A Reinforcement Learning based Distributed Search Algorithm For Hierarchical Peer-to-Peer Information Retrieval Systems

Haizheng Zhang
College of Information Science and Technology
Pennsylvania State University
University Park, PA 16803
hzhang@ist.psu.edu

Victor Lesser
Department of Computer Science
University Of Massachusetts
Amherst, MA 01003
lesser@cs.umass.edu

ABSTRACT

The dominant existing routing strategies employed in peer-to-peer(P2P) based information retrieval(IR) systems are similarity-based approaches. In these approaches, agents depend on the content similarity between incoming queries and their direct neighboring agents to direct the distributed search sessions. However, such a heuristic is myopic in that the neighboring agents may not be connected to more relevant agents. In this paper, an online reinforcement-learning based approach is developed to take advantage of the dynamic run-time characteristics of P2P IR systems as represented by information about past search sessions. Specifically, agents maintain estimates on the downstream agents' abilities to provide relevant documents for incoming queries. These estimates are updated gradually by learning from the feedback information returned from previous search sessions. Based on this information, the agents derive corresponding routing policies. Thereafter, these agents route the queries based on the learned policies and update the estimates based on the new routing policies. Experimental results demonstrate that the learning algorithm improves considerably the routing performance on two test collection sets that have been used in a variety of distributed IR studies.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Performance, Experimentation

Keywords

peer-to-peer information retrieval, multi-agent learning, distributed search control

AAMAS'07, May 14–18, 2007, Honolulu, Hawai'i, USA.

1. INTRODUCTION

Over the last few years there have been increasing interests in studying how to control the search processes in peer-to-peer(P2P) based information retrieval(IR) systems [6, 13, 14, 15]. In this line of research, one of the core problems that concerns researchers is to efficiently route user queries in the network to agents that are in possession of appropriate documents. In the absence of global information, the dominant strategies in addressing this problem are content-similarity based approaches [6, 13, 14, 15]. While the content similarity between queries and local nodes appears to be a creditable indicator for the number of relevant documents residing on each node, these approaches are limited by a number of factors. First of all, similarity-based metrics can be myopic since locally relevant nodes may not be connected to other relevant nodes. Second, the similarity-based approaches do not take into account the run-time characteristics of the P2P IR systems, including environmental parameters, bandwidth usage, and the historical information of the past search sessions, that provide valuable information for the query routing algorithms.

In this paper, we develop a reinforcement learning based IR approach for improving the performance of distributed IR search algorithms. Agents can acquire better search strategies by collecting and analyzing feedback information from previous search sessions. Particularly, agents maintain estimates, namely *expected utility*, on the downstream agents' capabilities of providing relevant documents for specific types of incoming queries. These estimates are updated gradually by learning from the feedback information returned from previous search sessions. Based on the updated *expected utility* information, the agents derive corresponding routing policies. Thereafter, these agents route the queries based on the learned policies and update the estimates on the *expected utility* based on the new routing policies. This process is conducted in an iterative manner. The goal of the learning algorithm, even though it consumes some network bandwidth, is to shorten the routing time so that more queries are processed per time unit while at the same time finding more relevant documents. This contrasts with the content-similarity based approaches where similar operations are repeated for every incoming query and the processing time keeps largely constant over time.

Another way of viewing this paper is that our basic approach to distributed IR search is to construct a hierarchical

978-81-904262-7-5 (RPS)

overlay network(agent organization) based on the content-similarity measure among agents' document collections in a bottom-up fashion. In the past work, we have shown that this organization improves search performance significantly. However, this organizational structure does not take into account the arrival patterns of queries, including their frequency, types, and where they enter the system, nor the available communication bandwidth of the network and processing capabilities of individual agents. The intention of the reinforcement learning is to adapt the agents' routing decisions to the dynamic network situations and learn from past search sessions. Specifically, the contributions of this paper include: (1) a reinforcement learning based approach for agents to acquire satisfactory routing policies based on estimates of the potential contribution of their neighboring agents; (2) two strategies to speed up the learning process. To our best knowledge, this is one of the first reinforcement learning applications in addressing distributed content sharing problems and it is indicative of some of the issues in applying reinforcement in a complex application.

The remainder of this paper is organized as follows: Section 2 reviews the hierarchical content sharing systems and the two-phase search algorithm based on such topology. Section 3 describes a reinforcement learning based approach to direct the routing process; Section 4 details the experimental settings and analyze the results. Section 5 discusses related studies and Section 6 concludes the paper.

2. SEARCH IN HIERARCHICAL P2P IR SYSTEMS

This section briefly reviews our basic approaches to hierarchical P2P IR systems. In a hierarchical P2P IR system illustrated in Fig.1, agents are connected to each other through three types of links: upward links, downward links, and lateral links. In the following sections, we denote the set of agents that are directly connected to agent A_i as $DirectConn(A_i)$, which is defined as

$$DirectConn(A_i) = NEI(A_i) \cup PAR(A_i) \cup CHL(A_i)$$

, where $NEI(A_i)$ is the set of neighboring agents connected to A_i through lateral links; $PAR(A_i)$ is the set of agents whom agent A_i is connected to through upward links and $CHL(A_i)$ is the set of agents that agent A_i connects to through downward links. These links are established through a bottom-up content-similarity based distributed clustering process[15]. These links are then used by agents to locate other agents that contain documents relevant to the given queries.

