.            Q_xj = Q_yq + 1;
8.       else
9.            ΔQ_xj = 0;
10.      if (ΔQ_xj < ε_2)
11.           ++ cntU_j;
12.      else
13.           cntU_j = 0;
14.      Adjust p_j based on cntU_j; Return;
     //multiple similar query vectors
15. Find all existing similar query vectors V_s and their entries E_s;
16. if (replace V_s by q)
     //Assume that q represents cluster m.
17.      p_m = weighted average_i{p_i in E_s};
18.      ΔQ_xq = Min_i{Q_xi in E_s} - (Q_yq + 1);
19.      if (ΔQ_xm < ε_2)
20.           ++ cntU_m;
21.      else
22.           cntU_m = 0;
23.      Adjust p_m based on cntU_m;
24.      Remove all entries in E_s;
25.      Add new entry (q, y, Q_yq + 1, p_m, cntU_m); Return;
26. else
27.      Reset each cntU_i in E_s;
28.      Adjust p_i based on cntU_i for each entry i in E_s; Return;
```

Fig. 4. Path update algorithm at node x when receiving QueryResponse message qr_{qs} from node y for semantic query qs: C-ISRL; path cost as evaluation score.

to query similarity scores. Let q_1, q_2, ..., q_n be the n query vectors that are similar to current query q. The probability of choosing the path for vector q_i is computed as below.

$$P(q, path(q_i)) = \frac{Sim(q, q_i)}{\sum_{j=1}^{n} Sim(q, q_j)}.$$

The deterministic policy offers better path quality. The probabilistic policy is more robust and can balance the network load.

D. Path exploration and update

Path exploration in C-ISRL is similar to the basic ISRL except that the exploration probabilities are updated somewhat differently. The differences are included in the discussion of path update.

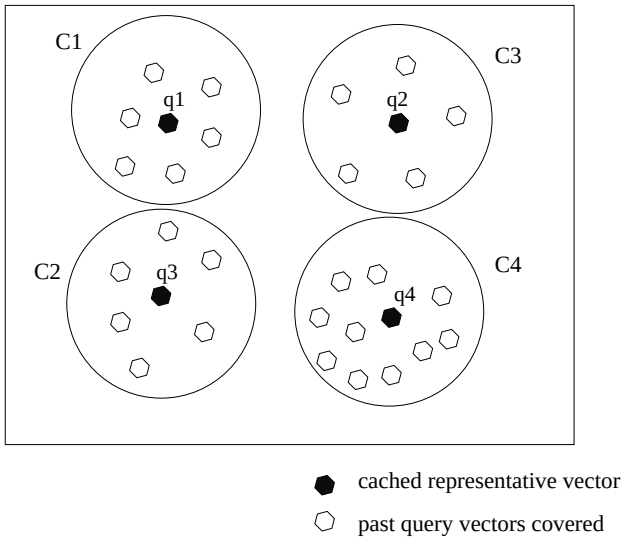


Fig. 5. An example of 4 query vector clusters at node x in C-ISRL.

Path update policy in C-ISRL is inspired by the maximum independent set (MIS) concept. A MIS of a network is a subset of nodes where each node in the network is either in the MIS or a neighbor of a node in the MIS. No nodes in the MIS are neighbors of each other. We can map a set of query vectors to a network. A distinct query vector becomes a node. If two vectors are similar to each other, then create a link between them. We want to ensure that no two representative query vectors are neighbors and every non-representative query vector is a neighbor of some representative query vector.

Following this inspiration, the goal of path update is that each node aim to keep one path for a number of query vectors. Each query vector acts as the representative for a distinct cluster of similar query vectors. Its path is the best one among all paths for this cluster. All query vectors kept in the path cache together cover the entire query set. A query vector covers another one if both are similar to each other. Because each query vector in the path cache is designed to cover a distinct set of query vectors, any two vectors in the path cache are not similar to each other. Figure 5 illustrates this idea. A solid pentagon denotes the existing representative query vector for a cluster. An empty pentagon refers to a past query vector similar to the representative vector. The circle denotes the coverage of a representative query vector that depends on the similarity threshold. The figure shows four clusters C_1 - C_4 kept at node x. The details of path update are described as follows.

Path updates occur while query response messages are transferred along the reverse query path.

The algorithm is shown in Figure 4. We use path cost to represent the path quality. When node x receives from node y QueryResponse message qr_{qs} for query vector q and source s, if q is not similar to any existing query vector, x initializes the exploration probability for cluster l that q represents, denoted by p_l, and adds q. If q is similar to only one existing query vector that represents cluster j, q becomes the new representative vector for cluster j if the path via y is shorter than the current best path for cluster j (the one via neighbor z). The amount of change to path cost is the difference between the costs of these two paths. The exploration probability for q and similar vectors in cluster j, denoted by p_j, is adjusted based on $cntU_j$ similarly to the basic ISRL.

q may have a set of (more than 1) similar query vectors in the path cache, denoted by V_s. For example, in Figure 6, q is similar to two existing query vectors q_1 and q_2. It means that the cluster represented by q overlaps with the clusters represented by q_1 and q_2. In this scenario, the multi-vector replacement policy probabilistically determines whether to replace V_s by q or to discard q. The replacement probability, $PR(q, V_s)$ is computed according to the following formula. In the formula, $|V_s|$ denotes the size of V_s. Q_{yq} denotes the cost of the best path for q and its similar vectors that is currently known to y. $Min_i\{Q_{xi}\}$ and $Max_i\{Q_{xi}\}$ refer to the costs of the best and worst paths among all vectors in V_s. c_s is a system parameter.

