Experience-Based Resource Description and Selection in Multiagent Information Retrieval

Leen-Kiat Soh

Department of Computer Science and Engineering, University of Nebraska, 115 Ferguson Hall, Lincoln, NE
Tel: (402) 472-6738 E-mail: lksoh@cse.unl.edu

Abstract

In this paper, we propose an agent-centric approach to resource description and selection in a multiagent information retrieval (IR). In the multiagent system, each agent learns from its experience through its interactions with other agents their capabilities and qualifications. Based on a distributed ontology learning framework, our methodology allows an agent to profile other agents in a dynamic translation table and a neighborhood profile, which together help determine resource description and selection process. Further, we report on the experiments and results of the first phase of our research, which focuses on the operational issues (e.g., real-time constraints, frequency of queries, number of threads, narrowness in ontology) on how the agents handle queries collaboratively.

Introduction

The multi-database model is the alternative to the single database model. As pointed out in (Callan 2000), the single database model can be successful if most of the important or valuable information on a network can be copied easily. However, due to information proprietary, costs (e.g., access, storage, management, duplication, and transmission), and distributedness of data, the multi-database information retrieval (IR) model is often times more suitable. Callan (2002) outlined three key stages of the multi-database model: (1) resource description in which the contents of each text database is described, (2) resource selection in which given an information need and a set of resource descriptions, a decision is made about which database(s) to search, and (3) result merging in which the ranked lists returned by each database are integrated into a single, coherent ranked list. Resource description is the discovery and representation of what each database contains, and is usually performed. The resource selection problem is the ranking of databases by how likely they are to satisfy the information need.

In this paper, we describe an innovative methodology based on a distributed ontology learning framework in a multiagent environment (Soh 2002a). In the multiagent system, each agent, safeguarding its database and processing queries, learns from its experience through its interactions with other agents. As a result of this learning, each agent learns the resource description of the other agents that it has come into contact, and learns the selection criteria for choosing which agents to approach to help respond to a query. The unique characteristic of our methodology is the agent treatment of resource description and selection:

• Each agent maintains a profile of other agents and thus keeps a unique set of resource descriptions. For example, agent A may think agent B is good at topic TI, but agent C may think agent B is poor at the same topic. This agent-centric viewpoint allows the system to be more adaptive to individual user’s information need and query behavior.

• Each agent is autonomous and makes decision whether to handle a query relayed by another agent based on its current status and availability. Thus, it is possible for the agent to turn down a request even though it has the data to satisfy the query. Thus, the decision making process is decentralized and localized at the agents, allowing the system to be more responsive, modular, and flexible.

In the following, we will first discuss some related work. Then, we briefly outline the overall framework of our research project. Then, we focus on the experience-based resource description and selection in a multiagent environment. Next, we describe the current phase of our project which is to understand how multiagent information retrieval is impacted by operational issues such as queries, the number of communication (negotiation) threads, the variability within the translation tables and so on. Finally, we conclude.

Related Work

Traditionally, resource descriptions can be created manually, or automatically through a unigram language model, or distributedly through a technique called query-based sampling. Manual creations (Voorhees et al. 1995, Chakravarthy and Haase 1995) might be difficult or expensive to apply in an environment with many databases (Callan 2002). The unigram language models are based on the frequencies of occurrences of keywords that occur in the databases (e.g., Callan, Lu and Croft 1995). The query-based sampling approach (Callan and Connell 2001) is a...
distributed, agent-like alternative where each resource provides cooperates by publishing resource descriptions for its document databases. The sampling requires minimal cooperation and makes no assumptions about how each provider operates internally. Our approach is similar to query-based sampling. However, our agents perform the sampling as a side effect of real-time query handling. Also, our resource description is kept dynamically. With our agent-centric viewpoint, our technique is adaptive to each agent’s experience and they may have different profiles of how well a particular agent deals with a particular topic of queries. Finally, our sampling is done whenever there is an interaction between two agents—thus the resource description changes constantly.