A typical agent A_i in our system uses two queues: a local search queue, LS_i, and a message forwarding queue MF_i. The states of the two queues constitute the internal states of an agent. The local search queue LS_i stores search sessions that are scheduled for local processing. It is a priority queue and agent A_i always selects the most promising queries to process in order to maximize the global utility. MF_i consists of a set of queries to forward on and is processed in a FIFO (first in first out) fashion. For the first query in MF_i, agent A_i determines which subset of its neighboring agents to forward it to based on the agent's routing policy π_i. These routing decisions determine how the search process is conducted in the network. In this paper, we call A_i as A_j's upstream agent and A_j as A_i's downstream agent if an agent A_i routes a query to agent A_j.

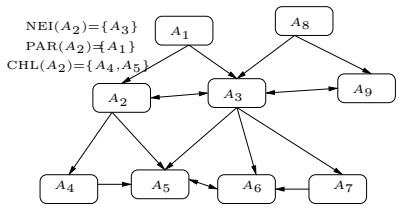

Figure 1: A fraction of a hierarchical P2PIR system

The distributed search protocol of our hierarchical agent organization is composed of two steps. In the first step, upon receipt of a query q_k at time t_l from a user, agent A_i initiates a search session s_i by probing its neighboring agents $A_j \in NEI(A_i)$ with the message $PROBE$ for the similarity value $Sim(q_k, A_j)$ between q_k and A_j. Here, A_i is defined as the query initiator of search session s_i. In the second step, A_i selects a group of the most promising agents to start the actual search process with the message $SEARCH$. These $SEARCH$ messages contain a TTL (Time To Live) parameter in addition to the query. The TTL value decreases by 1 after each hop. In the search process, agents discard those queries that either have been previously processed or whose TTL drops to 0, which prevents queries from looping in the system forever. The search session ends when all the agents that receive the query drop it or TTL decreases to 0. Upon receipt of $SEARCH$ messages for q_k, agents schedule local activities including local searching, forwarding q_k to their neighbors, and returning search results to the query initiator. This process and related algorithms are detailed in [15, 14].

3. A BASIC REINFORCEMENT LEARNING BASED SEARCH APPROACH

In the aforementioned distributed search algorithm, the routing decisions of an agent A_i rely on the similarity comparison between incoming queries and A_i's neighboring agents in order to forward those queries to relevant agents without flooding the network with unnecessary query messages. However, this heuristic is myopic because a relevant direct neighbor is not necessarily connected to other relevant agents. In this section, we propose a more general approach by framing this problem as a reinforcement learning task. In pursuit of greater flexibility, agents can switch between two modes: learning mode and non-learning mode. In the non-learning mode, agents operate in the same way as they do in the normal distributed search processes described in [14, 15]. On the other hand, in the learning mode, in parallel with distributed search sessions, agents also participate in a learning process which will be detailed in this section. Note that in the learning protocol, the learning process does not interfere with the distributed search process. Agents can choose to initiate and stop learning processes without affecting the system performance. In particular, since the learning process consumes network resources (especially bandwidth), agents can choose to initiate learning only when the network load is relatively low, thus minimizing the extra communication costs incurred by the learning algorithm.

The section is structured as follows, Section 3.1 describes

a reinforcement learning based model. Section 3.2 describes a protocol to deploy the learning algorithm in the network. Section 3.3 discusses the convergence of the learning algorithm.

3.1 The Model

An agent's routing policy takes the state of a search session as input and output the routing actions for that query. In our work, the state of a search session s_j is stipulated as:

$$QS_j = (q_k, ttl_j)$$

where ttl_j is the number of hops that remains for the search session s_j, q_k is the specific query. Q_L is an attribute of q_k that indicates which type of queries q_k most likely belong to. The set of Q_L can be generated by running a simple online classification algorithm on all the queries that have been processed by the agents, or an offline algorithm on a pre-designated training set. The assumption here is that the set of query types is learned ahead of time and belongs to the common knowledge of the agents in the network. Future work includes exploring how learning can be accomplished when this assumption does not hold. Given the query types set, an incoming query q_i can be classified to one query class $Q(q_i)$ by the formula:

$$Q(q_i) = arg\max_{Q_j} P(q_i|Q_j) \quad (1)$$

where $P(q_i|Q_j)$ indicates the likelihood that the query q_i is generated by the query class Q_j [8].