$$PR(q, V_s) = f(Q_{yq}, Min_i\{Q_{xi}\}) \cdot g(Q_{yq}, Max_i\{Q_{xi}\}) \cdot h(|V_s|),$$

$$f(Q_{yq}) = \begin{cases} 1 & \text{if } Q_{yq} < Min_i\{Q_{xi}\} \\ 0 & \text{otherwise}, \end{cases}$$

$$h(|V_s|) = \begin{cases} 1 & \text{if } |V_s| < c_s \\ 0 & \text{otherwise}, \end{cases}$$

$$g(Q_{yq}, Max_i\{Q_{xi}\}) = \frac{1/Q_{yq}}{1/Q_{yq} + 1/Max_i\{Q_{xi}\}}.$$

The function $f(Q_{yq})$ specifies that a qualified new query vector must have a path better (i.e. with a lower cost) than the best cached path among all similar query vectors. $h(|V_s|)$ stipulates

that it is not desirable to replace many cached similar query vectors by a single vector with a better path. [1] The function $g(Q_{yq}, Max_i\{Q_{xi}\})$ is used to weigh the quality of the new path for q and the worst existing path for similar vectors. If the worst existing path is significantly worse, the replacement probability is higher.

Assume that q represents cluster m, which also contains all vectors in V_s. When q replaces V_s, the amount of update to the cost of the best-so-far path for cluster m is the difference between the cost for the new path via y and the best existing path (with the lowest cost) among all vectors in V_s. The exploration probability for cluster m, p_m, is initialized to be the average of exploration probabilities of similar vectors in V_s weighted by the similarity scores. The detailed formula is:

$$p_m = \frac{1}{|V_s|} \sum_{i=1}^{|V_s|} (Sim(q, q_i) \cdot p_i).$$

In the formula, $Sim(q, q_i)$ denotes the similarity score between q and the $i-th$ vector in V_s, q_i. p_i refers to the exploration probability for cluster i represented by q_i. p_m is then adjusted similarly to the basic ISRL according to the chosen path exploration policy.

If we use discounted reward to evaluate paths, the path exploration and update is similar except that the Q values of paths need to be computed and compared differently.

One thing worth to mention is that C-ISRL does not increase searching overhead because of using the representative vector for a distinct group of similar queries. If there is no hint for queries similar to the current query vector, both C-ISRL and basic-ISRL will choose to explore a potential path. If such a hint exists, the hint will cause a very low false positive rate. It means that a search by following the hint for a similar query vector will be successful in most cases. In C-ISRL, documents in a P2P network are assumed to be localized. If we can find a file on a node, it is very likely to find similar files nearby. If the initial document distribution does not meet this assumption, the algorithm in [28] can be used to modify the distribution prior to executing C-ISRL.

[1] Trade-off: when $|V_s|$ is large, q can cover many vectors. However, each node needs to re-learn the paths for query vectors similar to those in $|V_s|$, but not similar to q.

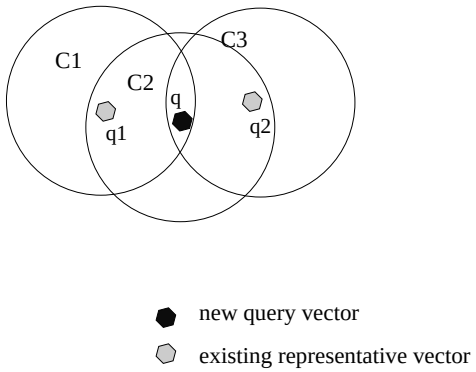


Fig. 6. Query cluster overlapping in CISRL.

VI. Discussion

In this section we discuss some more scenarios where MP-ISRL may be applied and other extensions of ISRL search.

A. Other application scenarios of MP-ISRL

The multi-path ISRL was presented earlier for k-thread semantic search. It can also be utilized in 1-thread semantic search for balancing loads on different paths. During each search, one out of k existing paths is chosen uniformly at random or probabilistically based on their Q values. In addition, the k paths serve as backup paths for each other and therefore increase search robustness. It is possible to use the discounted reward to evaluate paths in the basic ISRL if more than one desired file is located at different nodes and the semantic search is targeted at one similar file nearby.

The MP-ISRL can also be employed in non-semantic queries: looking up multiple replicas of a single file by ID. We can keep multiple paths to each queried file. But the discounted reward will not be applicable.

B. Other extensions of ISRL

The ISRL described previously can be extended in many ways. We can combine MP-ISRL and C-ISRL and have each node maintain multiple paths for each query cluster. Multiple existing paths for similar query clusters can be probabilistically selected for path exploitation as follows. Assume that $k < N$ paths need to be selected among N candidate paths. The probability of

choosing the i^{th} path when forwarding query vector q, $P(q, path_i)$, is computed according to the following formula:

$$P(q, path_i) = \frac{k + h_i^{\beta}}{\sum_{j=1}^{N}(k + h_j^{\beta})}.$$

, where h_i refers to the similarity score between the vector q and the vector associated with the i^{th} path. k and β are tunable system parameters.

Current ISRL models only utilize positive reinforcement. If a learner has a bad experience it does not receive punishment. We can extend the reward to be negative. One example is to define negative reward in terms of pheromone evaporation [29] (TTL associated with path information). We can also apply other reinforcement learning models, such as prioritized sweeping [30], to learn faster with less training data.