The major part of the resource selection problem is ranking resources by how likely they are to satisfy the information need (Callan 2000). Conventionally, the desired database ranking is one in which databases are ordered by the number of relevant documents they contain for a query (Gravano and Garcia-Molina 1995; French et al. 1998). Callan and Connell (2001) described CORI, a Bayesian inference network and an adaptation of the Okapi term frequency normalization formula, that ranks resources. In (Si et al. 2002; Xu and Croft 1999), the Kullback-Leibler (KL) divergence between the word frequency distribution of the query and the database was used to measure how well the content of the database matches with the query. Si and Callan (2003) proposed a ReDDE (Relevant Document Distribution Estimation) resource selection algorithm that explicitly tries to estimate the distribution of relevant documents across the set of available databases, considering both content similarity and database size when making its estimates. In particular, Wu and Crestani (2002) proposed a model that considers four aspects simultaneously when choosing a resource: document’s relevance to the given query, time, monetary cost, and similarity between resources. Our resource selection algorithm has several unique features: (a) it ranks the agents that safeguard the databases (or resources) instead of the database, based on the agents’ ability to satisfy a query, (b) it performs a task allocation and approaches the agents based on the ranking, and (c) it is based on an agent’s dynamic viewpoint of others that the agent maintains through experience.

**Framework**

In our original framework (Soh 2002a) for distributed ontology learning embedded in a multiagent environment, the objective is to improve communication and understanding among the agents while preserving agent autonomy. Each agent maintains a dictionary for its own experience and a translation table. The dictionary allows the agent to compare and discover relationships between a pair of words or concepts, while the translation table enables the agent to learn and record (a selected portion of) the vocabulary of its neighbors that is useful for the collaboration among the agents. The motivation for this distributed ontology learning is that each agent has its own experience and thus learns its own ontology depending on what it has been exposed to. As a result, different agents may use different words to represent the same experience. When two agents communicate, agent A may not understand agent B and that hinders collaboration. However, equipped with the distributed ontology learning capabilities, agents are able to evolve independently their own ontological knowledge while maintaining translation tables through learning to help sustain the collaborative effort. Please refer to (Soh 2002a, 2002b) for details on the design.

Our discussion here is related to (Williams and Tsatsoulis 2001) where ontology learning was conducted only between two agents via exchange of concepts (ontologies) where the agents were neither able to adapt to changes in concept definitions nor able to handle multiple assertions from different neighbors. Moreover, our framework addresses translation and interpretation of concepts, query processing and composition for collaboration among agents, and action planning based on traffic and agent activities, which indirectly control the learning rates of the agents.

The focus of the current phase of our research is on developing and analyzing the operational components of our framework, applied to a document retrieval problem. Each agent interacts with a user who submits queries based on keywords. These keywords are known as concepts in the agents. The objective of our design is to satisfy as many queries as possible and as well as possible. An agent may turn to its neighbors for help. Thus, this collaboration motivates the agents to perform distributed ontology learning to improve their performances.

**Methodology and Design**

In our design, when an agent receives a query, it checks the query against its ontology knowledge base. A query comes with a concept name and the number of documents or links desired. If the agent cannot satisfy the query, it will contact its neighbors. If the agent recognizes the concept name but does not have enough documents or links to fulfill the requirement, then it will approach its neighbors to obtain more links. If the agent does not recognize the concept name, then it passes the query to its neighbors. Every agent is equipped with \( N \) number of negotiation threads. For each contact, an agent has to activate one of these threads. So, if an agent does not have available inactive negotiation threads, it will not be able to collaborate with other agents. Hence, even if the agents do understand each other’s ontologies, it is possible that due to the query frequency and the resource constraints, the agents may not be able to utilize that understanding to help solve a query problem. When an agent obtains help from its neighbors, we say that collaboration has taken place.

**Agent Design**

As shown in Figure 1, there are nine modules. We will describe these here and further discuss the other six in he
next subsections. The Interface module interacts with the user to obtain queries and to provide queried results. Currently, we have (simulated) software users that automatically generate timed queries for our experiments. Each software user submits its queries through a socket connection with the interface.

The Query Processor module receives a query from the Interface module and processes it. It first checks the agent’s ontology base. If the query matches one of the concepts in the ontology, the module retrieves the number of links available. If the query does not find a match in the ontology, the module examines its translation table. If there are available translations, that means collaboration is possible.

The Activity Monitor module keeps track of the activities in a job vector—whether the agent is processing a query on its own, or negotiating with other neighbors for more links, or entertaining a request by a neighbor.

The Thread Manager module manages the threads of the agent. It is a low-level module that activates the threads, updates and monitors the thread activity.

There are three dynamic knowledge or databases: ontology, translation table, and profiles. The profiles keep track of the relationships between the agent and its neighbors, updating the neighborhood parameters. The ontology is a dictionary listing the concepts that the agent knows. Each concept has a list of supporting documents or links. The translation table consists of translations between each concept that the agent knows and its neighbors. Each translation is accompanied with a credibility value. In our framework, we base the credibility value between two concepts on the similarity between the two corresponding sets of documents defined by the two concepts. Table 1 shows an example of a translation table for an agent A1. In the example, A1 has four neighbors. It knows of concepts such as “basketball” and “car”. For “basketball”, there is a corresponding entry “NBA” pointing to neighbor N1, with a credibility of 2.1, N2’s “Bball” with a credibility of 1.0, and N4’s “Basketball” with a credibility of 3.4. However, it does not have a translation for “basketball” between itself and N3.