The set of atomic routing actions of an agent A_i is denoted as $\{\alpha_i\}$, where $\{\alpha_i\}$ is defined as $\alpha_i = \{\alpha_{i_0}, \alpha_{i_1}, ..., \alpha_{i_n}\}$. An element α_{i_j} represents an action to route a given query to the neighboring agent $A_{i_j} \in DirectConn(A_i)$. The routing policy π_i of agent A_i is stochastic and its outcome for a search session with state QS_j is defined as:

$$\pi_i(QS_j) = \{(\alpha_{i_0}, \pi_i(QS_i, \alpha_{i_0})), (\alpha_{i_1}, \pi_i(QS_i, \alpha_{i_1})), ...\} \quad (2)$$

Note that operator π_i is overloaded to represent either the probabilistic policy for a search session with state QS_j, denoted as $\pi_i(QS_j)$; or the probability of forwarding the query to a specific neighboring agent $A_{i_k} \in DirectConn(A_i)$ under the policy $\pi_i(QS_j)$, denoted as $\pi_i(QS_j, \alpha_{i_k})$. Therefore, equation (2) means that the probability of forwarding the search session to agent A_{i_0} is $\pi_i(QS_i, \alpha_{i_0})$ and so on. Under this stochastic policy, the routing action is non-deterministic. The advantage of such a strategy is that the best neighboring agents will not be selected repeatedly, thereby mitigating the potential "hot spots" situations.

The *expected utility*, $U_i^n(QS_j)$, is used to estimate the potential utility gain of routing query type QS_j to agent A_i under policy π_i^n. The superscript n indicates the value at the nth iteration in an iterative learning process. The *expected utility* provides routing guidance for future search sessions.

In the search process, each agent A_i maintains partial observations of its neighbors' states, as shown in Fig. 2. The partial observation includes non-local information such as the potential utility estimation of its neighbor A_m for query state QS_j, denoted as $U_m(QS_j)$, as well as the load information, L_m. These observations are updated periodically by the neighbors. The estimated utility information will be used to update A_i's expected utility for its routing policy.

Neighboring Agents	Expected Utility For Different Query Types			Load Information
	QS_0	QS_1	...	
A_0	$U_0^n(QS_0)$	$U_0^n(QS_1)$	...	L_0^n
A_1	$U_1^n(QS_0)$	$U_1^n(QS_1)$	...	L_1^n
A_2	$U_2^n(QS_0)$	$U_2^n(QS_1)$	...	L_2^n
A_3	$U_3^n(QS_0)$	$U_3^n(QS_1)$	...	L_3^n
...	...	...	...	...

Figure 2: Agent A_i's Partial Observation about its neighbors(A_0, A_1...)

The load information of A_m, L_m, is defined as

$$L_m = \frac{|MF_m|}{C_m}$$

, where $|MF_m|$ is the length of the message-forward queue and C_m is the service rate of agent A_m's message-forward queue. Therefore L_m characterizes the utilization of an agent's communication channel, and thus provide non-local information for A_m's neighbors to adjust the parameters of their routing policy to avoid inundating their downstream agents. Note that based on the characteristics of the queries entering the system and agents' capabilities, the loading of agents may not be uniform. After collecting the utilization rate information from all its neighbors, agent A_i computes $L_i^{'}$ as a single measure for assessing the average load condition of its neighborhood:

$$L_i^{'} = \frac{\sum_k L_k}{|DirectConn(A_i)|}$$

Agents exploit $L_i^{'}$ value in determining the routing probability in its routing policy.

Note that, as described in Section 3.2, information about neighboring agents is piggybacked with the query message propagated among the agents whenever possible to reduce the traffic overhead.

3.1.1 Update the Policy

An iterative update process is introduced for agents to learn a satisfactory stochastic routing policy. In this iterative process, agents update their estimates on the potential utility of their current routing policies and then propagate the updated estimates to their neighbors. Their neighbors then generate a new routing policy based on the updated observation and in turn they calculate the expected utility based on the new policies and continue this iterative process.

In particular, at time n, given a set of expected utility, an agent A_i, whose directly connected agents set is $DirectConn(A_i) = \{A_{i_0}, ..., A_{i_m}\}$, determines its corresponding stochastic routing policy for a search session of state QS_j based on the following steps:

(1) A_i first selects a subset of agents as the potential downstream agents from set $DirectConn(A_i)$, denoted as $PD_n(A_i, QS_j)$. The size of the potential downstream agent is specified as

$$|PD_n(A_i, QS_j)| = \min(|NEI(A_i), d_i^n + k)|$$

where k is a constant and is set to 3 in this paper; d_i^n, the forward width, is defined as the expected number of

neighboring agents that agent A_i can forward to at time n. This formula specifies that the potential downstream agent set $PD_n(A_i, QS_j)$ is either the subset of neighboring agents with $d_i^n + k$ highest expected utility value for state QS_j among all the agents in $DirectConn(A_i)$, or all their neighboring agents. The k is introduced based on the idea of a stochastic routing policy and it makes the forwarding probability of the $d_i^n + k$ highest agent less than 100%. Note that if we want to limit the number of downstream agents for search session s_j as 5, the probability of forwarding the query to all neighboring agents should add up to 5. Setting up d_i^n value properly can improve the utilization rate of the network bandwidth when much of the network is idle while mitigating the traffic load when the network is highly loaded. The d_i^{n+1} value is updated based on d_i^n, the previous and current observations on the traffic situation in the neighborhood. Specifically, the update formula for d_i^{n+1} is

$$d_i^{n+1} = d_i^n * (1 + \frac{1 - L_i^{'}}{|DirectConn(A_i)|})$$

In this formula, the forward width is updated based on the traffic conditions of agent A_i's neighborhood, i.e $L_i^{'}$, and its previous value.