Proactive hints can be integrated into current ISRL models. For example, while rewards are being collected on the reverse query path, each learner on the path can proactively spread these rewards to its neighbors. Another scheme is to proactively deploy random agents. The deploying node can pick a representative query vector with a known good path or a document vector in its local data store and have the agent spread the known query path or the path to the document vector.

A higher exploration probability may incur higher load on peers and therefore cause higher demand in peers' computing resources, particularly CPU power. When peers have heterogenous service capacities, we can include capacity as a factor in determining the exploration probability of a peer. One simple approach is to classify the service capacities of all peers into different levels, say totally L_c levels, with 1 being the level of the strongest service ability, L_c being the weakest. Then given a peer with a capacity at level l_i, the peer's capacity-aware exploration probability p_{qc} is computed according to the following formula,

$$p_{qc} = \frac{l_i}{L_c} p_q,$$

where p_q represents the exploration probability that is computed without taking into account of peer capacities.

Parameters	Value
Number of nodes	2000 or 5000
Graph model	Random graph
Number of documents	6000
Number of query sources	100
Number of unique query vectors	100
Query distribution	Uniform
Document distribution	Clustered based on similarity
Similarity threshold	0.43

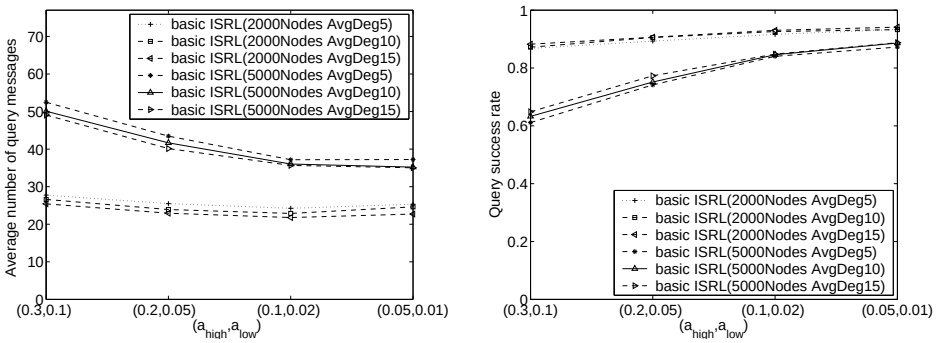
TABLE IV
MAJOR SYSTEM SETUP.

VII. EVALUATION

In this section, we present the experimental setup, evaluate the tradeoffs in ISRL design, and compare ISRL to existing searching schemes in unstructured P2Ps. The simulation setting and results in static network scenarios are described in Section A to D. The simulation in dynamic network scenarios is presented in Section E.

A. Simulation setup

We simulated the algorithms using random graphs that have 2000 and 5000 nodes and average degrees of 5, 10 and 15. The document collection in the P2P network was chosen from the well-known TREC data set [31]. All documents that are semantically similar are distributed to randomly selected neighborhoods in the P2P overlay. A number of document vectors are selected as semantic query vectors such that some query vectors are similar to each other. Queries arrive sequentially. Each query consists of a query source node and a query vector. 100 nodes are randomly chosen as query sources. The query distribution is uniform. Each unique query vector is requested for a total of 500 times from different query sources. The threshold that determines the semantic similarity between two documents is 0.43. Table IV shows some system parameters and their values. The performance metrics are the average number of messages per query, the average query success rate, and the average number of desired documents found. A query is considered successful if at least one desired document is found.



(a) Average number of query messages. (b) Average query success rate.

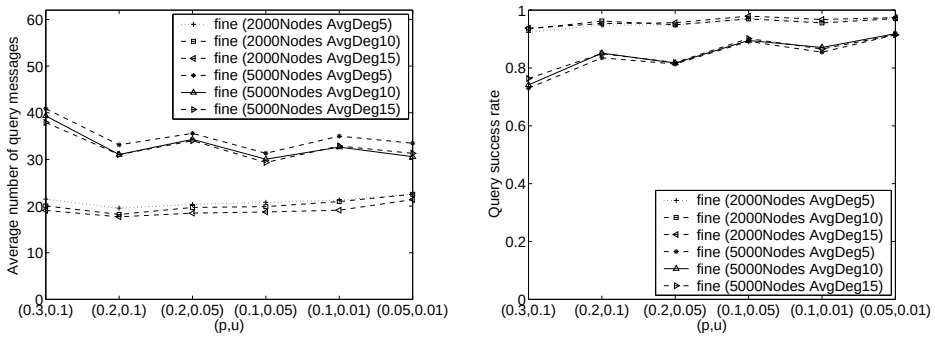
Fig. 7. Basic ISRL: coarse path exploration with different parameters. a_{high}, a_{low} correspond to the parameters α_{high} and α_{low}.

B. Evaluation of the basic ISRL

In this subsection, we first investigate the impact of different path exploration policies, then compare the basic ISRL approach to random walks.

1) Path exploration: coarse vs fine: Figure 7 shows the message loads and query success rates of coarse path exploration, as described in Section III.E, with different parameters in different networks. a_{high} and a_{low} in the x-axis refer to the aggressive exploration probability α_{high} and lazy exploration probability α_{low} respectively. The specific parameter values are chosen to reflect high, medium, and low exploration scenarios. Clearly, different (α_{high},α_{low}) values affect the performance dramatically in 5000-node networks, but not much in 2000-node networks. Smaller exploration probability values are better than larger ones. The best performance is achieved with α_{high} being 0.05 and α_{low} being 0.01 in all networks. We can also observe that the network degree does not affect the performance significantly, but the network size does.