<table>
<thead>
<tr>
<th>Concepts</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>basketball</td>
<td>NBA 2.1</td>
<td>Bball 1.0</td>
<td>NIL</td>
<td>Basketball 3.4</td>
</tr>
<tr>
<td>car</td>
<td>NIL</td>
<td>Auto 2.1</td>
<td>Car 1.0</td>
<td>Move 1.0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 A translation table example.

Resource Description

As previously discussed, we describe our resources by the agents that safeguard the resources, and not just the resources themselves. In our methodology, an agent is in charge of a database (resource) and is thus responsible for interfacing with other agents and users when its database is to be queried. Thus, the utility of an agent depends on two components: operational and ontological. For the ontological component, we look at the translation table as the credibility value captures the quality of the links returned by the neighboring agents. For the operational component, we look at the relationship between the agent and each of its neighbors, through the Neighborhood Profiler. Each neighbor is profiled along four dimensions: _numHelp (the number of times the agent provides help to the neighbor), _numSuccess (the number of times the agent successfully solicits help from the neighbor), _numRequestFrom (the number of times the agent receives a request from the neighbor), and _numRequestTo (the number of times the agent initiates a request to the neighbor) (Soh and Tsatsoulis 2002a). Based on these numbers, we derive helpfulness, usefulness, importance, and reliance of each neighbor, from the viewpoint of the agent. We compute a weighted sum of all the values from both components to derive a utility measure for each agent and allocate the query demand proportionally. For example, if the user specifies that he or she desires K links for his or her query, then neighbor i with the highest utility will be requested for the highest number of links.

The Negotiation Manager module manages the negotiation tasks. In our current design, the interaction between two agents does not involve negotiations as the two simply exchange information. However, our long-term plan views negotiation as an important part of ontology interpretation and query allocation in a distributed environment. We aim to have each agent plan its own retrieval schedule to better utilize its computation. For example, suppose an agent is searching its database for a query Q1. Now, the agent receives requests from two neighbors, one for Q2, and the other for Q3. The agent may opt to perform Q2 as it shares many keywords with Q1, which the agent is currently working on, and may opt to reject the request for Q3. The rejection is provided with the above reason and is communicated back to the agent that requested help in the first place. As a result, the requesting agent will be able to maintain a resource description of the rejecting agent with a better understanding. We are currently extending our

\[\text{ontology}\]

\[\text{translation table}\]

\[\text{profiles}\]
previous work in reflective negotiations (Soh and Tsatsoulis 2002b) to distributed ontology in this framework.

**Resource Selection**

For our resource selection, it is based on the utility of a neighbor and the current status of the agent. Three modules are involved at this step: Action Planner, Collaboration Manager, and Query Composer.

The Action Planner module serves as the main reasoning component of the agent: (a) If the number of internal links satisfies the query, then the action planner simply provides those links through the Interface module to the user; (b) otherwise, if the agent recognizes the concept (i.e., it does not have any supporting documents or links for the concept) requested in the query and finds available translations, it initiates its collaborative activities; (c) if the agent does not recognize the concept, it will relay the query to another agent; and (d) finally, if there are no available translations, the link retrieval process stops and the agent reports back to the user. Whether collaboration is feasible depends on the current status of the agent, as recorded by the Activity Monitor and Thread Manager modules. If the agent does not have enough resources for collaboration, the link retrieval process terminates.

The Collaboration Manager module takes over when the action planner calls for collaboration. The objective of this module is to form an appropriate group of neighboring agents to approach and distribute the query demands (link allocations) accordingly among them. To design such a collaboration plan, this module relies on the Neighborhood Profiler module, and the translation table. Each neighbor is given a utility measure based on the translation credibility value and the relationship between the agent and the neighbor. A neighbor has a high utility if the translation credibility of the query in question is high, if the past relationship is strong, and if there is not any current interaction. The collaboration manager ranks these neighbors based on the utility measure and then assigns the query demands accordingly, with the help of the Query Composer.