(2) For each agent A_{i_k} in the $PD_n(A_i, QS_j)$, the probability of forwarding the query to A_{i_k} is determined in the following way in order to assign higher forwarding probability to the neighboring agents with higher expected utility value:

$$\pi_i^{n+1}(QS_j, \alpha_{i_k}) = \frac{d_i^{n+1}}{|PD_n(A_i, QS_j)|} + \beta * \big(U_{i_k}(QS_j^{'}) - \frac{PDU(A_i, QS_j)}{|PD_n(A_i, QS_j)|}\big) \quad (3)$$

where

$$PDU_n(A_i, QS_j^{'}) = \sum_{o \in PD_n(A_i, QS_j)} U_o(QS_j^{'})$$

and $QS_j^{'}$ is the subsequent state of agent A_{i_k} after agent A_i forwards the search session with state QS_j to its neighboring agent A_{i_k}; If $QS_j = (q_k, ttl_0)$, then $QS_j^{'} = (q_k, ttl_0 - 1)$.

In formula 3, the first term on the right of the equation, $\frac{d_i^{n+1}}{|PD_n(A_i, QS_j)|}$, is used to to determine the forwarding probability by equally distributing the forward width, d_i^{n+1}, to the agents in $PD_n(A_i, QS_j)$ set. The second term is used to adjust the probability of being chosen so that agents with higher expected utility values will be favored. β is determined according to:

$$\beta = \min\big(\frac{m - d_i^{n+1}}{m * u_{max} - PDU_n(A_i, QS_j^{'})}, \frac{d_i^{n+1}}{PDU_n(A_i, QS_j^{'}) - m * u_{min}}\big) \quad (4)$$

where $m = |PD_n(A_i, QS_j)|$,

$$u_{max} = \max_{o \in PD_n(A_o, QS_j)} U_o(QS_j^{'})$$

and

$$u_{min} = \min_{o \in PD_n(A_o, QS_j)} U_o(QS_j^{'})$$

This formula guarantees that the final $\pi_i^{n+1}(QS_j, \alpha_{i_k})$ value is well defined, i.e,

$$0 \le \pi_i^{n+1}(QS_j, \alpha_{i_k}) \le 1$$

and

$$\sum_i \pi_i^{n+1}(QS_j, \alpha_{i_k}) = d_i^{n+1}$$

However, such a solution does not explore all the possibilities. In order to balance between exploitation and exploration, a λ-*Greedy* approach is taken. In the λ-*Greedy* approach, in addition to assigning higher probability to those agents with higher expected utility value, as in the equation (3). Agents that appear to be "not-so-good" choices will also be sent queries based on a dynamic exploration rate.

In particular, for agents in the set $PD_n(A_i, QS_j)$, $\pi_{i_1}^{n+1}(QS_j)$ is determined in the same way as the above, with the only difference being that d_i^{n+1} is replaced with $d_i^{n+1} * (1 - \lambda_n)$. The remaining search bandwidth is used for learning by assigning probability λ_n evenly to agents A_{i_2} in the set $DirectConn(A_i) - PD_n(A_i, QS_j)$.

$$\pi_{i_2}^{n+1}(QS_j, \alpha_{i_k}) = \frac{d_i^{n+1} * \lambda_n}{|DirectConn(A_i) - PD_n(A_i, QS_j)|} \quad (5)$$

where $PD_n(A_i, QS_j) \subset DirectConn(A_i)$. Note that the exploration rate λ is not a constant and it decreases overtime. The λ is determined according to the following equation:

$$\lambda_{n+1} = \lambda_0 * e^{-c_1 n} \quad (6)$$

where λ_0 is the initial exploration rate, which is a constant; c_1 is also a constant to adjust the decreasing rate of the exploration rate; n is the current time unit.

3.1.2 Update Expected Utility

Once the routing policy at step $n+1$, π_i^{n+1}, is determined based on the above formula, agent A_i can update its own expected utility, $U_i^{n+1}(QS_i)$, based on the the updated routing policy resulted from the formula 5 and the updated U values of its neighboring agents. Under the assumption that after a query is forwarded to A_i's neighbors the subsequent search sessions are independent, the update formula is similar to the Bellman update formula in Q-Learning:

$$U_i^{n+1}(QS_j) = (1 - \theta_i) * U_i^n(QS_j) + \theta_i * (R_i^{n+1}(QS_j) + \sum_k \pi_i^{n+1}(QS_j, \alpha_{i_k}) U_k^n(QS_j^{'})) \quad (7)$$

where $QS_j^{'} = (Q_j, ttl - 1)$ is the next state of $QS_j = (Q_j, ttl)$; $R_i^{n+1}(QS_j)$ is the expected local reward for query class Q_k at agent A_i under the routing policy π_i^{n+1}; θ_i is the coefficient for deciding how much weight is given to the old value during the update process: the smaller θ_i value is, the faster the agent is expected to learn the real value, while the greater volatility of the algorithm, and vice versa. $R^{n+1}(s)$ is updated according to the following equation:

$$R_i^{n+1}(QS_j) = R_i^n(QS_j) + \gamma_i * (r(QS_j) - R_i^n(QS_j)) * P(q_j|Q_j) \quad (8)$$

where $r(QS_j)$ is the local reward associated with the search session. $P(q_j|Q_j)$ indicates how relevant the query q_j is to the query type Q_j, and γ_i is the learning rate for agent A_i. Depending on the similarity between a specific query q_i and its corresponding query type Q_i, the local reward associated with the search session has different impact on the $R_i^n(QS_j)$ estimation. In the above formula, this impact is reflected by the coefficient, the $P(q_j|Q_j)$ value.

3.1.3 Reward function

After a search session stops when its TTL values expires, all search results are returned back to the user and are compared against the relevance judgment. Assuming the set of search results is SR, the reward $Rew(SR)$ is defined as:

$$Rew(SR) = \begin{cases} 1 & \text{if } |Rel(SR)| > c \\ \frac{|Rel(SR)|}{c} & \text{otherwise.} \end{cases}$$

where SR is the set of returned search results, $Rel(SR)$ is the set of relevant documents in the search results. This equation specifies that users give 1.0 reward if the number of returned relevant documents reaches a predefined number c. Otherwise, the reward is in proportion to the number of relevant documents returned. This rationale for setting up such a cut-off value is that the importance of recall ratio decreases with the abundance of relevant documents in real world, therefore users tend to focus on only a limited number of searched results.

The details of the actual routing protocol will be introduced in Section 3.2 when we introduce how the learning algorithm is deployed in real systems.

3.2 Deployment of the Learning algorithm

This section describes how the learning algorithm can be used in either a single-phase or a two-phase search process. In the single-phase search algorithm, search sessions start from the initiators of the queries. In contrast, in the two-step search algorithm, the query initiator first attempts to seek a more appropriate starting point for the query by introducing an exploratory step as described in Section 2. Despite the difference in the quality of starting points, the major part of the learning process for the two algorithms is largely the same as described in the following paragraphs.

Before learning starts, each agent initializes the *expected utility* value for all possible states as 0. Thereafter, upon receipt of a query, in addition to the normal operations described in the previous section, an agent A_i also sets up a timer to wait for the search results returned from its downstream agents. Once the timer expires or it has received response from all its downstream agents, A_i merges and forwards the search results accrued from its downstream agents to its upstream agent. Setting up the timer speeds up the learning because agents can avoid waiting too long for the downstream agents to return search results. Note that these detailed results and corresponding agent information will still be stored at A_i until the feedback information is passed from its upstream agent and the performance of its downstream agents can be evaluated. The duration of the timer is related to the TTL value. In this paper, we set the timer to

$$t_{timer} = ttl_i * 2 + t_f$$

, where $ttl_i * 2$ is the sum of the travel time of the queries in the network, and t_f is the expected time period that users would like to wait.

The search results will eventually be returned to the search session initiator A_0. They will be compared to the relevance judgment that is provided by the final users (as described in the experiment section, the relevance judgement for the query set is provided along with the data collections). The reward will be calculated and propagated backward to the agents along the way that search results were passed. This is a reverse process of the search results propagation. In the process of propagating reward backward, agents update estimates of their own potential utility value, generate an up-to-dated policy and pass their updated results to the neighboring agents based on the algorithm described in Section 3. Upon change of expected utility value, agent A_i sends out its updated utility estimation to its neighbors so that they can act upon the changed expected utility and corresponding state. This update message includes the potential reward as well as the corresponding state $QS_i = (q_k, ttl_l)$ of agent A_i. Each neighboring agent, A_j, reacts to this kind of update message by updating the expected utility value for state $QS_j(q_k, ttl_l + 1)$ according to the newly-announced changed expected utility value. Once they complete the update, the agents would again in turn inform related neighbors to update their values. This process goes on until the TTL value in the update message increases to the TTL limit.

To speed up the learning process, while updating the expected utility values of an agent A_i's neighboring agents we specify that

$$U_m(Q_k, ttl_0) >= U_m(Q_k, ttl_1) \text{ iff } ttl_0 > ttl_1$$

Thus, when agent A_i receives an updated expected utility value with ttl_1, it also updates the expected utility values with any $ttl_0 > ttl_1$ if $U_m(Q_k, ttl_0) < U_m(Q_k, ttl_1)$ to speed up convergence. This heuristic is based on the fact that the utility of a search session is a non-decreasing function of time t.

3.3 Discussion

In formalizing the content routing system as a learning task, many assumptions are made. In real systems, these assumptions may not hold, and thus the learning algorithm may not converge. Two problems are of particular note,

(1) This content routing problem does not have Markov properties. In contrast to IP-level based packet routing, the routing decision of each agent for a particular search session s_j depends on the routing history of s_j. Therefore, the assumption that all subsequent search sessions are independent does not hold in reality. This may lead to "double counting" problem that the relevant documents of some agents will be counted more than once for the state where the TTL value is more than 1. However, in the context of the hierarchical agent organizations, two factors mitigate this problems: first, the agents in each content group form a tree-like structure. With the absense of the cycles, the estimates inside the tree would be close to the accurate value. Secondly, the stochastic nature of the routing policy partly remedies this problem.