The performance of fine path exploration, as described in Section III.E, with different parameters is illustrated in Figure 8. p and u correspond to the initial probability p_q and the step decrease value μ. It is observed that different values of (p_q, μ) have a major influence on the performance in 5000-node networks, and a minor influence in 2000-node networks. Similar to coarse exploration, the network size, not the network degree, is a performance factor. The best parameter values are p_q being 0.05 and μ being 0.01. Compared to coarse exploration, the average number of query messages generated by fine exploration is reduced approximately 14% to 25%. The query success rate achieved by fine exploration is increased around 7% to 19%. Fine exploration is superior to coarse exploration.

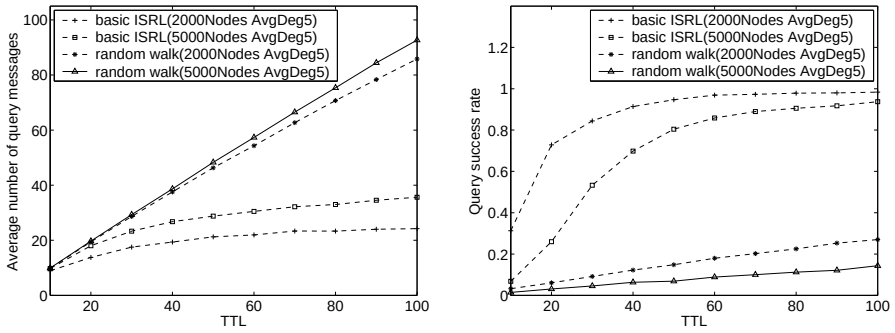


(a) Average number of query messages. (b) Average query success rate.

Fig. 8. Basic ISRL: fine path exploration with different parameters. p, u correspond to the parameters p_q and μ.

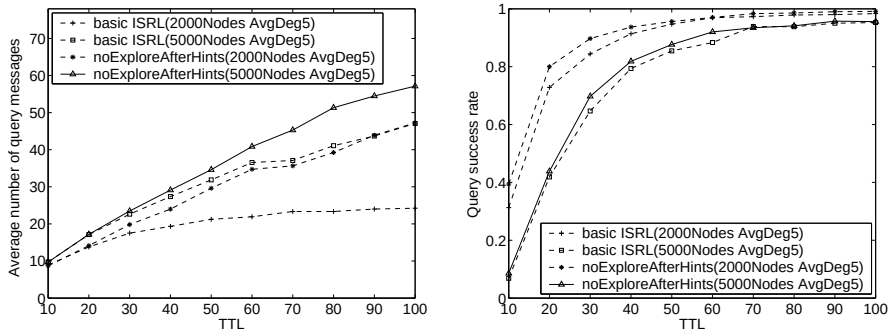
2) Comparison to 1-walker random walks: The basic ISRL is evaluated against 1-walker random walks where each node randomly forwards a query to one neighbor. Figure 9 illustrates their performance differences with varying TTLs in random networks with 2000/5000 nodes and average degree 5. The basic ISRL uses the path update policy: fine tuning with parameters, p_q being 0.05 and μ being 0.01. The path evaluation is based on the path cost. It is observed from Figure 9 (a) that in a given network, the message consumption and the query success rate in both schemes increase with large TTL values because each query can travel far with larger TTL values and therefore more queries can be successfully resolved. However, random walks incur much more traffic than the basic ISRL at higher TTLs because the random walk does not utilize any hints and many messages are wasted. The basic ISRL reduces traffic by exploiting discovered good paths and exploring better paths. In Figure 9 (b), the success rate for the basic ISRL increases dramatically at the beginning. This corresponds to an aggressive exploration period. When many paths have been discovered for most queries, successive new queries will be delivered along the existing paths in most cases. The increase in query success rate then becomes trivial. Both the basic ISRL and the 1-walker random walk achieve a higher query success rate with fewer number of messages in 2000-node networks than 5000-node networks. The basic ISRL outperforms the 1-walker random walk in both types of networks.

3) Comparison to no-exploration-after-hints: To investigate the benefit of path exploration, we also compare the basic ISRL to the algorithm that does not explore new paths after discovering the first one (referred to as no-exploration-after-hints). Figure 10 illustrates their performances in 2000-node and 5000-node networks with average degree 5. The path exploration in the basic ISRL



(a) Average number of query messages. (b) Average query success rate.

Fig. 9. Basic ISRL vs 1-thread random walks in 2000-node and 5000-node networks with average degree 5.