The Query Composer module composes a specific query for each neighbor to be approached based on the allocation of query demands. As previously mentioned, each query is associated with a link requirement that specifies the number of links desired. A query will also include the name of the originator and a time stamp when it is first generated. If the query is based on a translation, then the translated concept name is used. If the agent does not recognize a concept and needs to relay a query it has received to a neighbor, it simply uses the queried concept directly.

**Implementation**

We have implemented all the nine modules (some albeit partially) of our agent as depicted in Figure 1 in C++. Each agent receives its user queries from a software user through a socket connection, and communicates with other agents through a central relay server module through socket connections as well. Each agent generates and maintains its neighborhood profile during runtime dynamically.

For our experiments, each agent is equipped with a translation table right from the start. Note that in our original distributed ontology framework (Soh 2002a), the entries in a translation table are learned over time based on the experience of each agent. In this paper, we focus on the operational design of collaborative understanding of distributed ontologies and assume that each agent has a translation table to begin with.

In addition, each agent is equipped with an ontology database. This database lists all the concept terms that an agent knows. For each concept, there is a list of links (or documents) that are examples that illustrate the concept. Indeed, when interpreting two concepts, we simply compare the similarities of the two lists of links supporting the two concepts. Currently, we are building this interpretation module.

**Discussion of Results**

We have performed a set of experiments, aimed at studying (a) the learning of useful neighbors for sharing queries, (b) the efficiency of query handling in different real-time scenarios and with different resource constraints, and (c) the effects of different ontological concepts and query demands on collaborative understanding. In this Section, we will describe our experimental setup and then discuss the results. For further details of our experiments and results, please refer to (Soh 2003).

**Experimental Setup**

Here is the setup of our experiments, with five agents supporting a software user each.

All agents are neighbors and can communicate among themselves. All five agents and their threads are run on the same CPU. Every agent has a unique set of nine concepts in its ontology. Each concept has five supporting links. Each agent has a translation table where each cell of the table indicates the translation between a local concept and a foreign concept in a neighbor and the translation’s credibility value. If a translation is not available, we use the symbol NIL.

Each software user has a query configuration file. Thus, instead of manually submitting these queries, the software user simply reads them from the file and sends them to the corresponding agent. For each query in a configuration file there are (a) a cycle number, (b) the queried concept name, and (c) the number of link desired. The cycle number indicates when the query will be submitted to the agent. (A cycle’s time varies as this measures a loop of activities of an agent.) Each configuration file has about 300 cycles, and two batches of exactly the same query scenarios. We want to investigate whether the agents are able to improve in their response time in the second batch after learning how to form collaborations better through neighborhood profiling. Query scenarios vary in the number of queries, “density” of queries within a time period, the degree of
demand (number of links) in the queries, and so on. Some
scenarios impose the need to collaborate on the agents;
some require the agents to process many queries within a
short time, at the same time; some require the agents to
relay the queries.

Given the above query scenarios, we further vary two
sets of parameters: the number of negotiation threads and
the credibility values in the translation tables. We vary the
number of negotiation threads between 0 and 5. When the
number is 0, the agents do not have collaborative capabili-
ties since they cannot contact other agents. When the
number is 5, an agent can simultaneously conduct 5 nego-
tiations. Thus, this number is relevant to operational con-
straints. There are also six sets of translation tables. In the
first set, all credibility values of all translations are above
zero. In this situation, every concept that one agent knows
has four translations. In the second set, one of the agents
has what term as a “narrow ontology”. That is, its transla-
tion table contains many NIL translations, above 50%. In
the third set, two agents have narrow ontologies. In the
fourth set, three agents do; in the fifth set, four agents do;
finally, all agents do. With these sets, we want to see how
successful the agents are in satisfying high-demand que-
ries. This is relevant to ontological constraints.

Given the six different numbers of negotiation threads
and six sets of translation tables, we carried out a total of
36 runs using the same set of query scenarios.

The experiments ran on a Linux platform on a 256 MB
RAM, 1.3 GHz computer.

Parameters Collected
Our experiments concentrated on two sets of parameters:
(1) Neighborhood Profile Parameters: For each neighbor,
an agent collects parameters documenting the outcomes of
their past interactions. These parameters are also used in
the computation of a neighbor’s utility measure, as de-
scribed in our Resource Description section.
(2) Query Result Parameters: For each query, an agent
collects parameters documenting the characteristics of the
query and the query outcome. Table 2 documents the defi-
nitions of these parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>numLinksDesired</td>
<td>The # of links desired by the query</td>
</tr>
<tr>
<td>numLinksRetrieved</td>
<td>The # of links retrieved at the end of the retrieval process and presented to the user, smaller than numLinksDesired</td>
</tr>
<tr>
<td>successQuality</td>
<td>numLinksRetrieved/numLinksDesired</td>
</tr>
<tr>
<td>duration</td>
<td>The actual elapsed time between the receipt of a query and the presentation of the query results to the user</td>
</tr>
</tbody>
</table>

Table 2 Query result parameters.