(2) Another challenge for this learning algorithm is that in a real network environment observations on neighboring agents may not be able to be updated in time due to the communication delay or other situations. In addition, when neighboring agents update their estimates at the same time, oscillation may arise during the learning process[1].

This paper explores several approaches to speed up the learning process. Besides the aforementioned strategy of updating the expected utility values, we also employ an "active update" strategy where agents notify their neighbors whenever its *expected utility* is updated. Thus a faster convergence speed can be achieved. This strategy contrasts to the "Lazy update", where agents only echo their neighboring agents with their expected utility change when they exchange information. The trade off between the two approaches is the network load versus learning speed.

The advantage of this learning algorithm is that once a routing policy is learned, agents do not have to repeatedly compare the similarity of queries as long as the network topology remains unchanged. Instead, agent just have to determine the classification of the query properly and follow the learned policies. The disadvantage of this learning-based approach is that the learning process needs to be conducted whenever the network structure changes. There are many potential extensions for this learning model. For example, a single measure is currently used to indicate the traffic load for an agent's neighborhood. A simple extension would be to keep track of individual load for each neighbor of the agent.

4. EXPERIMENTS SETTINGS AND RESULTS

The experiments are conducted on TRANO simulation toolkit with two sets of datasets, *TREC-VLC-921* and *TREC-123-100*. The following sub-sections introduce the TRANO testbed, the datasets, and the experimental results.

4.1 TRANO Testbed

TRANO (Task Routing on Agent Network Organization) is a multi-agent based network based information retrieval testbed. TRANO is built on top of the Farm [4], a time based distributed simulator that provides a data dissemination framework for large scale distributed agent network based organizations. TRANO supports importation and exportation of agent organization profiles including topological connections and other features. Each TRANO agent is composed of an agent view structure and a control unit. In simulation, each agent is pulsed regularly and the agent checks the incoming message queues, performs local operations and then forwards messages to other agents .

4.2 Experimental Settings

In our experiment, we use two standard datasets, *TREC-VLC-921* and *TREC-123-100* datasets, to simulate the collections hosted on agents. The *TREC-VLC-921* and *TREC-123-100* datasets were created by the U.S. National Institute for Standard Technology(NIST) for its TREC conferences. In distributed information retrieval domain, the two data collections are split to 921 and 100 sub-collections. It is observed that dataset *TREC-VLC-921* is more heterogeneous than *TREC-123-100* in terms of source, document length, and relevant document distribution from the statistics of the two data collections listed in [13]. Hence, *TREC-VLC-921* is much closer to real document distributions in P2P environments. Furthermore, *TREC-123-100* is split into two sets of

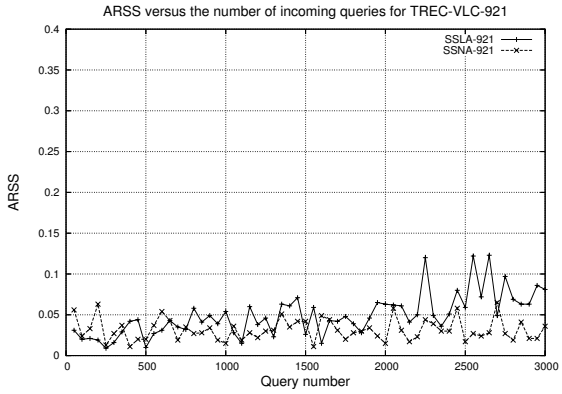

Figure 3: ARSS(Average reward per search session) versus the number of search sessions for 1phase search in *TREC-VLC-921*

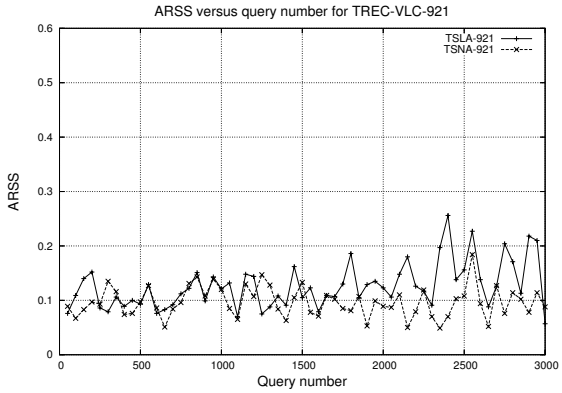

Figure 4: ARSS(Average reward per search session) versus the number of search sessions for 2phase search in *TREC-VLC-921*

sub-collections in two ways: randomly and by source. The two partitions are denoted as *TREC-123-100-Random* and *TREC-123-100-Source* respectively. The documents in each subcollection in dataset *TREC-123-100-Source* are more coherent than those in *TREC-123-100-Random*. The two different sets of partitions allow us to observe how the distributed learning algorithm is affected by the homogeneity of the collections.