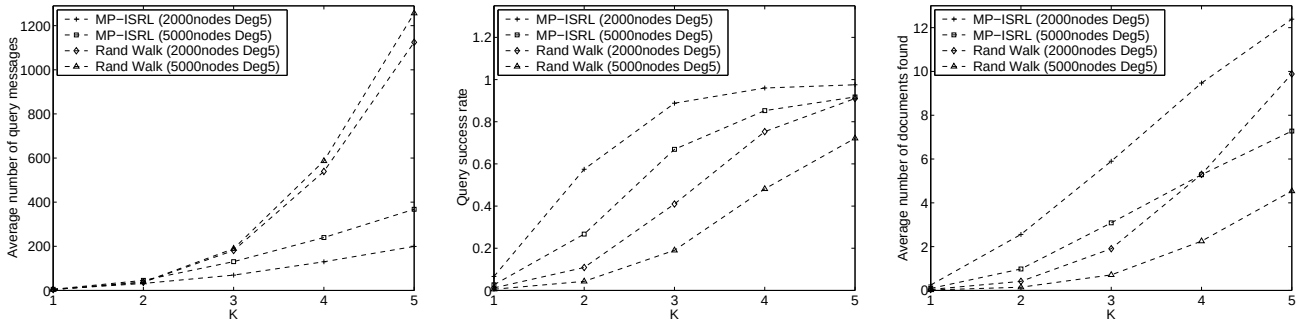
(a) Average number of query messages. (b) Average query success rate.

Fig. 10. Basic ISRL vs no-explore-after-hints in 2000-node and 5000-node networks with average degree 5.

is fine tuning with parameters (p_q, μ) being (0.05,0.01) for 2000-node networks and (0.02,0.01) for 5000-node networks. The path evaluation is based on the path cost. In Figure 10 (a), the algorithm that always exploits the first discovered path consumes more messages than the basic ISRL. This is because the first discovered path may not be the best path and the basic ISRL intends to find the best path to a desired file. Figure 10 (b) shows that both the basic ISRL and the no-exploration-after-hints can learn from the past query results. Therefore the query success rate increases as the TTL increases. The no-exploration-after-hints has a higher query success rate because it always utilizes the discovered path. However, it achieves this with a much heavier traffic than the basic ISRL because ISRL can discover better paths while no-exploration-after-hints can not.

C. Evaluation of MP-ISRL

The MP-ISRL is compared to random walks where each node randomly forwards a received query message to k neighbors. It employs fine exploration with parameters, p_q being 0.05 and μ



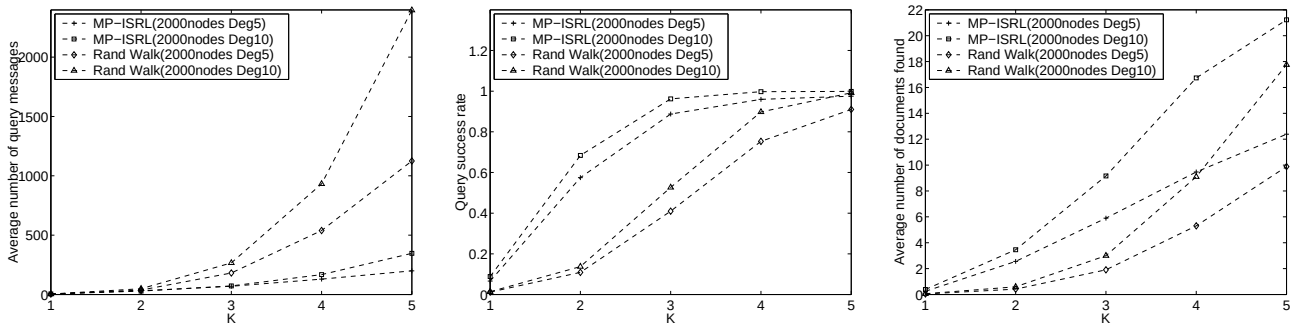
(a) Average number of query messages. (b) Average query success rate. (c) Average number of documents found.

Fig. 11. MP-ISRL vs. Random walk in 2000-node and 5000-node networks with average degree 5.

being 0.01. The path evaluation policy is the path cost. We did experiments in 2000-node and 5000-node random graphs with average degree 5, 10, and 15. The performance in the networks with average degree 15 is close to that in networks with average degree 10 and therefore is not shown in the figures. Figure 11 demonstrates the performance with varying k values in networks with average degree 5 and a variable number of nodes. The TTL value is chosen as 5 because it is large enough to show the performance differences between MP-ISRL and random walks. It is also chosen for faster simulation speed.

MP-ISRL achieves a higher success rate with much fewer messages starting from $k = 3$ than random walks because it can learn and utilize discovered paths. At large k values, the random walk can reach a significant number of nodes. Therefore its success rate is also high. As for the number of discovered documents, MP-ISRL can find more desired documents than random walks at mid-size k values due to its higher success rate. Both MP-ISRL and random walks deliver more queries successfully with less message load and find more desired documents in smaller networks (2000-node) than larger networks (5000-node). MP-ISRL is superior to random walks in both types of networks.

The performances of MP-ISRL and random walks in networks with varying node degrees (5 and 10) and fixed network size (2000 nodes) are illustrated in Figure 12. In denser networks (average degree 10), both MP-ISRL and the random walk resolve more queries successfully, generate more query messages, and locate more desired documents. This is due to the fact that in denser networks each node has more neighbors to forward queries to. MP-ISRL performs better than random walks in both degree scenarios.



(a) Average number of query messages. (b) Average query success rate. (c) Average number of documents found.

Fig. 12. MP-ISRL vs. Random walk in networks with 2000 nodes and varying average degrees (5 and 10).