Results
For a detailed discussion on the results, please refer to (Soh 2003). Here we briefly report on some observations.
(1) The average _successQuality of a user’s queries in-
creases as expected when the number of threads increases.
This is because for high-demand queries that call for col-
laborations, the agent has more resources to use.
(2) The average _successQuality of a user’s queries drops
significantly whenever the corresponding agent has a nar-
row ontology. However, the drops are more significant
when the number of threads is smaller. This indicates that
link retrieval, in our application, benefits from the collabor-
ative distributed ontology design. Also, with a higher
number of negotiation threads, queries are satisfied more
successfully (high average values), and also more consis-
tently (low standard deviation values).
(4) The number of narrow ontologies does not impact the
success quality. From the operational point of view, this
was not expected. When the number of narrow ontologies
within the multiagent system increases, we expected that
more agents would relay queries to their neighbor, and that
would cause the negotiation threads to be used more fre-
quently, which would in turn cause the system to not be
able to handle subsequent queries and yield a lower suc-
cess quality. We are currently investigating the reasons
behind this observation.
(5) When the number of threads increases, it takes longer
for a query to be responded to. This observation was not
anticipated. However, upon further analysis, we realize the
following. When an agent has more threads, not only it
can approach more neighbors for help, but it also receives
more requests for help from other agents. As a result, the
agent manages more tasks and slows down its processes
for retrieving and supplying results to the software users.
This indicates an oversight in our design with regards to the
efficiency of our implementation.
(6) The multiagent system where the agents do not have
narrow ontologies have the highest average _duration
value. This is because these agents are more resourceful
and able to satisfy queries better in terms of content; and
that also costs the agents more communication and proc-
essing.
(7) An agent is able to negotiate more successfully when
the number of threads increases. This is expected since
with more threads available, an agent is able to entertain
more requests. This would help guide the design of dis-
tributed ontology learning in our work.

Conclusions
In this paper, we have described our work-in-progress
with collaborative understanding of distributed ontologies in
a multiagent framework, focusing on the operational com-
ponents. In general, we see that the number of negotiation
threads available to each agent in the system has a key role
in determining the _successQuality of a query task, the
average _successRate of a negotiation, and the degree of
collaboration among agents. We also see that the number
of “narrow” ontologies influences the agents’ behaviors
negligibly. Our current work includes (1) devising a result
merging scheme based on the response time and utility of
the agents, (2) completing the interpretation module to add
complexity into the negotiation protocols, and (3) investi-
gating the usefulness of the utility measure and its impact on the accuracy of translation.

In our design, each agent is able to learn. Currently, it learns the distributed ontologies and stores this information in its translation table. It also learns about the helpfulness and usefulness of the neighbors and captures this information in its neighborhood profile. To have a more robust system for resource description and selection, and result merging, we are developing our agents to also:

(1) Learn to recommend another agent. For example, if an agent realizes that a neighbor $N_1$ has constantly provided the best links for a query that it receives from user $U_2$, then it should inform $U_2$ to directly query $N_1$.

(2) Learn to recognize when to hand a query itself and when to relay the query. For example, if an agent thinks handling a query itself (and performing the subsequent interactions) will add to its knowledge (translation table and ontology), then it should do so.

(3) Learn to allocate query demand effectively. For example, some neighbors may be too busy to entertain query requests; some may be able to satisfy queries only opportunistically (by combining a request with another query already being processed); some may not have the processing threads. 

(4) Learn to match-make. For example, if agent $A$ always relays queries about a topic $T_1$ to agent $B$, and agent $B$ ends up getting the links from agent $C$, then agent $B$ should recognize that $A$ and $C$ should communicate directly.

(5) Learn to give up on a query. For example, if agent $A$ realizes that its neighbors are taking too much time to satisfy a query, it should learn to terminate requests to neighbors that are uncharacteristically slow.

The common thread of our learning strategies is to facilitate a dynamic, adaptive multiagent system for efficient and effective IR. In our future work, we plan to define efficiency and effectiveness in terms of recall and precision as well as response time and computation. That efficiency and effectiveness will in turn drive our agents to learn to improve their performance.

**Acknowledgments**

The author would like to thank Jing Fei Xu for her programming and running the experiments for this project.

**References**