The hierarchical agent organization is generated by the algorithm described in our previous algorithm [15]. During the topology generation process, degree information of each agent is estimated by the algorithm introduced by Palmer et al. [9] with parameters $\alpha = 0.5$ and $\beta = 0.6$. In our experiments, we estimate the upward limit and downward degree limit using linear discount factors 0.5, 0.8 and 1.0. Once the topology is built, queries randomly selected from the query set $301-350$ on *TREC-VLC-921* and query set $1-50$ on *TREC-123-100-Random* and *TREC-123-100-Source* are injected to the system based on a Poisson distribution

$$P(N(t) = n) = \frac{(\lambda t)^n}{n!} e^{-\lambda}$$

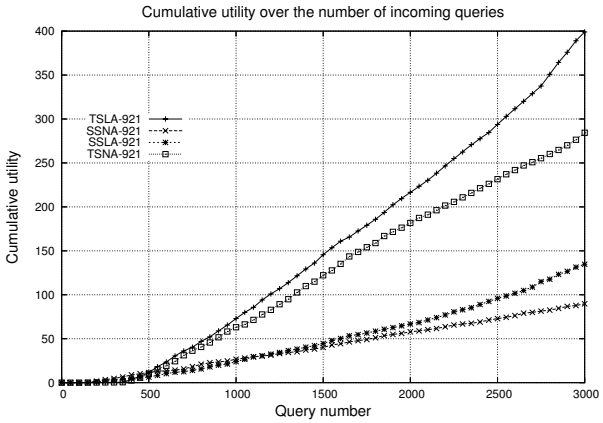

Figure 5: The cumulative utility versus the number of search sessions *TREC-VLC-921*

In addition, we assume that all agents have an equal chance of getting queries from the environment, i.e, λ is the same for every agent. In our experiments, λ is set as 0.0543 so that the mean of the incoming queries from the environment to the agent network is 50 per time unit. The service time for the communication queue and local search queue, i.e $t_{Q_{ij}}$ and t_{rs}, is set as 0.01 time unit and 0.05 time units respectively. In our experiments, there are ten types of queries acquired by clustering the query set $301 - 350$ and $1 - 50$.

4.3 Results analysis and evaluation

Figure 3 demonstrates the ARSS(Average Reward per Search Session) versus the number of incoming queries over time for the the Single-Step based Non-learning Algorithm (SSNA), and the Single-Step Learning Algorithm(SSLA) for data collection *TREC-VLC-921*. It shows that the average reward for *SSNA* algorithm ranges from $0.02 - 0.06$ and the performance changes little over time. The average reward for *SSLA* approach starts at the same level with the *SSNA* algorithm. But the performance increases over time and the average performance gain stabilizes at about 25% after query range $2000 - 3000$.

Figure 4 shows the ARSS(Average Reward per Search Session) versus the number of incoming queries over time for the the Two-Step based Non-learning Algorithm(TSNA), and the Two-Step Learning Algorithm(TSLA) for data collection *TREC-VLC-921*. The *TSNA* approach has a relatively consistent performance with the average reward ranges from $0.05 - 0.15$. The average reward for *TSLA* approach, where learning algorithm is exploited, starts at the same level with the *TSNA* algorithm and improves the average reward over time until $2000-2500$ queries joining the system. The results show that the average performance gain for *TSLA* approach over *TNLA* approach is 35% after stabilization.

Figure 5 shows the cumulative utility versus the number of incoming queries over time for *SSNA*, *SSLA*,*TSNA*, and *TSLA* respectively. It illustrates that the cumulative utility of non-learning algorithms increases largely linearly over time, while the gains of learning-based algorithms accelerate when more queries enter the system. These experimental results demonstrate that learning-based approaches consistently perform better than non-learning based routing algorithm. Moreover, two-phase learning based algorithm is better than single-phase based learning algorithm because the maximal reward an agent can receive from searching its neighborhood within TTL hops is related to the total number of the relevant documents in that area. Thus, even the optimal routing policy can do little beyond reaching these relevant documents faster. On the contrary, the two-step-based learning algorithm can relocate the search session to a neighborhood with more relevant documents. The *TSLA* combines the merits of both approaches and outperforms them.

Table 1 lists the cumulative utility for datasets *TREC-123-100-Random* and *TREC-123-100-Source* with hierarchical organizations. The five columns show the results for four different approaches. In particular, column *TSNA-Random* shows the results for dataset *TREC-123-100-Random* with the *TSNA* approach. The column *TSLA-Random* shows the results for dataset *TREC-123-100-Random* with the *TSLA* approach. There are two numbers in each cell in the column *TSLA-Random*. The first number is the actual cumulative utility while the second number is the percentage gain in terms of the utility over *TSNA* approach. Columns *TSNA-Source* and *TSLA-Source* show the results for dataset *TREC-123-100-Source* with *TSNA* and *TSLA* approaches respectively. Table 1 shows that the performance improvement for *TREC-123-100-Random* is not as significant as the other datasets. This is because that the documents in the sub-collection of *TREC-123-100-Random* are selected randomly which makes the collection model, the signature of the collection, less meaningful. Since both algorithms are designed based on the assumption that document collections can be well represented by their collection model, this result is not surprising.