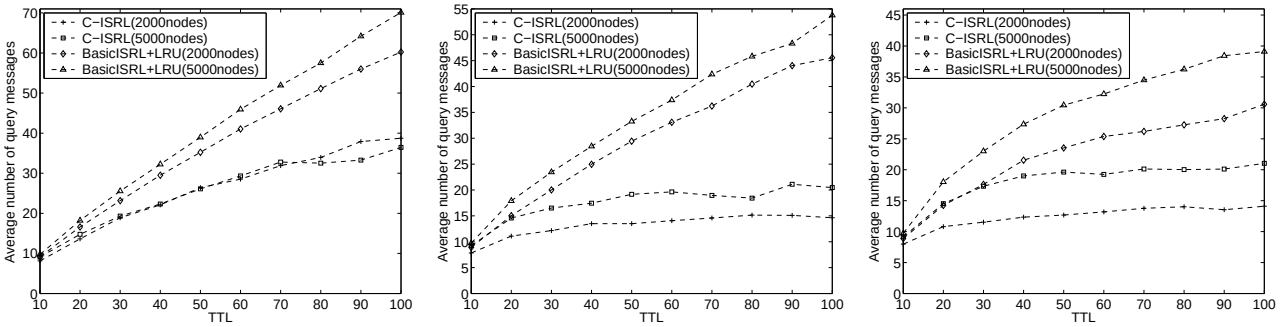
D. Evaluation of C-ISRL

C-ISRL is compared to basic ISRL that uses LRU (least recently used) as path cache replacement policy. We simulated three scenarios with varying cache sizes on random graphs that have 2000/5000 nodes and average degree 5. These scenarios are

1) The cache size is less than the number of query clusters.
2) The cache size is equal to the number of query clusters.
3) The cache size is greater than the number of query clusters.

In the simulation, the path evaluation policy is the path cost. The path exploration is fine tuning. The parameters (p_q,μ) used in C-ISRL are (0.05,0.01) for 2000-node networks with Scenario 1), (0.1, 0.05) for 5000-node networks with Scenario 1), and (0.2, 0.1) for all other cases. The parameters (p_q,μ) used in the basic ISRL with LRU are (0.05, 0.01) for all cases. The query vectors are classified into 10 query clusters. The query forwarding is probabilistic when there are two or more cached similar query vectors. Figure 13 and Figure 14 show the average number of query messages and the average query success rate in these scenarios.

It is observed that C-ISRL surpasses the basic ISRL (+ LRU) in all three scenarios. When the cache size is small (less than the number of query clusters), C-ISRL has a satisfactory performance but the basic ISRL (+ LRU) performs poorly. In both schemes, the average number of query messages increases almost linearly as TTL increases. This is because the small cache size causes many past query vectors to be replaced by newer query vectors. Higher cache misses cause no hints for a large number of queries. Therefore more messages have to be delivered with increasing TTLs. As for the query success rate, both schemes deliver more queries successfully at larger TTLs. However, because of many cache misses, the success rates at most TTLs are low. The



(a) Cache size $<$ number of query clusters. (b) Cache size $=$ number of query clusters. (c) Cache size $>$ number of query clusters.

Fig. 13. C-ISRL vs. basic ISRL and LRU in 2000-node and 5000-node networks with average degree 5: the average number of query messages at varying cache sizes.

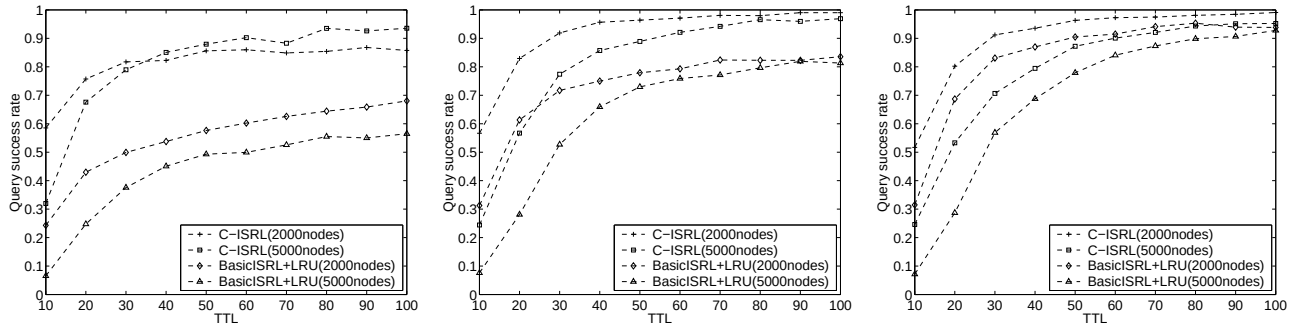
difference in the performance of C-ISRL in both sizes of networks is not significant. The basic ISRL (+ LRU) performs better in smaller networks than in larger ones.

When the cache size is increased to be equal to the number of query clusters, cache misses are reduced. Both schemes perform better and C-ISRL is superior to the basic ISRL (+ LRU). At larger TTL values, the average number of messages consumed by C-ISRL is significantly less and the query success rate of C-ISRL is much higher than the basic ISRL (+ LRU). The improvement in performance is because C-ISRL is able to keep one path for each set of similar query vectors in this scenario. Many queries can be resolved by following the cached paths for similar past queries. In addition, C-ISRL aims to improve the cached paths for similar query vectors. In this scenario, both algorithms perform better in 2000-node networks than in 5000-node ones.

When the cache size is larger than the number of query clusters, the performance of the basic ISRL (+ LRU) improves dramatically. The best success rate is at mid-$90s$ and the average message consumption is reduced further. Due to the increase in the cache size, in the basic ISRL (+ LRU) many queries can utilize cached paths for the same query vectors. This leads to higher success rates and fewer messages. C-ISRL does not benefit much in this scenario because it is intended to keep one path (the best path) for each group of similar query vectors. However, it still outperforms the basic ISRL (+ LRU). In this scenario, both approaches have a higher query success rate and a lighter message load in 2000-node networks than in 5000-node ones.