Overall, Figures 4, 5, and Table 1 demonstrate that the reinforcement learning based approach can considerably enhance the system performance for both data collections. However, it remains as future work to discover the correlation between the magnitude of the performance gains and the size of the data collection and/or the extent of the heterogeneity between the sub-collections.

5. RELATED WORK

The content routing problem differs from the network-level routing in packet-switched communication networks in that content-based routing occurs in application-level networks. In addition, the destination agents in our content-routing algorithms are multiple and the addresses are not known in the routing process. IP-level Routing problems have been attacked from the reinforcement learning perspective[2, 5, 11, 12]. These studies have explored fully distributed algorithms that are able, without central coordination to disseminate knowledge about the network, to find the shortest paths robustly and efficiently in the face of changing network topologies and changing link costs. There are two major classes of adaptive, distributed packet routing algorithms in the literature: distance-vector algorithms and link-state algorithms. While this line of studies carry a certain similarity with our work, it has mainly focused on packet-switched communication networks. In this domain, the destination of a packet is deterministic and unique. Each agent maintains estimations, probabilistically or deterministically, on the distance to a certain destination through its neighbors. A variant of Q-Learning techniques is deployed

Table 1: Cumulative Utility for Datasets *TREC-123-100-Random* and *TREC-123-100-Source* with Hierarchical Organization; The percentage numbers in the columns "TSLA-Random" and "TSLA-Source" demonstrate the performance gain over the algorithm without learning

Query number	TSNA-Random	TSLA-Random	TSNA-Source	TSLA-Source
500	25.15	28.45 13%	24.00	21.05 -13%
1000	104.99	126.74 20%	93.95	96.44 2.6%
1250	149.79	168.40 12%	122.64	134.05 9.3%
1500	188.94	211.05 12%	155.30	189.60 22%
1750	235.49	261.60 11%	189.14	243.90 28%
2000	275.09	319.25 16%	219.0	278.80 26%

to update the estimations to converge to the real distances.

It has been discovered that the locality property is an important feature of information retrieval systems in user modeling studies[3]. In P2P based content sharing systems, this property is exemplified by the phenomenon that users tend to send queries that represent only a limited number of topics and conversely, users in the same neighborhood are likely to share common interests and send similar queries [10]. The learning based approach is perceived to be more beneficial for real distributed information retrieval systems which exhibit locality property. This is because the users' traffic and query patterns can reduce the state space and speed up the learning process. Related work in taking advantage of this property include [7], where the authors attempted to address this problem by user modeling techniques.

6. CONCLUSIONS

In this paper, a reinforcement-learning based approach is developed to improve the performance of distributed IR search algorithms. Particularly, agents maintain estimates, namely *expected utility*, on the downstream agents' ability to provide relevant documents for incoming queries. These estimates are updated gradually by learning from the feedback information returned from previous search sessions. Based on the updated *expected utility* information, the agents modify their routing policies. Thereafter, these agents route the queries based on the learned policies and update the estimates on the *expected utility* based on the new routing policies. The experiments on two different distributed IR datasets illustrates that the reinforcement learning approach improves considerably the cumulative utility over time.

7. REFERENCES

[1] S. Abdallah and V. Lesser. Learning the task allocation game. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 850–857, New York, NY, USA, 2006. ACM Press.

[2] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

[3] J. C. French, A. L. Powell, J. P. Callan, C. L. Viles, T. Emmitt, K. J. Prey, and Y. Mou. Comparing the performance of database selection algorithms. In *Research and Development in Information Retrieval*, pages 238–245, 1999.

[4] B. Horling, R. Mailler, and V. Lesser. Farm: A scalable environment for multi-agent development and evaluation. In *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237, Berlin, 2004. Springer-Verlag.

[5] M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications*, 1993.

[6] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *In ECIR'05*, 2005.

[7] J. Lu and J. Callan. User modeling for full-text federated search in peer-to-peer networks. In *ACM SIGIR 2006*. ACM Press, 2006.

[8] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[9] C. R. Palmer and J. G. Steffan. Generating network topologies that obey power laws. In *Proceedings of GLOBECOM '2000*, November 2000.

[10] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-topeer systems. In *INFOCOM*, 2003.

[11] D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 832–839, 1997.

[12] J. N. Tao and L. Weaver. A multi-agent, policy gradient approach to network routing. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[13] H. Zhang, W. B. Croft, B. Levine, and V. Lesser. A multi-agent approach for peer-to-peer information retrieval. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, July 2004.

[14] H. Zhang and V. Lesser. Multi-agent based peer-to-peer information retrieval systems with concurrent search sessions. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, May 2006.

[15] H. Zhang and V. R. Lesser. A dynamically formed hierarchical agent organization for a distributed content sharing system. In *2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004), 20-24 September 2004, Beijing, China*, pages 169–175. IEEE Computer Society, 2004.