E. Dynamic network scenarios

To simulate network dynamics, we let $pctDynamics$ number of existing nodes leave and the same number of new nodes join periodically. This way we can maintain a fixed network size.



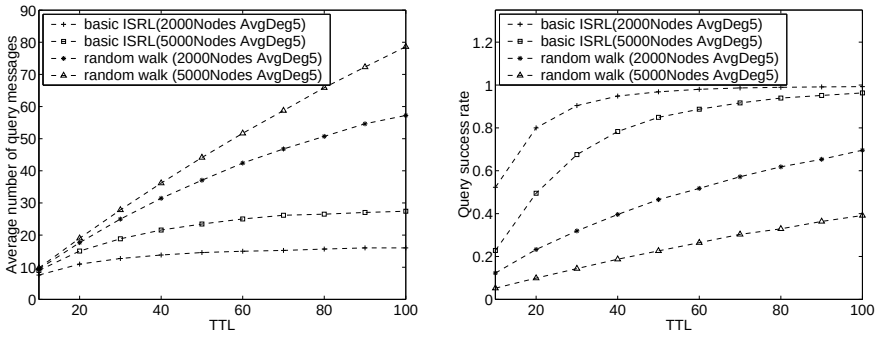
(a) Cache size $<$ number of query clusters. (b) Cache size $=$ number of query clusters. (c) Cache size $>$ number of query clusters.

Fig. 14. C-ISRL vs. basic ISRL and LRU in 2000-node and 5000-node networks with average degree 5: the average query success rate at varying cache sizes.

The random friend seeding policy [32] is used to initialize the neighbor set of a new node. Between each topology change, we randomly select 100 requester nodes to issue queries for 20 query vectors that are also randomly chosen. Each chosen query vector is requested for a total of 600 times from different requesters. In each simulation run, the network changes 15 times. The $MaxAge$ for path expiration is set to be three times of topology change period. The document collection and query vector set are the same as static network scenarios. The document distribution is random. The networks are random graphs with 2000 and 5000 nodes and average degree 5. $pctDynamics$ is 5 percent of the network size.

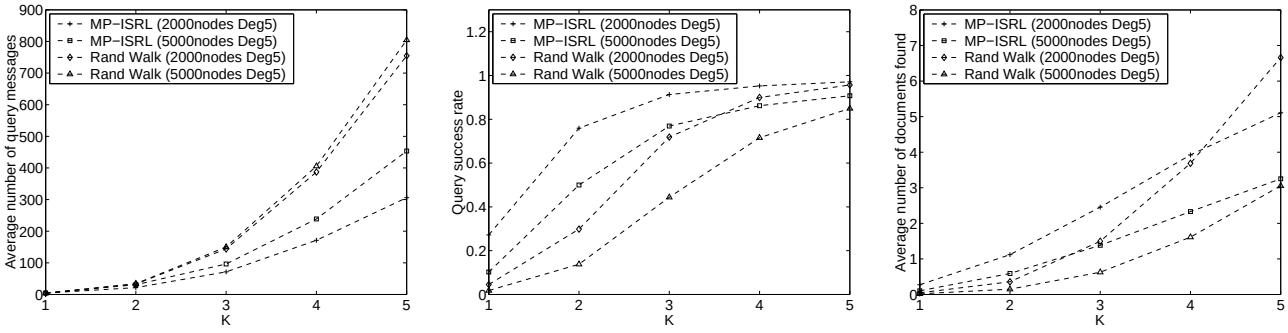
Figure 15 shows the performance of the basic ISRL in contrast with the random walk. The path evaluation is path cost. The path exploration policy is fine tuning with parameters (p_q, μ) being $(0.05, 0.01)$. The trends in the figures are similar to the static network scenario. The basic ISRL outperforms the random walk in both 2000-node and 5000-node networks. This is because the basic ISRL can adapt itself to dynamic networks. It can discover new paths to desired documents due to node joins, re-discover paths due to node leaves/failures, and exploit already discovered paths. The random walk can react to topology changes but can not exploit already discovered paths. The performance of the basic ISRL in Figure 15 seems better than that in Figure 9 due to the difference in the documentation distribution in two figures. Given the same exploration strategy, when documents are randomly spread all over the network as in Figure 15, on average an exploration is more likely to succeed than when documents are clustered around a number of nodes as in Figure 9.

The performances of MP-ISRL and the random walk (randomly forward a query to k neighbors)



(a) Average number of query messages. (b) Average query success rate.

Fig. 15. Basic ISRL vs 1-thread random walks in 2000-node and 5000-node networks with average degree 5 (dynamic network scenario).



(a) Average number of query messages. (b) Average query success rate. (c) Average number of documents found.

Fig. 16. MP-ISRL vs. Random walk in 2000-node and 5000-node networks with average degree 5 (dynamic network scenario).

in the dynamic networks are plotted in Figure 16. Path cost is used to evaluate the quality of discovered paths. MP-ISRL explores paths using fine tuning with parameters (p_q, μ) being $(0.05, 0.01)$. The maximum TTL value is 5 and k varies from 1 to 5. Clearly, in dynamic networks MP-ISRL still resolves more queries successfully and incurs less number of query messages than the random walk. This is evident in both 2000-node and 5000-node networks. An interesting observation is that MP-ISRL finds more documents than the random walk in 5000-node network with all k values and 2000-node networks with $k \leq 4$. But MP-ISRL locates less documents than the random walk in 2000-node networks with k being 5. This is because the random walk can reach almost any node with a very large k (the same as the average node degree in the simulation). MP-ISRL tries to directly optimize the average path length, not the average number of desired documents. In addition, MP-ISRL only forwards queries along the discovered paths after it converges. There may be less than k discovered paths for some query vectors.

VIII. Conclusion

In this paper, we proposed a searching scheme in unstructured P2Ps called ISRL (Intelligent Search by Reinforcement Learning). It systematically learns the best path to reach desired files and adjusts itself in a dynamic network. Three design models are presented, the basic ISRL, MP-ISRL and C-ISRL. We discussed important design issues such as balancing path exploration and exploitation and offered solutions. Simulation results show that the basic ISRL achieves a higher query success rate with a lighter message load than the 1-walker random walk and the algorithm that always utilizes the first discovered hint. MP-ISRL can deliver more queries successfully at a lower cost than the random walk. C-ISRL outperforms the basic ISRL with LRU being the cache replacement policy at different cache sizes. In the future, we will conduct in-depth simulations of the extensions described in the discussion section.

References

[1] X. Li and J. Wu, "Searching techniques in peer-to-peer networks," in *Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, Edited by J. Wu*. Auerbach Publications, 2006.

[2] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. of the 16th ACM International Conference on Supercomputing (ICS'02)*, 2002.

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

[4] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *Proc. of the 11th ACM Conference on Information and Knowledge Management (CIKM'02)*, 2002.

[5] T. M. Mitchell, *Machine learning*. WCB / McGraw-Hill, 1997.

[6] L. Gatani, G. L. Re, A. Urso, and S. Gaglio, "Reinforcement learning for P2P searching," in *Proc. of the International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, 2005.

[7] "Gnutella," http://www.gnutella.com.

[8] "Ultrapeers: Another step towards gnutella scalability," http://www.limewire.com.

[9] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Ditributed caching and adaptive search in multilayer P2P networks," in *Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

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

[11] Y. Liu, L. Xiao, X. Liu, L. M. Ni, and X. Zhang, "Location awareness in unstructured peer-to-peer systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 163–174, Feb. 2005.

[12] L. Xiao, Y. Liu, and L. M. Ni, "Improving unstructured peer-to-peer systems by adaptive connection establishment," *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1091–1103, Sept. 2005.

[13] I. Jawhar and J. Wu, "A two-level random walk search protocol for peer-to-peer networks," in *Proc. of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, 2004.

[14] C. Yang, J. Wu, and X. Li, "Dominating-set-based searching in peer-to-peer networks," *International Journal of High Performance Computing and Networking*, vol. 3, no. 4, 2005.

[15] B. Yang and H. Garcia-Molina, "Designing a super-peer network," in *Proc. of the 19th International Conference on Data Engineering*, 2003.

[16] X. Li and J. Wu, "Cluster-based intelligent search in unstructured P2P networks," in *Proc. of 2005 IEEE International Workshop on Mobile and Distributed Computing (MDC'05)*, 2005.

[17] D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search in peer-to-peer networks," in *Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[18] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proc. of ACM SIGCOMM'03*, 2003.

[19] V. Vishnumurthy and P. Francis, "On random node selection in p2p and overlay networks," in *Proc. of INFOCOM'06*, 2006.

[20] T. Fei, S. Tao, L. Gao, and R. Guerin, "How to select a good alternative path in large peer-to-peer systems," in *Proc. of INFOCOM'06*, 2006.

[21] X. Li and J. Wu, "A hybrid searching scheme in unstructured P2P networks," in *Proc. of 2005 International Conference on Parallel Processing (ICPP'05)*, 2005.

[22] A. Kumar, J. Xu, and E. W. Zegura, "Efficient and scalable query routing for unstructured peer-to-peer networks," in *Proc. of IEEE INFOCOM'05*, 2005.

[23] M. Bawa, G. Manku, and P. Raghavan, "Sets: Search enhanced by topic segmentation," in *Proc. of the 26th Annual International ACM SIGIR Conference*, 2003.

[24] Y. Zhu and Y. Hu, "Ess: Efficient semantic search on gnutella-like P2P systems," in *Technical Report, Department of ECECS, University of Cincinnati*, 2004.

[25] P. Golle, K. Leyton-Brown, and I. Mironov, "Incentives for sharing in peer-to-peer networks," in *Proc. of the ACM Electronic Commerce'01*, 2001.

[26] M. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. Society for Industrial and Applied Mathematics (SIAM), 1999.

[27] H. Schtze and C. Silverstein, "Projections for efficient document clustering," in *Proc. of ACM SIGIR'97*, 1997.

[28] Y. Zhu, X. Yang, and Y. Hu, "Making search efficient on gnutella-like p2p systems," in *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[29] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[30] A. Moore and C. Atkeson, "Prioritized sweeping: reenforcement learning with less data and less real time," *Machine learning*, vol. 13, 1993.

[31] "Trec dataset," http://www.trec.org.

[32] N. Daswani and H. Garcia-Molina, "Pong cache poisoning in guess," in *Technical report, Stanford University*, 2003.